JYVÄSKYLÄ STUDIES IN COMPUTING
218

Mirko Myllykoski

On GPU-Accelerated Fast Direct Solvers and Their Applications in Image Denoising

JYVÄSKYLÄN YLIOPISTO

JYVÄSKYLÄ STUDIES IN COMPUTING 218

Mirko Myllykoski

On GPU-Accelerated Fast Direct Solvers and Their Applications in Image Denoising

Esitetään Jyväskylän yliopiston informaatioteknologian tiedekunnan suostumuksella
julkisesti tarkastettavaksi yliopiston Agora-rakennuksen auditoriossa 3
syyskuun 12. päivänä 2015 kello 12.

Academic dissertation to be publicly discussed, by permission of
the Faculty of Information Technology of the University of Jyväskylä,
in building Agora, auditorium 3, on September 12, 2015 at 12 o'clock noon.

UNIVERSITY OF JYVÄSKYLÄ

JYVÄSKYLÄ 2015

On GPU-Accelerated Fast Direct Solvers and Their Applications in Image Denoising

JYVÄSKYLÄ STUDIES IN COMPUTING 218

Mirko Myllykoski

On GPU-Accelerated Fast Direct Solvers and Their Applications in Image Denoising

UNIVERSITY OF JYVÄSKYLÄ

JYVÄSKYLÄ 2015

Editors
Timo Männikkö
Department of Mathematical Information Technology, University of Jyväskylä
Pekka Olsbo, Ville Korkiakangas
Publishing Unit, University Library of Jyväskylä

URN:ISBN:978-951-39-6277-7
ISBN 978-951-39-6277-7 (PDF)

ISBN 978-951-39-6276-0 (nid.)
ISSN 1456-5390

Jyväskylä University Printing House, Jyväskylä 2015

ABSTRACT

Myllykoski, Mirko
On GPU-Accelerated Fast Direct Solvers and Their Applications in Image Denoising
Jyväskylä: University of Jyväskylä, 2015, 94 p.(+included articles)
(Jyväskylä Studies in Computing
ISSN 1456-5390; 218)
ISBN 978-951-39-6276-0 (nid.)
ISBN 978-951-39-6277-7 (PDF)
Finnish summary
Diss.

This dissertation focuses on block cyclic reduction (BCR) type fast direct solvers, graphics processing unit (GPU) computation, and image denoising. The fast direct solvers are specialized methods for solving certain types of linear systems. They take into account specific characteristics of the system and are therefore able to solve the system much more efficiently than less specialized methods. In particular, this dissertation focuses on symmetric block tridiagonal linear systems that can be presented in a separable form using the so-called Kronecker matrix tensor product. Modern GPUs can provide significantly more floating point processing power than traditional central processing units (CPUs) and could therefore potentially improve the efficiency of fast direct solvers. Image denoising is a process in which a given noisy image is cleared of excess noise. Recently, higher order models that utilize mean curvature information in their regularization term have received a lot of attention. These models are expensive to solve, but fast direct solvers and GPUs could be a solution to this problem. A total of five articles are included in this article-style dissertation. The first three articles deal with the BCR methods and two present GPU implementations of different variants and compare the implementations against similar CPU implementations. The two remaining articles focus on a so-called L^1-mean curvature image denoising model. The fourth article introduces a new augmented Lagrangian-based solution algorithm and the fifth article describes an efficient GPU implementation of the algorithm. The included articles show that GPUs can provide significant performance benefits in the context of the BCR type fast direct solvers and higher order image denoising models.

Keywords: alternating direction methods of multipliers, augmented Lagrangian method, block cyclic reduction, fast direct solver, fast Poisson solver, GPU computing, image denoising, image processing, mean curvature, OpenCL, parallel computing, PSCR method, separable block tridiagonal linear system

Author	Mirko Myllykoski Department of Mathematical Information Technology University of Jyväskylä Finland E-mail: `mirko.myllykoski@jyu.fi` `mirko.myllykoski@gmail.com`
Supervisors	Professor Tuomo Rossi Department of Mathematical Information Technology University of Jyväskylä Finland Professor Jari Toivanen Department of Mathematical Information Technology University of Jyväskylä Finland
Reviewers	Professor Xue-Cheng Tai Department of Mathematics University of Bergen Norway Professor Gundolf Haase Institute for Mathematics and Scientific Computing Karl-Franzens-University Graz Austria
Opponent	Professor Olivier Pironneau Jacques-Louis Lions Laboratory Pierre-and-Marie-Curie University France

ACKNOWLEDGEMENTS

I would like to thank my supervisors Professor Tuomo Rossi and Professor Jari Toivanen. Their guidance and support has been invaluable to the success of my PhD studies and the completion of this dissertation. I would like to thank my other collaborators, Professor Roland Glowinski and Professor Tommi Kärkkäinen. It was a sincere pleasure to collaborate with them. I am grateful to the reviewers of my dissertation, Professor Xue-Cheng Tai and Professor Gundolf Haase, for their constructive comments and suggestions. I would also like to thank Professor Olivier Pironneau for being my opponent.

My research was mainly funded by the Academy of Finland (grant #252549), but additional funding was provided by the Jyväskylä Doctoral Program in Computing and Mathematical Sciences (COMAS) and the Foundation of Nokia Corporation (Nokian säätiö).

Finally, I would like to thank my family, relatives, and friends for providing support throughout my whole life. I am also grateful to my numerous co-workers for their advice and support.

Jyväskylä
August 19, 2015

Mirko Myllykoski

COMMON MATHEMATICAL NOTATIONS AND GLOSSARY

$\mathbb{K}$	$\mathbb{R}$ or $\mathbb{C}$
$\bar{\mathbb{R}}$	$\mathbb{R} \cup \{-\infty, +\infty\}$
v_{x_i}	$\partial_{x_i} v$
$\otimes$	Kronecker matrix tensor product
$\mathbf{I}_n$	$n \times n$ identity matrix
$\\|\mathbf{x}\\|$	$\\|\mathbf{x}\\|_{l^2}$ or $\\|\mathbf{x}\\|_{L^2}$
$\\|\mathbf{x}\\|_\beta$	$\sqrt{\\|\mathbf{x}\\|^2 + \beta}$
ABC	Absorbing Boundary Condition
ADMM	Alternating Direction Method of Multipliers
AL	Augmented Lagrangian
ALF	Augmented Lagrangian Functional
BCR	Block Cyclic Reduction
BV	Bounded Variation
CPU	Central Processing Unit
CR	Cyclic Reduction
CU	Computing Unit
EL	Euler-Lagrange
GMC	Generalized Mean Curvature
GPU	Graphics Processing Unit
L1MC	L^1-Mean Curvature
MC	Mean Curvature
OpenCL	Open Computing Language
PCR	Parallel Cyclic Reduction
PDE	Partial Differential Equation
PE	Processing Element
PML	Perfectly Matched Layer
PSCR	Partial Solution variant of the Cyclic Reduction
SIMD	Single Instruction, Multiple Data
SIMT	Single Instruction, Multiple Thread
SPMD	Single Program, Multiple Data
TGV	Total Generalized Variation
TV	Total Variation

LIST OF FIGURES

CONTENTS

LIST OF INCLUDED ARTICLES

PI Mirko Myllykoski, Tuomo Rossi, and Jari Toivanen. Fast Poisson solvers for graphics processing units. *In P. Manninen and P. Öster, editors, Applied Parallel and Scientific Computing, volume 7782 of Lecture Notes in Computer Science, pages 265–279*, 2013.

PII Mirko Myllykoski and Tuomo Rossi. A parallel radix-4 block cyclic reduction algorithm. *Numerical Linear Algebra with Applications, 21(4):540–556*, 2014.

PIII Mirko Myllykoski, Tuomo Rossi, and Jari Toivanen. On solving separable block tridiagonal linear systems using a GPU implementation of radix-4 PSCR method. *Submitted to SIAM Journal on Scientific Computing*.

PIV Mirko Myllykoski, Roland Glowinski, Tommi Kärkkäinen, and Tuomo Rossi. A new augmented Lagrangian approach for L^1-mean curvature image denoising. *SIAM Journal on Imaging Sciences, 8(1):95–125*, 2015.

PV Mirko Myllykoski, Roland Glowinski, Tommi Kärkkäinen, and Tuomo Rossi. A GPU-accelerated augmented Lagrangian based L^1-mean curvature image denoising algorithm implementation. *In M. Gavrilova, V. Skala, editors, WSCG 2015: 23rd International Conference in Central Europe on Computer Graphics, Visualization and Computer Vision'2015: Full Papers Proceedings, pages 119–128*, 2015.

1 INTRODUCTION

This chapter explains the background and the motivations behind the conducted research. In addition, the research questions this dissertation seeks to answer are listed and the overall structure of the dissertation is detailed. Finally, the included articles, their relationships to each other and the common themes of the dissertation, and the author's contribution to them are listed and discussed.

1.1 Background and research motivations

The three main themes of this dissertation are *block cyclic reduction* (BCR) type fast direct solvers, their *graphics processing unit* (GPU) implementations, and their applications, particularly in the field of image denoising. The *fast direct solvers* are a special subset of linear system solver algorithms. They can be used to solve special types of linear systems that arise, for example, when one wishes to numerically solve a variety of mathematical models describing the real world. The fast direct solvers are said to be fast because their arithmetical complexities are often orders of magnitude smaller than the arithmetical complexities of those methods, that do not take into account the special properties of the numerical problem in question. Memory requirements are an equally important factor, and fast direct solvers often require significantly less memory for the same reasons.

Fast Poisson solvers, which are specialized direct methods for solving the Poisson's boundary value problem, are a very typical example of the fast direct solver algorithms. They include, for example, Hockney's Fourier transformation-based method [93] and Buneman's BCR variant [43]. More generalized fast direct solvers, such as a *partial solution variant of the cyclic reduction* (PSCR) method [106, 108, 165], are also capable of solving much more complicated problems. For example, the PSCR method can be applied, with certain assumptions, to a Helmholtz equation [89] or it can be used as part of a preconditioner [143]. In particular, the PSCR method is very suitable for solving so-called separable block tridiagonal linear systems that arise as subproblems in many algorithms.

Modern GPUs are capable of providing significantly more floating-point processing power than traditional *central processing units* (CPUs). This is mainly due to GPUs' specialized parallel computing-orientated architecture and their large memory bandwidth, which are both required when processing 3D graphics. Because modern graphics engines are extremely complex and sophisticated, GPUs had to respond to the challenge by becoming more and more flexible. This led to the development of modern programmable GPUs. These devices can actually be utilized in a much wider range of applications than just 3D graphics. Because of this, several compute-intensive software programs seek to take advantage of the GPUs in their operation, and GPUs are already widely used in scientific computing as well (see, e.g., [56, 87, 86, 97, 98]).

While significant research on the overall topic of fast direct solvers and some GPU research on certain types of fast Poisson solvers (see, e.g., [31, 151]) already existed when this study began, there was not GPU-related research on more generalized methods, such as the PSCR method. This was even though GPU computing had become a relatively popular research topic during the preceding years. Part of the reason for this could have been that GPU hardware imposes a number of restrictions on the implementation. In particular, the cores inside a GPU are very limited in their functionality, and memory utilization-related limitations set many obstacles. For these reasons, traditional CPU-oriented formulations may not be suitable for GPU computation in their present form. However, for the reasons mentioned in the proceeding paragraphs, it is likely that there exists demand for effective GPU implementations of different fast direct solvers, and thus, meeting the challenge imposed by GPU hardware is a worthwhile undertaking.

Image processing provides a natural application area for the above-discussed fast direct solvers and their GPU implementations. In particular, the area of image denoising is considered in this dissertation. *Image denoising*, or more generally noise reduction, is a process in which a given noisy signal is cleared from excess noise. This process has numerous practical applications as all recording devices have some traits that make them susceptible to interference. For example, the number of photons received by any surface fluctuates in accordance with the central limit theorem. Thus, even the input signals caused by a constant radiation source are susceptible to interference, particularly when the exposure time is short. The thermal noise caused by random electrons straying from their designated paths is another common source of incorrect input signals in both analog and digital sensors. Faulty equipment may also lead to interference.

In many situations, the amount of noise must be significantly reduced before the desired information can be successfully extracted from the signal. The motivations behind removing noise from an image can be aesthetic, in which case the goal is to make the image more pleasing to the human eye, or the denoising can be a part of a preprocessing stage of a much broader image processing application, in which case the goal is to reduce the impact the noise would have on the later stages of the algorithm.

The (partial) removal of the noise can be achieved in many different ways, but quite often the task is formulated as a *multi-objective minimization problem* where the objective function tries to balance between preserving as much information as possible and keeping the amount of noise remaining in the achieved solution minimal. Solving this minimization problem often leads to a sequence of subproblems that are in some cases solvable by the fast direct solvers. A GPU-accelerated implementation would have many benefits over a conventional CPU implementation. In particular, GPU-acceleration would bring significant performance benefits if utilized properly and the image can be kept in the video memory, thus avoiding the need to transfer large quantities of data from one memory space to another in some situations.

1.2 Research questions

The objectives of this research are to modify existing fast direct solvers so that they can be implemented effectively on GPUs and to implement the modified methods using a suitable GPU programming framework. The objectives also include to develop GPU-friendly solution algorithms for higher order image denoising models. The effectiveness of the GPU implementations is demonstrated by applying them to a variety of numerical problems and comparing them against equivalent CPU implementations.

The four research questions this dissertation seeks to answer are the following:

RQ1: Can fast direct solver methods be modified such that they are better suited to GPU computation?

RQ2: Can GPUs provide significant performance benefits in the context of fast direct solvers?

RQ3: Is it possible to develop efficient GPU-friendly numerical methods for higher order image denoising models?

RQ4: Can GPUs provide significant performance benefits in the context of the aforementioned higher order models?

1.3 Structure of the work

The structure of this dissertation is as follows: In addition to the above-listed research motivations and questions, Chapter 1 lists the included articles and the author's contributions to them. Chapter 2 introduces the reader to GPU computation and the *Open computing language* (OpenCL) programming framework that was used to implement the fast direct solvers and image denoising algorithms considered in this dissertation. Chapter 3 deals with separable linear systems and BCR type fast direct solvers. Chapter 4 provides a brief summary of the *augmented Lagrangian* (AL) methods used in the image denoising algorithm considered in this dissertation. Chapter 5 provides a brief introduction to image denois-

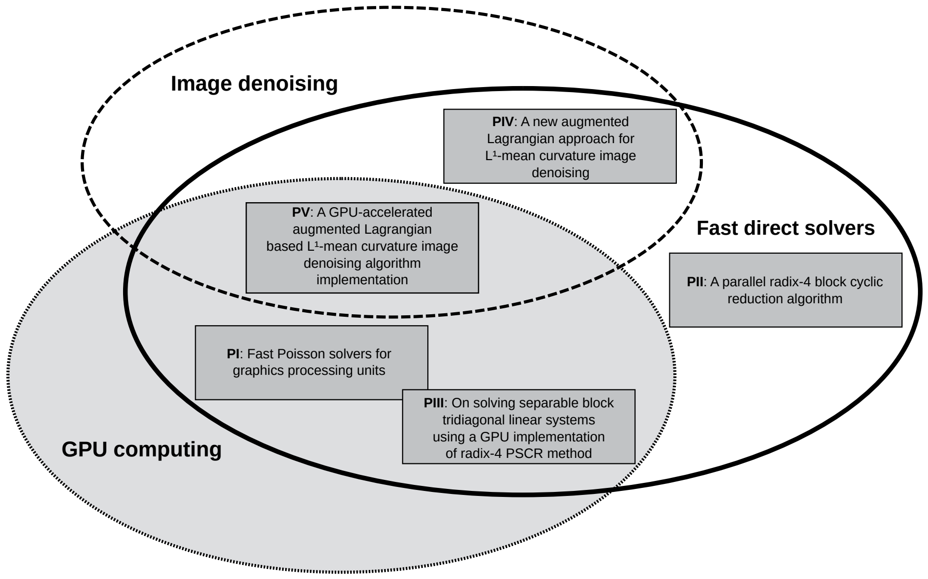


Image denoising
PIV: A new augmented Lagrangian approach for L¹-mean curvature image denoising
PV: A GPU-accelerated augmented Lagrangian based L¹-mean curvature image denoising algorithm implementation
Fast direct solvers
PII: A parallel radix-4 block cyclic reduction algorithm
PI: Fast Poisson solvers for graphics processing units
PIII: On solving separable block tridiagonal linear systems using a GPU implementation of radix-4 PSCR method
GPU computing

FIGURE 1 The included articles and how they relate to the three main themes of the dissertation.

ing methods and, in particular, to the so-called L^1-*mean curvature* (L1MC) image denoising model. Chapter 6 lists the research contributions of the included articles, and Chapter 7 provides the final conclusions. Chapter 8 offers some ideas on how to further develop the subject matter and presents some preliminary results.

1.4 Author's contributions to the included articles

A total of five articles are included in this dissertation. The author's contributions are mainly related to implementing the considered algorithms, performing comparisons between different variants and implementations, and analyzing the obtained results in various ways. However, the author has done some algorithm development in two of the included articles. The overarching theme of the dissertation is the fast direct solvers, and all five included articles are in some way related to that topic. The included article [PII] is the purest example as it is almost entirely focused on fast direct solver type algorithm development. The included articles [PI] and [PIII] bring in the GPU computing perspective and are largely implementation-oriented. The included article [PIV] introduces a suitable application area for the obtained GPU implementations in the form of an image denoising algorithm, and included article [PV] brings together all three main themes by presenting a GPU-accelerated image denoising method that utilizes a fast direct solver. The connections of the included articles with respect to the three main themes are illustrate in Figure 1.

The *first included article* [PI] focused on BCR type linear system solvers. Two (simplified) GPU implementations were presented and compared against equivalent CPU implementations. In addition, the article described a simplified *cyclic reduction* (CR)-based tridiagonal system solver and investigated the impact of increasing the so-called radix number of the algorithm. The results were presented at the Workshop on State-of-the-Art in Scientific and Parallel Computing (PARA2012) in Helsinki, Finland in the summer of 2012. A revised proceedings article was published in *Applied Parallel and Scientific Computing* (volume 7782 of Lecture Notes in Computer Science) in 2013. The author derived explicit formulas for a radix-4 BCR method; wrote the CPU and GPU implementations; performed comparisons and analyses; and wrote the majority of the article.

The *second included article* [PII] was prepared simultaneously with the first included article. The article presented an alternative way of deriving a parallel radix-4 BCR method for systems with a particular type of coefficient matrix. A GPU implementation was previously presented in [PI]. The article was published as a journal article in *Numerical Linear Algebra with Applications* in 2014 (available online since 2013). The author derived the majority of the formulas; wrote CPU implementations; performed comparisons and analyses; and wrote the majority of the article.

The *third included article* [PIII] extended the previous work presented in [PI]. The article presented a generalized GPU implementation of a radix-4 PSCR method and compared it against an equivalent CPU implementation. Initial results were presented at the SIAM Conference on Parallel Processing for Scientific Computing in Portland, Oregon, USA in early 2014. The article was submitted as a journal article to *SIAM Journal on Scientific Computing* in early 2015. The author wrote the GPU implementation; performed comparisons and analyzes; and wrote the majority of the article.

The *fourth included article* [PIV] focused on the L1MC image denoising model. The article presented an alternative AL-based solution algorithm and demonstrated the performance of the proposed algorithm by numerical means. One of the arising subproblems can be solved using the PSCR method. In addition, two subproblems can be solved point-wise without inter-process communication, which makes them very suitable for GPU computing. The article was published as a journal article in *SIAM Journal on Imaging Sciences* in 2015. The author contributed a few minor details to the actual formulation of the algorithm; wrote a CPU implementation; performed comparisons and analyses; and wrote 40–50% of the article.

The *fifth included article* [PV] is a natural follow-up to [PIV]. The article presented a GPU-accelerated version of the image denoising algorithm presented in [PIV] and compared it against an equivalent CPU implementation. The article was presented in 23rd International Conference in Central Europe on Computer Graphics, Visualization and Computer Vision 2015 (WSCG 2015) in Plzeň in the summer of 2015 and appeared in the full paper proceeding of the conference. The author wrote the GPU implementation; performed comparisons and analyzes; and wrote the majority of the article.

Remark 1.1 *The author has presented implementations and numerical results similar to those presented in the included articles [PI] and [PIII] in his Master's thesis [125].*

2 GPU COMPUTING AND OPENCL

This chapter introduces the reader to general GPU computing concepts and terminology. Some additional information on contemporary Nvidia GPUs is provided because that information is essential for the proper understanding of the GPU implementations and the numerical results presented in the included articles.

2.1 OpenCL

OpenCL [104] is an open standard for general purpose parallel programming on heterogeneous platforms. The aim is to provide software developers a portable and efficient framework which is able to tap into the colossal computing power of the modern multi-core CPUs, GPUs, and other "accelerator" devices.

2.1.1 Computing units and processing elements

The OpenCL platform model divides computer hardware into a *host* and one or more *OpenCL devices*. In most situations, the main CPU acts as the host. A OpenCL device may in turn be, for example, a GPU or a synergistic processing element found inside IBM Cell processors. The main computational tasks are performed by the OpenCL devices while the host manages and submits commands to the OpenCL devices. An OpenCL device contains one or more *computing units* (CUs) that are further divided into one or more *processing elements* (PEs). Figure 2 shows an example of an OpenCL platform that contains four OpenCL devices, each OpenCL device contains four CUs, and each CU contains 10 PEs.

On the logical level, the PEs within a CU execute instructions either together as a *single instruction, multiple data* (SIMD) unit or separately as *single program, multiple data* (SPMD) units. The former means that the PEs within a CU share the same program counter but process different data, that is, the CU behaves similarly to a vector processor. The latter means that each PE within a CU has its own

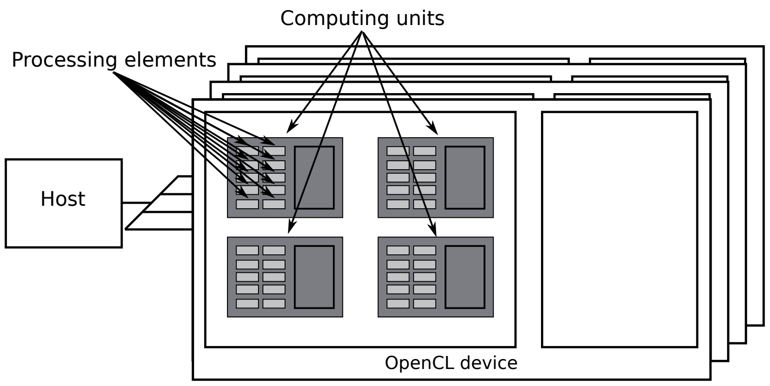


FIGURE 2 An example of an OpenCL platform.

program counter and can therefore operate independently. However, the actual hardware level implementation can be very different. For example, contemporary Nvidia GPUs implement what Nvidia calls *single instruction, multiple thread* (SIMT) architecture which is, in a certain sense, a hybrid between the SIMD and the SPMD architectures.

2.1.2 Kernels, work-groups, and work-items

An OpenCL application is executed on the host in the same way as normal applications but with the difference that the application can submit commands to the OpenCL devices using the OpenCL application programming interface. The OpenCL device-side program code execution is based around the concept of a special subroutine called (OpenCL) *kernel*. The kernels are written using a partially restricted ISO C99 programming language that can be extended with standard and platform specific extensions. An OpenCL application can place kernel launch commands and other operations into *command-queues* from which they are then automatically scheduled onto the OpenCL devices.

An OpenCL application has to define an index space for each enqueued kernel launch command. This index space is then used to launch an instance of the kernel, or a *work-item* (thread), at each point of the index space. This index space can be one- , two-, or three-dimensional and defines a unique *global index number* for each work-item. All work-items start from the beginning of the kernel program code, but the indexing allows the execution paths of different work-items to branch off. The work-items are grouped into *work-groups*, and each work-group is also given a unique index number. Each work-item is assigned a *local index number* inside the work-group. Figure 3 shows an example where each work-group (illustrated by the gray boxes) contains four work-items (illustrated by the black strings). The index pairs in the top left corners of the boxes correspond to the work-group indexes, the index pairs below the boxes correspond to the global work-item indexes, and the indexes inside the boxes correspond to the local work-item indexes.

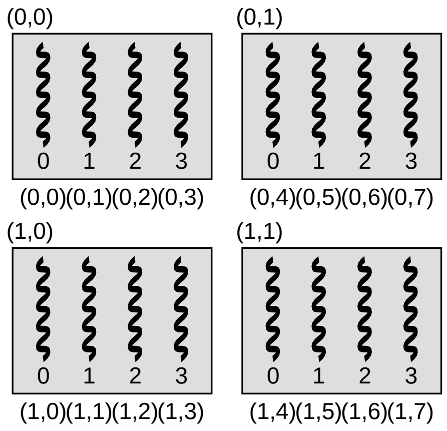


FIGURE 3 An example of how the work-items are grouped into work-groups and how they are indexed.

2.1.3 Memory hierarchy

OpenCL has a multi-level memory model. Each work-item has access to four memory regions as follows:

Global memory: A memory region to which all work-items in all work-groups have read and write access. It is usually the largest memory region of the four and is implemented off-chip on the OpenCL device. Thus, the memory bandwidth is often relatively low when compared to the other three available memory regions. The reads and writes to global memory may or may not be cached, depending on the hardware implementation.

Local memory: A memory region which is visible to a single work-group. All work-items within the work-group have read and write access. It is usually implemented as a dedicated memory region inside the CUs and consequently has much higher bandwidth than the global memory. The local memory is usually used to allocate space for variables that are shared by all work-items within the work-group.

Constant memory: A section of the global memory that remains constant during the execution of a kernel. Reads from the constant memory are often cached. Only the host has write access to the constant memory.

Private memory: A region of memory that is visible to a single work-item. It is usually mapped to a section of the global memory.

Figure 4 shows an example of an OpenCL device where the local memory is implemented as a dedicated region of memory on the CUs.

The host allocates, manages and has read and write access to global and constant memory. Local and private memory are allocated and managed automatically by the OpenCL runtime system. The host and OpenCL memory spaces are separate by default. However, they can interact through explicit copying and memory region mapping commands issued by the host.

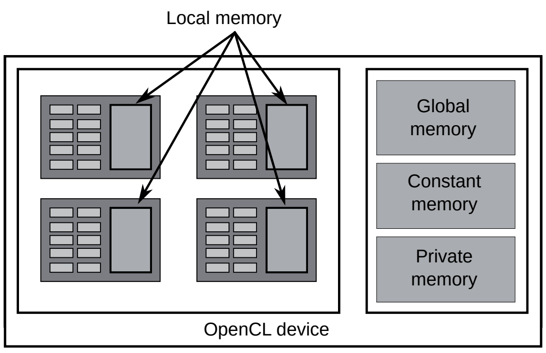


FIGURE 4 An example of an OpenCL device where the local memory is implemented as a dedicated region of memory on the CUs.

OpenCL implements a *relaxed consistency memory model*. The memory is always consistent for a single work-item, but the state of memory visible to different work-items is not guaranteed to be consistent. Memory consistency can be achieved momentarily within a work-group by using a special work-group *barrier command*. The work-items within a work-group will stall until all work-items have reached the barrier and all previous memory operations have been completed. The memory is consistent for all work-items within all work-groups right after a kernel is launched. Thus, a kernel that requires global memory consistency must be divided into multiple subkernels.

2.2 Nvidia's GPU hardware

The OpenCL specifications do not tell much about the actual underlying hardware implementation. However, programmers should be aware of these details in order to produce efficient program code. In addition, this information is essential to understand the GPU implementations and the numerical results presented in the included articles. For these reasons, this subsection provides some additional information on Nvidia's current hardware. The capabilities of a given Nvidia GPU depend on the *compute capability number* of the device. For example, a Nvidia GeForce GTX580 GPU that is a few years old has the compute capability of 2.0, while a much newer computing-orientated Nvidia Tesla K40c GPU has a compute capability of 3.5. The content of this subsection is based on Reference [128] and is general enough to be applied to most concurrent Nvidia GPUs.

Nvidia refers to CUs as *multithreaded streaming multiprocessors*. These units implement the aforementioned SIMT architecture. Each CU is capable of executing hundreds of work-items concurrently. The work-items of a single work-group execute concurrently on a single CU, and multiple work-groups can execute concurrently on a CU. Each work-group is partitioned into subgroups referred to

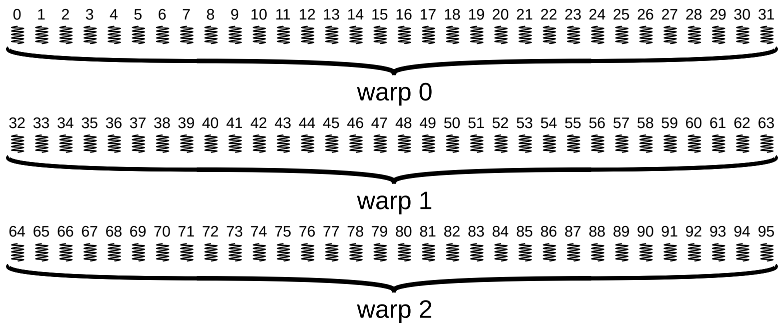


FIGURE 5 A work-group consisting of 96 work-items is divided into three warps.

as *warps*. In contemporary devices, each warp is made out of 32 work-items and contains work-items with consecutive, increasing index numbers. Figure 5 shows an example of a work-group consisting of 96 work-items and how the work-items are grouped into warps.

Each work-item within a warp has its own instruction address counter and register state, and can therefore operate independently. The stage of each warp is stored on-chip during the entire lifetime of the warp, which means that context switching has practically zero cost. A warp executes the kernel program code one common instruction at a time. During each cycle, a *warp scheduler* selects a warp that has work-items ready to execute their next common instruction and issues the instruction to those work-items. This allows twofold instruction pipelining:

– *instruction-level parallelism* within a single work-item because multiple mutually independent instructions from the same work-item can be scheduled simultaneously and
– *work-item-level parallelism* through simultaneous hardware multithreading.

Diverging execution paths are executed serially, and the work-items that do not follow the common path are disabled. Thus, full efficiency is achieved only when all 32 work-items within a warp follow the same execution path. The PEs inside the same CU share certain resources, such as a register file, special function units, load/store units, and caches. These resources are partitioned among the work-groups, which limits the number of work-groups that can reside and be processed simultaneously in a CU.

Each CU inside a compute capability 2.0 GPU contains 32 PEs, 4 special function units for single-precision floating-point transcendental functions, and 2 warp schedulers. At every instruction issue time, each warp scheduler can issue an integer or a single-precision instruction for a warp that is ready to execute. Only one warp scheduler can be active while issuing double-precision instructions. Each CU inside a compute capability 3.5 GPU contains 192 PEs, 32 special function units, and 4 warp schedulers. At every instruction issue time, each scheduler can issue two independent single-precision instructions for any warp that is ready to execute.

Figure 6 shows an example of a compute capability 2.0 CU. The CU has two active work-groups (wg0 and wg1) and each work-group consists of four warps

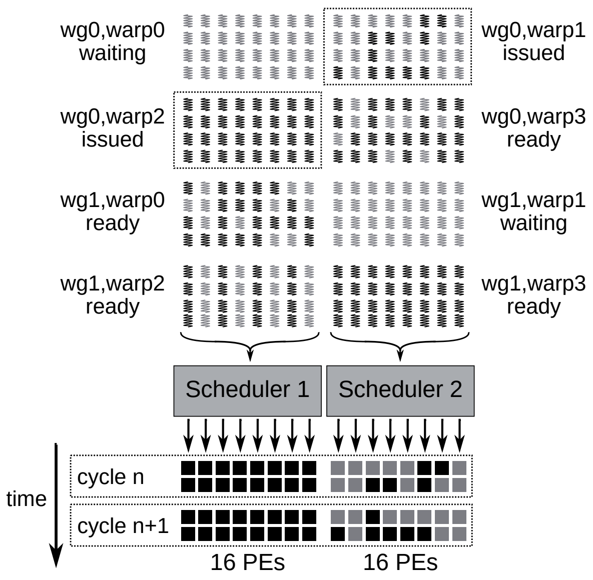


FIGURE 6 An example of how warps are scheduled on a Compute Capability 2.0 GPU.

(warp0, warp1, warp2, and warp3). The black strings correspond to the work-items that are ready to execute the next instruction, and gray strings correspond to work-items that are disabled for the current instruction issue time. Both schedulers have three warps that contain at least one work-item that is ready to execute the next instruction. The first warp scheduler has issued the warp wg0,warp2 to the first half of the PEs (illustrated by the black and gray squares), and the second warp scheduler has issued warp wg0,warp1 to the second half of the PEs. Thus, the instructions are scheduled over two clock cycles. Because the warp scheduled by the second warp scheduler contains work-items that are disabled, only 21 PEs are actually utilized during the first cycle and only 22 are actually utilized during the second cycle.

Latency, that is, the number of clock cycles it takes for a warp to be ready to execute its next instruction, has a huge impact on the performance. A full performance is achieved only when each warp scheduler has an instruction to issue for a warp at every clock cycle. In that situation, the latency is completely hidden. Memory access operations also lead to latencies, in particular when the off-chip global memory is used, because in that case usual latency times are measured in hundreds of clock cycles.

The global memory request issued by the work-items within a warp are coalesced into one or more combined memory transactions. The global memory can be accessed via 32-, 64-, or 128-byte memory transactions that must be naturally aligned by the size of the memory transaction. Scattered or misaligned memory requests are handled automatically but lead to redundant memory traffic. Compute capability 2.0 and 3.5 GPUs have level 1 and 2 caches for global memory transactions.

The local memory is divided into 32 *memory banks* organized in such a way that successive 32-bit (or 64-bit for compute capability 3.5 devices) words map to successive memory banks. Each memory bank can serve one memory request during a clock cycle, and thus, the highest local memory bandwidth is achieved when the warp accesses all memory banks simultaneously. Simultaneous memory requests that map to the same memory bank but different 32-bit (64-bit) words cause a *memory bank conflict* and are processed sequentially. Compute capability 2.0 and 3.5 GPUs support up to 48 KB of local memory per CU.

2.3 An example: a scalar sum

The following source code listing shows how a scalar sum can be effectively computed on a (Nvidia) GPU using the local memory:

```
#define THRESHOLD 16

/*
 * A function that computes the sum of the elements
 * in a vector. The number of elements in the vector
 * must be smaller than twice the size of the work-group.
 *
 * Arguments:
 *   lwork : A pointer to the vector in the local memory.
 *   size  : The vector lenghth.
 * Returns:
 *   The sum of the elements in the vector.
 */
float lsum(__local float *lwork, int size) {
    // Get the local index number.
    int local_id = get_local_id(0);

    if(size < 1)
        return 0.0f;

    // Sums that are bigger than a given threshold are
    // processed in parallel. The actual summation is
    // performed recursively in pairs.
    while(THRESHOLD < size) {

        // Calculates how many element pairs are
        // actually added together during this recursion
        // step.
        int pairs = size/2;

        // Updates the vector lenghth in preparation for
        // the next recursion step.
        size = pairs + (size & 0x1);

        // Each work-item adds togetger two elements.
        if(local_id < pairs)
            lwork[local_id] += lwork[size+local_id];

        // Syncronize using the barrier command.
        barrier(CLK_LOCAL_MEM_FENCE);
    }

    // Sums that are smaller than the given threshold are
    // processed sequentially by a single work-item.
    if(local_id == 0) {
        float sum = 0.0f;
        for(int i = 0; i < size; i++)
            sum += lwork[i];
        lwork[0] = sum;
    }

    // Syncronize using the barrier command.
    barrier(CLK_LOCAL_MEM_FENCE);

    return lwork[0];
}
```

The above implementation is very similar to the prefix sum algorithm presented in [88]. Figure 7 shows an example where the elements of a vector consisting of 35 single-precision floating-point numbers are summed together. During the first recursion step, a total of 17 element pairs are added together in parallel by 17 work-items. The results are then stored on top of the original vector without memory bank conflicts. The work-items then call the barrier synchronization subroutine in order to achieve memory consistency. The same pairwise summation is then repeated during the second recursion step by nine work-items. After this point, the size of the remaining sum is smaller than a given threshold and the remaining elements are summed by a single work-item. The reason for this is that the pairwise summation causes additional overhead that will have a significant impact if the size of the remaining sum is small.

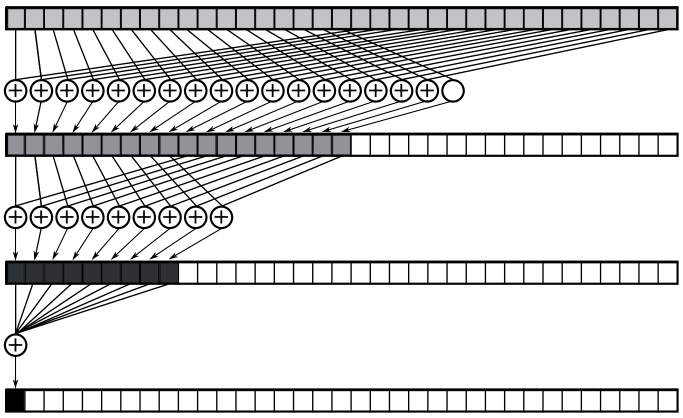


FIGURE 7 An example of how a scalar sum can be effectively computed on a GPU.

3 CYCLIC REDUCTION METHODS

This chapter serves as an introduction to separable linear systems and CR-based solvers. The chapter begins by defining the Kronecker matrix tensor product and providing a few examples of numerical problems that give rise to such linear systems. The chapter then proceeds to describe the CR method and its parallel variant. Finally, a particular radix-2 BCR method and a generalized radix-q PSCR method are presented.

3.1 Kronecker matrix tensor product

The fast direct solvers considered in this dissertation can be applied to certain types of block tridiagonal linear systems having coefficient matrices that can be presented in a separable form using the so-called Kronecker matrix tensor product:

Definition 3.1 (Kronecker matrix tensor product) *The Kronecker matrix tensor product is defined for matrices $\mathbf{B} \in \mathbb{K}^{n_1 \times m_1}$ and $\mathbf{C} \in \mathbb{K}^{n_2 \times m_2}$ as*

$$\mathbf{B} \otimes \mathbf{C} = \begin{bmatrix} b_{1,1}\mathbf{C} & b_{1,2}\mathbf{C} & \dots & b_{1,m_1}\mathbf{C} \\ b_{2,1}\mathbf{C} & b_{2,2}\mathbf{C} & \dots & b_{2,m_1}\mathbf{C} \\ \vdots & \vdots & \ddots & \vdots \\ b_{n_1,1}\mathbf{C} & b_{n_1,2}\mathbf{C} & \dots & b_{n_1,m_1}\mathbf{C} \end{bmatrix} \in \mathbb{K}^{n_1 n_2 \times m_1 m_2}.$$

In this dissertation, the field $\mathbb{K}$ can be either $\mathbb{R}$ or $\mathbb{C}$.

The product has numerous properties that will prove useful later on:

Lemma 3.1 *The Kronecker matrix tensor product is bilinear and associative, that is:*

– *Let $\mathbf{B} \in \mathbb{K}^{n_1 \times m_1}$ and $\mathbf{C}, \mathbf{D} \in \mathbb{K}^{n_2 \times m_2}$. Then,*

$$\mathbf{B} \otimes (\mathbf{C} + \mathbf{D}) = \mathbf{B} \otimes \mathbf{C} + \mathbf{B} \otimes \mathbf{D} \in \mathbb{K}^{n_1 n_2 \times m_1 m_2}.$$

– *Let* $\mathbf{B}, \mathbf{C} \in \mathbb{K}^{n_1 \times m_1}$ *and* $\mathbf{D} \in \mathbb{K}^{n_2 \times m_2}$. *Then,*

$$(\mathbf{B} + \mathbf{C}) \otimes \mathbf{D} = \mathbf{B} \otimes \mathbf{D} + \mathbf{C} \otimes \mathbf{D} \in \mathbb{K}^{n_1 n_2 \times m_1 m_2}.$$

– *Let* $\mathbf{B} \in \mathbb{K}^{n_1 \times m_1}$, $\mathbf{C} \in \mathbb{K}^{n_2 \times m_2}$, *and* $k \in \mathbb{K}$. *Then,*

$$(k\mathbf{B}) \otimes \mathbf{C} = \mathbf{B} \otimes (k\mathbf{C}) = k(\mathbf{B} \otimes \mathbf{C}) \in \mathbb{K}^{n_1 n_2 \times m_1 m_2}.$$

– *Let* $\mathbf{B} \in \mathbb{K}^{n_1 \times m_1}$, $\mathbf{C} \in \mathbb{K}^{n_2 \times m_2}$, *and* $\mathbf{D} \in \mathbb{K}^{n_3 \times m_3}$. *Then,*

$$(\mathbf{B} \otimes \mathbf{C}) \otimes \mathbf{D} = \mathbf{B} \otimes (\mathbf{C} \otimes \mathbf{D}) \in \mathbb{K}^{n_1 n_2 n_3 \times m_1 m_2 m_3}.$$

Lemma 3.2 *Let* $\mathbf{B} \in \mathbb{K}^{n_1 \times m_1}$, $\mathbf{C} \in \mathbb{K}^{n_2 \times m_2}$, $\mathbf{D} \in \mathbb{K}^{m_1 \times n_1}$ *and* $\mathbf{E} \in \mathbb{K}^{m_2 \times n_2}$. *Then,*

$$(\mathbf{B} \otimes \mathbf{C})(\mathbf{D} \otimes \mathbf{E}) = (\mathbf{BD} \otimes \mathbf{CE}) \in \mathbb{K}^{n_1 n_2 \times n_1 n_2}.$$

Lemma 3.3 *Let* $\mathbf{D} \in \mathbb{K}^{n_1 \times n_1}$ *and* $\mathbf{E} \in \mathbb{K}^{n_2 \times n_2}$ *be nonsingular matrices. Then the product matrix* $\mathbf{D} \otimes \mathbf{C} \in \mathbb{K}^{n_1 n_2 \times n_1 n_2}$ *is also nonsingular and*

$$(\mathbf{D} \otimes \mathbf{E})^{-1} = \mathbf{D}^{-1} \otimes \mathbf{E}^{-1} \in \mathbb{K}^{n_1 n_2 \times n_1 n_2}.$$

These three lemmas follow from the definition of the Kronecker matrix tensor product.

3.2 Separable linear systems

The most general form of a *separable linear system* considered in this dissertation is $\mathbf{Au} = \mathbf{f}$, with

$$\mathbf{A} = \mathbf{A}_1 \otimes \mathbf{M}_2 + \mathbf{M}_1 \otimes \mathbf{A}_2 + c(\mathbf{M}_1 \otimes \mathbf{M}_2) \in \mathbb{K}^{n_1 n_2 \times m_1 m_2}, \tag{3.1}$$

where the *factor matrices* $\mathbf{A}_1, \mathbf{M}_1 \in \mathbb{K}^{n_1 \times n_1}$ and $\mathbf{A}_2, \mathbf{M}_2 \in \mathbb{K}^{n_2 \times n_2}$ are symmetric and tridiagonal, and $c \in \mathbb{K}$. This means, that the coefficient matrix $\mathbf{A}$ is symmetric and block tridiagonal.

Separable linear systems with a coefficient matrix of the form (3.1) typically correspond to two-dimensional *partial differential equations* (PDEs). However, the methods discussed in this dissertation can also be applied to linear systems arising from three-dimensional PDEs, in which case the coefficient matrix $\mathbf{A}$ is of the form

$$\mathbf{A}_1 \otimes \mathbf{M}_2 \otimes \mathbf{M}_3 + \mathbf{M}_1 \otimes \mathbf{A}_2 \otimes \mathbf{M}_3 + \mathbf{M}_1 \otimes \mathbf{M}_2 \otimes \mathbf{A}_3 + c(\mathbf{M}_1 \otimes \mathbf{M}_2 \otimes \mathbf{M}_3), \tag{3.2}$$

where the factor matrices $\mathbf{A}_3, \mathbf{M}_3 \in \mathbb{K}^{n_3 \times n_3}$ are symmetric and tridiagonal.

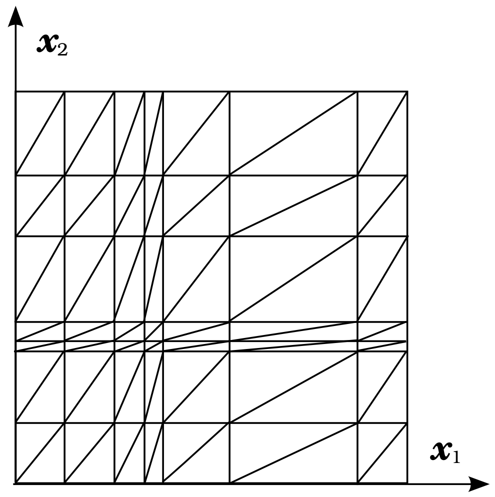


FIGURE 8 An example of an orthogonal finite element triangulation.

3.2.1 Poisson equation

Let Ω be a rectangular domain of $\mathbb{R}^d$ with $d \in \{2, 3\}$, and let $x_{j,l}$, $l = 0, 1, \dots, n_j + 1$, be the mesh points of an orthogonal finite element triangulation of Ω in the x_j-direction such that $x_{j,0}, x_{j,n_j+1} \in \partial\Omega$. Figure 8 shows an example of such a triangulation. Let us first consider a two-dimensional *Poisson equation* posed on Ω with Dirichlet boundary conditions:

$$\begin{cases} - \triangle u = f, & \text{in } \Omega, \\ u = g, & \text{on } \partial\Omega. \end{cases} \tag{3.3}$$

When the above boundary value problem is discretized using (mass lumped) linear finite elements, we get the following separable block tridiagonal linear system:

$$(\mathbf{A}_1 \otimes \mathbf{M}_2 + \mathbf{M}_1 \otimes \mathbf{A}_2)\mathbf{u} = \mathbf{f}, \tag{3.4}$$

where, for $j = 1, 2$,

$$\mathbf{A}_j = \begin{bmatrix} b_{j,1} & a_{j,1} & & \\ a_{j,1} & b_{j,2} & \ddots & \\ & \ddots & \ddots & a_{j,n_j-1} \\ & & a_{j,n_j-1} & b_{j,n_j} \end{bmatrix} \in \mathbb{R}^{n_j \times n_j} \tag{3.5}$$

and

$$\mathbf{M}_j = \begin{bmatrix} d_{j,1} & & & \\ & d_{j,2} & & \\ & & \ddots & \\ & & & d_{j,n_j} \end{bmatrix} \in \mathbb{R}^{n_j \times n_j}, \tag{3.6}$$

with

$$b_{j,l} = \frac{h_{j,l} + h_{j,l+1}}{h_{j,l}h_{j,l+1}}, \; a_{j,l} = -\frac{1}{h_{j,l}}, \; \text{and} \; d_{j,l} = \frac{h_{j,l} + h_{j,l+1}}{2}. \tag{3.7}$$

Above, $h_{j,l} = x_{j,l} - x_{j,l-1}$. An equivalent three-dimensional Poisson boundary value problem leads to

$$(\mathbf{A}_1 \otimes \mathbf{M}_2 \otimes \mathbf{M}_3 + \mathbf{M}_1 \otimes \mathbf{A}_2 \otimes \mathbf{M}_3 + \mathbf{M}_1 \otimes \mathbf{M}_2 \otimes \mathbf{A}_3)\mathbf{u} = \mathbf{f}. \tag{3.8}$$

If the discretization grid is equidistant, then the two-dimensional case (3.4) can be alternatively written as

$$(\mathbf{A} \otimes \mathbf{I}_{n_2} + \mathbf{I}_{n_1} \otimes (\mathbf{D} - 2\mathbf{I}_{n_2}))\mathbf{u} = \mathbf{f}, \tag{3.9}$$

where $\mathbf{A} = \operatorname{tridiag}\{-1, 2, -1\} \in \mathbb{R}^{n_1 \times n_1}$ and $\mathbf{D} = \operatorname{tridiag}\{-1, 4, -1\} \in \mathbb{R}^{n_2 \times n_2}$. The above translates to the following block tridiagonal linear system:

$$\begin{bmatrix} \mathbf{D} & -\mathbf{I}_{n_2} & & \\ -\mathbf{I}_{n_2} & \mathbf{D} & \ddots & \\ & \ddots & \ddots & -\mathbf{I}_{n_2} \\ & & -\mathbf{I}_{n_2} & \mathbf{D} \end{bmatrix} \begin{bmatrix} \mathbf{u}_1 \\ \mathbf{u}_2 \\ \vdots \\ \mathbf{u}_{n_1} \end{bmatrix} = \begin{bmatrix} \mathbf{f}_1 \\ \mathbf{f}_2 \\ \vdots \\ \mathbf{f}_{n_1} \end{bmatrix}. \tag{3.10}$$

In the three-dimensional case, we get

$$\begin{bmatrix} \mathbf{D} & -\mathbf{I}_{n_2n_3} & & \\ -\mathbf{I}_{n_2n_3} & \mathbf{D} & \ddots & \\ & \ddots & \ddots & -\mathbf{I}_{n_2n_3} \\ & & -\mathbf{I}_{n_2n_3} & \mathbf{D} \end{bmatrix} \begin{bmatrix} \mathbf{u}_1 \\ \mathbf{u}_2 \\ \vdots \\ \mathbf{u}_{n_1} \end{bmatrix} = \begin{bmatrix} \mathbf{f}_1 \\ \mathbf{f}_2 \\ \vdots \\ \mathbf{f}_{n_1} \end{bmatrix} \tag{3.11}$$

with

$$\mathbf{D} = \begin{bmatrix} \mathbf{B} & -\mathbf{I}_{n_3} & & \\ -\mathbf{I}_{n_3} & \mathbf{B} & \ddots & \\ & \ddots & \ddots & -\mathbf{I}_{n_3} \\ & & -\mathbf{I}_{n_3} & \mathbf{B} \end{bmatrix} \in \mathbb{R}^{n_2n_3 \times n_2n_3}, \tag{3.12}$$

where $\mathbf{B} = \operatorname{tridiag}\{-1, 6, -1\} \in \mathbb{R}^{n_3 \times n_3}$.

The methods discussed in the included articles [PI] and [PII] can be applied to the special form (3.9) assuming that $n_1 = 2^{k_1} - 1$ for some positive integer k_1. The GPU implementations discussed in the included article [PI] assume, moreover, that $n_2 = 2^{k_2} - 1$ and $n_3 = 2^{k_3} - 1$ for some positive integers k_2 and k_3. The general cases (3.4) and (3.8) can be solved using the generalized GPU implementation depicted in the included article [PIII].

3.2.2 Helmholtz equation

As an another example, consider the following approximation of a *Helmholtz equation*

$$\begin{aligned} -\triangle u - \omega^2 u &= f, \text{ in } \mathbb{R}^d, \\ \lim_{r\to\infty} r^{(d-1)/2} \left(\frac{\partial u}{\partial r} - i\omega u \right) &= 0, \end{aligned} \tag{3.13}$$

where $d \in \{2, 3\}$, ω is the wave number, and i denotes the imaginary unit. The second equation in (3.13) is the so-called *Sommerfeld radiation condition* that poses u to be a radiating solution. Before any attempts to discretize the equation are made, the unbounded domain $\mathbb{R}^d$ is usually truncated to finite a one, which means that the Sommerfeld radiation condition must somehow be approximated at the truncation boundary. Two popular techniques, which are relevant for the purposes of this dissertation, are a *perfectly matched layer* (PML) [16, 17] and an *absorbing boundary condition* (ABC) [11, 66, 74]. In both cases, discretization using bilinear (or trilinear) finite elements on an orthogonal finite-element mesh produces a separable block tridiagonal linear system [89].

A second-order ABC applied to a two-dimensional Helmholtz equation of the form (3.13) leads to

$$\begin{aligned} -\triangle u - \omega^2 u &= f, \text{ in } \Omega, \\ \nabla u \cdot \mathbf{n} - i\omega u - \frac{i}{2\omega}\nabla(\nabla u \cdot \mathbf{s}) \cdot \mathbf{s} &= 0, \text{ on the faces of } \partial\Omega, \text{ and} \\ \nabla u \cdot \mathbf{s} - \frac{3i\omega}{2} u &= 0, \text{ in the corners of } \partial\Omega, \end{aligned} \tag{3.14}$$

where Ω is the rectangular truncated domain, $\mathbf{n}$ denotes the outward unit normal to $\partial\Omega$, and $\mathbf{s}$ denotes the tangential unit vector on $\partial\Omega$. Discretization using bilinear finite elements leads to the following separable block tridiagonal linear system [89]:

$$(\mathbf{A}_1 \otimes \mathbf{M}_2 + \mathbf{M}_1 \otimes \mathbf{A}_2 - \omega^2 \mathbf{M}_1 \otimes \mathbf{M}_2)\mathbf{u} = \mathbf{f}, \tag{3.15}$$

where, for $j = 1, 2$,

$$\mathbf{A}_j = \begin{bmatrix} b_{j,1} & a_{j,1} & & \\ a_{j,1} & b_{j,2} & \ddots & \\ & \ddots & \ddots & a_{j,n_j-1} \\ & & a_{j,n_j-1} & b_{j,n_j} \end{bmatrix} \in \mathbb{C}^{n_j \times n_j} \tag{3.16}$$

and

$$\mathbf{M}_j = \begin{bmatrix} d_{j,1} & c_{j,1} & & \\ c_{j,1} & d_{j,2} & \ddots & \\ & \ddots & \ddots & c_{j,n_j-1} \\ & & c_{j,n_j-1} & d_{j,n_j} \end{bmatrix} \in \mathbb{C}^{n_j \times n_j}, \tag{3.17}$$

with

$$\begin{aligned}
b_{j,l} &= \frac{h_{j,l-1}+h_{j,l}}{h_{j,l-1}h_{j,l}},\ l = 2,3,\dots,n_j-1,\\
b_{j,1} &= \frac{1}{h_{j,1}} - \frac{i\omega}{2},\ b_{j,n_j} = \frac{1}{h_{j,n_j-1}} - \frac{i\omega}{2},\\
d_{j,l} &= \frac{h_{j,l-1}+h_{j,l}}{3},\ l = 2,3,\dots,n_j-1,\\
d_{j,1} &= \frac{h_{j,1}}{3} + \frac{i}{2\omega},\ d_{j,n_j} = \frac{h_{j,n_j-1}}{3} + \frac{i}{2\omega},\\
a_{j,l} &= -\frac{1}{h_{j,l}},\ \text{ and }\ c_{j,l} = \frac{h_{j,l}}{6}.
\end{aligned} \tag{3.18}$$

Above, $h_{j,l} = x_{j,l} - x_{j,l-1}$, where $x_{j,l}$, $l = 1,2,\dots,n_j$, are the mesh points in the x_j-direction such that $x_{j,1}, x_{j,n_j} \in \partial\Omega$. The generalized GPU implementation depicted in the included article [PIII] can be applied to the systems arising from two- and three-dimensional Helmholtz equations treated either with the PML or the ABCs.

3.3 Scalar cyclic reduction method

CR methods are are a well-known group of recursive algorithms for solving (block) tridiagonal linear systems. The basic idea is to form a sequence of subsystems decreasing in size. Usually, the system size is reduced by a factor of two in each reduction step, that is, the method's *radix number* is two. The reduced subsystems are then solved in the reverse order during the back substitution stage of the algorithm. Methods with a higher radix number are also a possibility, as will be seen later in Section 3.5.

3.3.1 Traditional formulation

Let us assume that we have a tridiagonal linear system of the form

$$\mathbf{Du} = \mathbf{f}, \tag{3.19}$$

with the coefficient matrix

$$\mathbf{D} = \begin{bmatrix} b_1 & c_1 & & \\ a_2 & b_2 & \ddots & \\ & \ddots & \ddots & c_{n-1} \\ & & a_n & b_n \end{bmatrix} \in \mathbb{K}^{n\times n}, \tag{3.20}$$

where the vectors $\mathbf{a}, \mathbf{b}, \mathbf{c} \in \mathbb{K}^n$ form a band representation for the coefficient matrix ($a_1 = 0$, $c_n = 0$). For the sake of simplicity, we assume that $n = 2^k - 1$ for some positive integer k. The method can be easily generalized for an arbitrary $n \in \mathbb{N}$.

Consider an even-numbered row and the odd-numbered rows above and below it:

$$\begin{aligned}
a_{2j-1}u_{2j-2} + b_{2j-1}u_{2j-1} + c_{2j-1}u_{2j} \qquad\qquad\qquad\qquad\qquad\qquad &= f_{2j-1} \\
a_{2j}u_{2j-1} + \quad b_{2j}u_{2j} + \quad c_{2j}u_{2j+1} \qquad\qquad\qquad &= f_{2j} \\
a_{2j+1}u_{2j} + b_{2j+1}u_{2j+1} + c_{2j+1}u_{2j+2} &= f_{2j+1} \ ,
\end{aligned}$$

where $j = 1, 2, \ldots, 2^{k-1} - 1$ and $u_0 = u_{2^k} = 0$. Let us multiply the upper odd-numbered row by $\alpha_j = -a_{2j}/b_{2j-1}$ and the lower odd-numbered row by $\beta_j = -c_{2j}/b_{2j+1}$. Now, we have

$$\begin{aligned}
\alpha_j a_{2j-1}u_{2j-2} - a_{2j}u_{2j-1} + \alpha_j c_{2j-1}u_{2j} \qquad\qquad\qquad\qquad\qquad\qquad &= \alpha_j f_{2j-1} \\
a_{2j}u_{2j-1} + \quad b_{2j}u_{2j} + c_{2j}u_{2j+1} \qquad\qquad\qquad &= f_{2j} \\
\beta_j a_{2j+1}u_{2j} - c_{2j}u_{2j+1} + \beta_j c_{2j+1}u_{2j+2} &= \beta_j f_{2j+1} \ ,
\end{aligned}$$

and can eliminate the unknowns u_{2j-1} and u_{2j+1} from the 2jth row by adding the three rows together. In the same way, we can eliminate all odd-numbered unknowns from the system. Thus, by denoting $\mathbf{a}^{(0)} = \mathbf{a}$, $\mathbf{b}^{(0)} = \mathbf{b}$, $\mathbf{c}^{(0)} = \mathbf{c}$, $\mathbf{f}^{(0)} = \mathbf{f}$, and $\mathbf{u}^{(0)} = \mathbf{u}$, we have a new tridiagonal system

$$\begin{bmatrix} b_1^{(1)} & c_1^{(1)} & & \\ a_2^{(1)} & b_2^{(1)} & \ddots & \\ & \ddots & \ddots & c_{2^{k-1}-2}^{(1)} \\ & & a_{2^{k-1}-1}^{(1)} & b_{2^{k-1}-1}^{(1)} \end{bmatrix} \begin{bmatrix} u_1^{(1)} \\ u_2^{(1)} \\ \vdots \\ u_{2^{k-1}-1}^{(1)} \end{bmatrix} = \begin{bmatrix} f_1^{(1)} \\ f_2^{(1)} \\ \vdots \\ f_{2^{k-1}-1}^{(1)} \end{bmatrix}, \tag{3.21}$$

with

$$\begin{aligned}
\alpha_j^{(1)} &= -a_{2j}^{(0)}/b_{2j-1}^{(0)}, \\
\beta_j^{(1)} &= -c_{2j}^{(0)}/b_{2j+1}^{(0)}, \\
a_j^{(1)} &= \alpha_j^{(1)} a_{2j-1}^{(0)},\ a_1^{(1)} = 0, \\
c_j^{(1)} &= \beta_j^{(1)} c_{2j+1}^{(0)},\ c_{2^{k-1}-1}^{(1)} = 0, \\
b_j^{(1)} &= b_{2j}^{(0)} + \alpha_j^{(1)} c_{2j-1}^{(0)} + \beta_j^{(1)} a_{2j+1}^{(0)}, \text{ and} \\
f_j^{(1)} &= f_{2j}^{(0)} + \alpha_j^{(1)} f_{2j-1}^{(0)} + \beta_j^{(1)} f_{2j+1}^{(0)}.
\end{aligned} \tag{3.22}$$

Above, we denote $u_j^{(1)} = u_{2j}^{(0)} = u_{2j}$. Obviously, this same *reduction* operation can be applied recursively to this new system. By taking this idea even further, we can actually form a sequence of tridiagonal subsystems. The remaining reduced

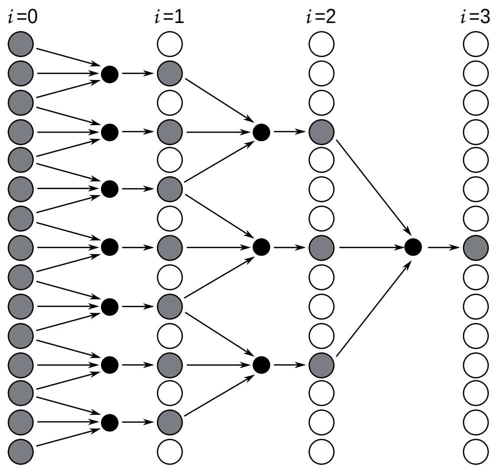


FIGURE 9 An example of how the CR method accesses the memory during the reduction stage.

systems are defined, for each reduction step $i = 2, 3, \ldots, k-1$, as

$$\begin{bmatrix} b_1^{(i)} & c_1^{(i)} & & \\ a_2^{(i)} & b_2^{(i)} & \ddots & \\ & \ddots & \ddots & c_{2^{k-i}-2}^{(i)} \\ & & a_{2^{k-i}-1}^{(i)} & b_{2^{k-i}-1}^{(i)} \end{bmatrix} \begin{bmatrix} u_1^{(i)} \\ u_2^{(i)} \\ \vdots \\ u_{2^{k-i}-1}^{(i)} \end{bmatrix} = \begin{bmatrix} f_1^{(i)} \\ f_2^{(i)} \\ \vdots \\ f_{2^{k-i}-1}^{(i)} \end{bmatrix} \tag{3.23}$$

with

$$\begin{aligned} \alpha_j^{(i)} &= -a_{2j}^{(i-1)} / b_{2j-1}^{(i-1)}, \\ \beta_j^{(i)} &= -c_{2j}^{(i-1)} / b_{2j+1}^{(i-1)}, \\ a_j^{(i)} &= \alpha_j^{(i)} a_{2j-1}^{(i-1)},\ a_1^{(i)} = 0, \\ c_j^{(i)} &= \beta_j^{(i)} c_{2j+1}^{(i-1)},\ c_{2^{k-i}-1}^{(i)} = 0, \\ b_j^{(i)} &= b_{2j}^{(i-1)} + \alpha_j^{(i)} c_{2j-1}^{(i-1)} + \beta_j^{(i)} a_{2j+1}^{(i-1)}, \text{ and} \\ f_j^{(i)} &= f_{2j}^{(i-1)} + \alpha_j^{(i)} f_{2j-1}^{(i-1)} + \beta_j^{(i)} f_{2j+1}^{(i-1)}. \end{aligned} \tag{3.24}$$

Above, we denote $u_j^{(i)} = u_{2j}^{(i-1)} = u_{2^i j}$. Figure 9 illustrates how the CR method accesses the memory during the reduction stage when the data is kept in place.

Finally, after $k-1$ reduction steps, we get

$$b_1^{(k-1)} u_1^{(k-1)} = f_1^{(k-1)} \iff u_1^{(k-1)} = f_1^{(k-1)} / b_1^{(k-1)}. \tag{3.25}$$

Because $u_j^{(i)} = u_{j/2}^{(i+1)}$, $j = 2, 4, \ldots, 2^{k-i} - 2$, and

$$u_j^{(i)} = \frac{f_j^{(i)} - a_j^{(i)} u_{j-1}^{(i)} - c_j^{(i)} u_{j+1}^{(i)}}{b_j^{(i)}},\ j = 1, 3, \ldots, 2^{k-i} - 1, \tag{3.26}$$

with $u_0^{(i)} = u_{2^{k-i}}^{(i)} = 0$, we can solve the previously generated subsystems in the reverse order by using the following *back substitution* formula

$$u_j^{(i)} = \begin{cases} \left(f_j^{(i)} - a_j^{(i)} u_{(j-1)/2}^{(i+1)} - c_j^{(i)} u_{(j+1)/2}^{(i+1)}\right) / b_j^{(i)}, & \text{when } j \notin 2\mathbb{N}, \\ u_{j/2}^{(i+1)}, & \text{when } j \in 2\mathbb{N}, \end{cases} \tag{3.27}$$

where $i = k-2, k-3, \dots, 0$, $j = 1, 2, \dots, 2^{k-i}-1$, and $u_0^{(i+1)} = u_{2^{k-i-1}}^{(i+1)} = 0$. Finally, we have $\mathbf{u} = \mathbf{D}^{-1}\mathbf{f} = \mathbf{u}^{(0)}$. The arithmetical complexity of the method is $\mathcal{O}(n)$.

The CR method offers some possibilities for parallelization because each reduction and back substitution step can be performed in parallel. For this reason, the method has been used extensively in GPU computing (see, e.g., [79, 103, 114, 150, 180]). It was also used in the included articles [PI] and [PIII]. However, each step is dependent on the preceding steps, and the number of parallel tasks is reduced by a factor of two on each reduction step, which means that the last reduction steps make use of only a fraction of the available cores. Thus, the CR method is not optimal for systems with a very large number of cores, such as GPUs. In addition, the memory access pattern disperses exponentially if the data is stored in place, as shown in Figure 9, which may be a disadvantage in some platforms. In particular, the dispersing memory access pattern causes problems with Nvidia GPUs because the scattered global memory requests cannot be processed effectively and the local memory requests placed in the same memory bank lead to memory bank conflicts.

Remark 3.1 *As noted in [72], the CR method can also be formulated using the Schur's complement. The idea is to rearrange the linear system (3.23) as*

$$\begin{bmatrix} \mathbf{A}^{(i)} & \mathbf{B}^{(i)} \\ \mathbf{C}^{(i)} & \mathbf{D}^{(i)} \end{bmatrix} \begin{bmatrix} \hat{\mathbf{u}}^{(i)} \\ \check{\mathbf{u}}^{(i)} \end{bmatrix} = \begin{bmatrix} \hat{\mathbf{f}}^{(i)} \\ \check{\mathbf{f}}^{(i)} \end{bmatrix}, \tag{3.28}$$

where $\mathbf{A}^{(i)} = \operatorname{diag}\left\{b_2^{(i)}, b_4^{(i)}, \dots, b_{2^{k-i}-2}^{(i)}\right\}$, $\mathbf{D}^{(i)} = \operatorname{diag}\left\{b_1^{(i)}, b_3^{(i)}, \dots, b_{2^{k-i}-1}^{(i)}\right\}$, $\hat{\mathbf{u}}^{(i)} = \left[u_2^{(i)}, u_4^{(i)}, \dots, u_{2^{k-i}-2}^{(i)}\right]^T$, $\check{\mathbf{u}}^{(i)} = \left[u_1^{(i)}, u_3^{(i)}, \dots, u_{2^{k-i}-1}^{(i)}\right]^T$, $\hat{\mathbf{f}}^{(i)} = \left[f_2^{(i)}, f_4^{(i)}, \dots, f_{2^{k-i}-2}^{(i)}\right]^T$, $\check{\mathbf{f}}^{(i)} = \left[f_1^{(i)}, f_3^{(i)}, \dots, f_{2^{k-i}-1}^{(i)}\right]^T$,

$$\mathbf{B}^{(i)} = \begin{bmatrix} a_2^{(i)} & c_2^{(i)} & & & \\ & a_4^{(i)} & c_4^{(i)} & & \\ & & \ddots & \ddots & \\ & & & a_{2^{k-i}-2}^{(i)} & c_{2^{k-i}-2}^{(i)} \end{bmatrix} \in \mathbb{K}^{(2^{k-i}-2)\times(2^{k-i}-1)}, \tag{3.29}$$

and

$$\mathbf{C}^{(i)} = \begin{bmatrix} c_1^{(i)} & & & \\ a_3^{(i)} & c_3^{(i)} & & \\ & \ddots & \ddots & \\ & & a_{2^{k-i}-3}^{(i)} & c_{2^{k-i}-3}^{(i)} \\ & & & a_{2^{k-i}-1}^{(i)} \end{bmatrix} \in \mathbb{K}^{(2^{k-i}-1)\times(2^{k-i}-2)}. \tag{3.30}$$

Now, each reduction step can be rewritten as

$$\begin{aligned} &\left(\mathbf{A}^{(i-1)} - \mathbf{B}^{(i-1)} \left(\mathbf{D}^{(i-1)}\right)^{-1} \mathbf{C}^{(i-1)}\right) \mathbf{u}^{(i)} = \\ &\qquad \underbrace{\hat{\mathbf{f}}^{(i-1)} - \mathbf{B}^{(i-1)} \left(\mathbf{D}^{(i-1)}\right)^{-1} \check{\mathbf{f}}^{(i-1)}}_{=\mathbf{f}^{(i)}} \end{aligned} \tag{3.31}$$

and the odd-numbered rows can be solved from

$$\mathbf{D}^{(i)} \check{\mathbf{u}}^{(i)} + \mathbf{C}^{(i)} \mathbf{u}^{(i+1)} = \check{\mathbf{f}}^{(i)}. \tag{3.32}$$

Similar Schur's complement formulations allow us to eliminate an arbitrary number of rows per reduction step, thus in principle leading to CR type methods with higher radix numbers.

3.3.2 Parallel cyclic reduction

Parallel cyclic reduction (PCR) method [95] provides a partial solution to the limitations of the CR method. Let us assume without loss of generality that we have a tridiagonal linear system of the form (3.19) with $n = 2^k$ for some positive integer k. Again, the method can be easily generalized for an arbitrary $n \in \mathbb{N}$. Consider the following three rows:

$$\begin{aligned} a_{j-1}u_{j-2} + b_{j-1}u_{j-1} + c_{j-1}u_j \qquad\qquad &= f_{j-1} \\ a_j u_{j-1} + \quad b_j u_j + \quad c_j u_{j+1} \qquad &= f_j \\ a_{j+1}u_j + b_{j+1}u_{j+1} + c_{j+1}u_{j+2} &= f_{j+1} \ , \end{aligned}$$

where $j = 1, 2, \ldots, 2^k$. The rows rows 0 and $2^k + 1$ are assumed to be zeros. As before, we can eliminate the unknowns u_{j-1} and u_{j+1} from the jth row by multiplying the row above the jth row by $\alpha_j = -a_j/b_{j-1}$ and the row below the jth row by $\beta_j = -c_j/b_{j+1}$ and then adding the three rows together. Once this reduction operation is applied to all rows, we effectively get two new tridiagonal linear systems. The first subsystem involves only the even-numbered unknowns, and the second subsystem involves only the odd-numbered unknowns.

More specifically, let $\mathbf{a}^{(0)} = \mathbf{a}$, $\mathbf{b}^{(0)} = \mathbf{b}$, $\mathbf{c}^{(0)} = \mathbf{c}$, $\mathbf{f}^{(0)} = \mathbf{f}$, and $\mathbf{u}^{(0)} = \mathbf{u}$. The two new tridiagonal subsystems are of the form

$$\begin{bmatrix} b_1^{(0,0)} & c_1^{(0,0)} & & \\ a_2^{(0,0)} & b_2^{(0,0)} & \ddots & \\ & \ddots & \ddots & c_{2^{k-1}}^{(0,0)} \\ & & a_{2^{k-1}}^{(0,0)} & b_{2^{k-1}}^{(0,0)} \end{bmatrix} \begin{bmatrix} u_1^{(0,0)} \\ u_2^{(0,0)} \\ \vdots \\ u_{2^{k-1}}^{(0,0)} \end{bmatrix} = \begin{bmatrix} f_1^{(0,0)} \\ f_2^{(0,0)} \\ \vdots \\ f_{2^{k-1}}^{(0,0)} \end{bmatrix} \tag{3.33}$$

and

$$\begin{bmatrix} b_1^{(0,1)} & c_1^{(0,1)} & & \\ a_2^{(0,1)} & b_2^{(0,1)} & \ddots & \\ & \ddots & \ddots & c_{2^{k-1}}^{(0,1)} \\ & & a_{2^{k-1}}^{(0,1)} & b_{2^{k-1}}^{(0,1)} \end{bmatrix} \begin{bmatrix} u_1^{(0,1)} \\ u_2^{(0,1)} \\ \vdots \\ u_{2^{k-1}}^{(0,1)} \end{bmatrix} = \begin{bmatrix} f_1^{(0,1)} \\ f_2^{(0,1)} \\ \vdots \\ f_{2^{k-1}}^{(0,1)} \end{bmatrix}, \tag{3.34}$$

where, for $l = 0, 1$,

$$\begin{aligned} \alpha_j^{(0,l)} &= -a_{2j-l}^{(0)}/b_{2j-l-1}^{(0)}, \\ \beta_j^{(0,l)} &= -c_{2j-l}^{(0)}/b_{2j-l+1}^{(0)}, \\ a_j^{(0,l)} &= \alpha_j^{(0,l)} a_{2j-l-1}^{(0)},\ a_1^{(0,l)} = 0, \\ c_j^{(0,l)} &= \beta_j^{(0,l)} c_{2j-l+1}^{(0)},\ c_{2^{k-1}}^{(0,l)} = 0, \\ b_j^{(0,l)} &= b_{2j-l}^{(0)} + \alpha_j^{(0,l)} c_{2j-l-1}^{(0)} + \beta_j^{(0,l)} a_{2j-l+1}^{(0)}, \text{ and} \\ f_j^{(0,l)} &= f_{2j-l}^{(0)} + \alpha_j^{(0,l)} f_{2j-l-1}^{(0)} + \beta_j^{(0,l)} f_{2j-l+1}^{(0)}. \end{aligned} \tag{3.35}$$

Above, we denote $u_j^{(0,l)} = u_{2j-l}^{(0)} = u_{2j-l}$.

Naturally, this reduction operation can be applied recursively to these two new subsystems. The remaining reduced systems are defined, for each reduction step $i = 2, 3, \ldots, k$, as

$$\begin{bmatrix} b_1^{(l)} & c_1^{(l)} & & \\ a_2^{(l)} & b_2^{(l)} & \ddots & \\ & \ddots & \ddots & c_{2^{k-i}}^{(l)} \\ & & a_{2^{k-i}}^{(l)} & b_{2^{k-i}}^{(l)} \end{bmatrix} \begin{bmatrix} u_1^{(l)} \\ u_2^{(l)} \\ \vdots \\ u_{2^{k-i}}^{(l)} \end{bmatrix} = \begin{bmatrix} f_1^{(l)} \\ f_2^{(l)} \\ \vdots \\ f_{2^{k-i}}^{(l)} \end{bmatrix}, \tag{3.36}$$

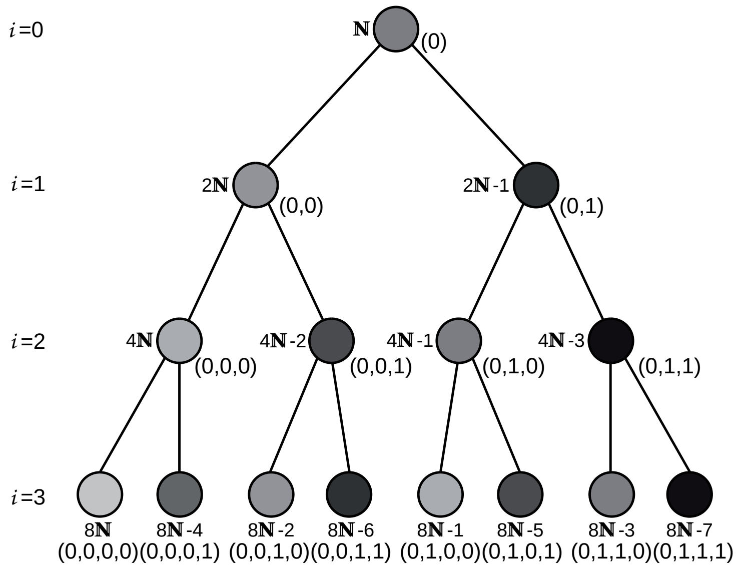


FIGURE 10 A four-level binary tree representation for the tridiagonal subsystems generated by the PCR method.

where $l = (l_0, l_1, l_2, \dots, l_i) \in \{0\} \times \{0,1\}^i$,

$$\begin{aligned}
\alpha_j^{(l_0,\dots,l_i)} &= -a_{2j-l_i}^{(l_0,\dots,l_{i-1})} / b_{2j-l_i-1}^{(l_0,\dots,l_{i-1})}, \\
\beta_j^{(l_0,\dots,l_i)} &= -c_{2j-l_i}^{(l_0,\dots,l_{i-1})} / b_{2j-l_i+1}^{(l_0,\dots,l_{i-1})}, \\
a_j^{(l_0,\dots,l_i)} &= \alpha_j^{(l_0,\dots,l_i)} a_{2j-l_i-1}^{(l_0,\dots,l_{i-1})},\ a_1^{(l_0,\dots,l_i)} = 0, \\
c_j^{(l_0,\dots,l_i)} &= \beta_j^{(l_0,\dots,l_i)} c_{2j-l_i+1}^{(l_0,\dots,l_{i-1})},\ c_{2^{k-i}}^{(l_0,\dots,l_i)} = 0, \\
b_j^{(l_0,\dots,l_i)} &= b_{2j-l_i}^{(l_0,\dots,l_{i-1})} + \alpha_j^{(l_0,\dots,l_i)} c_{2j-l_i-1}^{(l_0,\dots,l_{i-1})} + \beta_j^{(l_0,\dots,l_i)} a_{2j-l_i+1}^{(l_0,\dots,l_{i-1})}, \text{ and} \\
f_j^{(l_0,\dots,l_i)} &= f_{2j-l_i}^{(l_0,\dots,l_{i-1})} + \alpha_j^{(l_0,\dots,l_i)} f_{2j-l_i-1}^{(l_0,\dots,l_{i-1})} + \beta_j^{(l_0,\dots,l_i)} f_{2j-l_i+1}^{(l_0,\dots,l_{i-1})}.
\end{aligned} \tag{3.37}$$

Above, we denote $u_j^{(l_0,\dots,l_i)} = u_{2j-l_i}^{(l_0,\dots,l_{i-1})} = u_{L(l_0,\dots,l_i,j)}$, where

$$L : \{0\} \times \{0,1\}^i \times \mathbb{N} \to \mathbb{N},\ L(l_0,\dots,l_i,j) = 2^i j - \sum_{t=1}^{i} 2^{t-1} l_t. \tag{3.38}$$

In all subsystems, the rows rows 0 and $2^{k-i}+1$ are assumed to be zeros.

Figure 10 illustrates how the tridiagonal subsystem relate to each other and Figure 11 illustrates how the PCR method accesses the memory when the data is kept in place. The different shades in Figures 10 and 11 correspond to the different subsystems.

Finally, after k reduction steps, we get

$$\begin{aligned}
b_1^{(l)} u_1^{(l)} = f_1^{(l)} &\iff u_1^{(l)} = f_1^{(l)} / b_1^{(l)} \\
&\iff u_{L(l,1)} = f_1^{(l)} / b_1^{(l)},
\end{aligned} \tag{3.39}$$

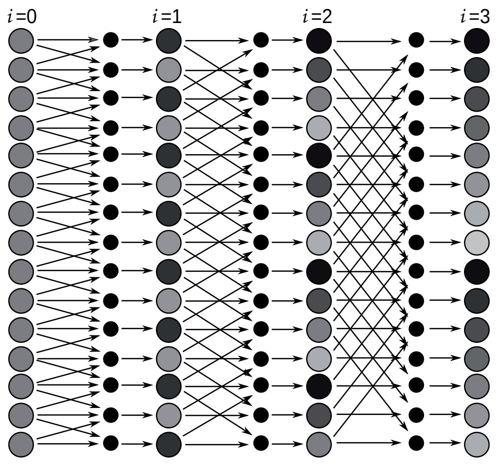


FIGURE 11 An example of how the PCR method accesses the memory when the data is kept in place.

for all $l \in \{0\} \times \{0,1\}^k$. Note that the PCR method does not involve a back substitution stage.

Contrary to the CR method, the number of parallel tasks stays constant as the method progresses. In addition, the data can be stored in place without memory access pattern dispersion. Thus, the PCR method is more suitable for multicore architectures and, in particular, to GPUs. Unfortunately the arithmetical complexity of the PCR method is $\mathcal{O}(n \log n)$, which will have a significant impact on large linear systems. However, the PCR method property of splitting the system into multiple independent subsystems has been exploited in GPU computation (see, e.g., [57, 105, 180]). The basic idea is to take a few PCR reduction steps and then solve the arising subsystems in parallel using the *Thomas method* [55]. The Thomas method is basically a special variant of the well-known LU-decomposition method for tridiagonal matrices with $\mathcal{O}(n)$ complexity. A similar hybrid approach was also used in the included article [PIII].

Remark 3.2 *As noted in [72], the Schur's complement idea discussed in Remark 3.1 can be extended to the PCR method as well. For example, the first reduction step can be rewritten as*

$$\begin{aligned}
\left(\mathbf{A}^{(0)} - \mathbf{B}^{(0)} \left(\mathbf{D}^{(0)}\right)^{-1} \mathbf{C}^{(0)}\right) \mathbf{u}^{(0,0)} &= \underbrace{\hat{\mathbf{f}}^{(0)} - \mathbf{B}^{(0)} \left(\mathbf{D}^{(0)}\right)^{-1} \check{\mathbf{f}}^{(0)}}_{=\mathbf{f}^{(0,0)}} \\
\left(\mathbf{D}^{(0)} - \mathbf{C}^{(0)} \left(\mathbf{A}^{(0)}\right)^{-1} \mathbf{B}^{(0)}\right) \mathbf{u}^{(0,1)} &= \underbrace{\check{\mathbf{f}}^{(0)} - \mathbf{C}^{(0)} \left(\mathbf{A}^{(0)}\right)^{-1} \hat{\mathbf{f}}^{(0)}}_{=\mathbf{f}^{(0,1)}}.
\end{aligned} \tag{3.40}$$

3.4 Block cyclic reduction method

Generally speaking, the reduction and back substitution formulas presented in Subsection 3.3.1 can be generalized for an arbitrary field $\mathbb{K}$ provided that the divisions are replaced by multiplicative inverses when necessary. In particular, if the field $\mathbb{K}$ consists of matrices, then the reduction principle can be applied to block tridiagonal linear systems. Actually, these BCR methods have a long history starting in the year 1965 [93]. The first formulation used different reduction formulas and was notoriously numerically unstable. However, the so-called Buneman's variant [43] managed to stabilize the method by slightly modifying the formulation.

An even earlier attempt to stabilize the first formulation yielded the so-called FACR(l) method [94, 155], which combined the BCR method with the so-called Fourier analysis method [93]. These early formulations can be applied only to a very small subset of block tridiagonal linear systems, but a more generalized method was presented in [160]. Later, a partial fraction technique was applied to matrix rational polynomials occurring in the Buneman formulas [161] which resulted in a parallel variant. In addition to the above-listed formulations, the following variants are noteworthy: [7, 59, 91, 144, 148, 156, 157, 158, 159]. In addition, the convergence and stability properties of the method have received a lot of attention; see, for example, [29, 91, 144, 141, 176]. A recent survey regarding the CR method and its many applications can be found in [29].

Remark 3.3 *This dissertation focuses on block tridiagonal linear systems but the basic idea of the CR method can be generalized to Toeplitz-like block Hessenberg matrices [25, 26, 27], banded Toeplitz systems [28], factorizing matrix polynomial and power series [21, 22, 23], solving quadratic and nonlinear matrix equations [26, 27, 30], and solving algebraic Riccati equations [24, 85].*

3.4.1 Basic reduction formulas

Let us assume that we have a separable block tridiagonal linear system of the form

$$\begin{bmatrix} \mathbf{D} & -\mathbf{I}_{n_2} & & \\ -\mathbf{I}_{n_2} & \mathbf{D} & \ddots & \\ & \ddots & \ddots & -\mathbf{I}_{n_2} \\ & & -\mathbf{I}_{n_2} & \mathbf{D} \end{bmatrix} \begin{bmatrix} \mathbf{u}_1 \\ \mathbf{u}_2 \\ \vdots \\ \mathbf{u}_{n_1} \end{bmatrix} = \begin{bmatrix} \mathbf{f}_1 \\ \mathbf{f}_2 \\ \vdots \\ \mathbf{f}_{n_1} \end{bmatrix}, \tag{3.41}$$

where $\mathbf{D} \in \mathbb{R}^{n_2 \times n_2}$, $\mathbf{u}_i, \mathbf{f}_i \in \mathbb{R}^{n_2}$, and $n_1 = 2^k - 1$ for some positive integer k. As noted in Subsection 3.2.1, taking $\mathbf{D} = \text{tridiag}\{-1, 4, -1\}$ corresponds to a particular type of Poisson boundary value problem.

During the first reduction step, the odd-numbered block rows are multiplied by the matrix $\mathbf{D}^{-1}$, after which they can be eliminated from the system in the same way as described in Subsection 3.3.1. Thereafter, the remaining block tridiagonal subsystems are defined as follows: Let $\mathbf{T}^{(0)} = \mathbf{I}_{n_2}$, $\mathbf{D}^{(0)} = \mathbf{D}$, and $\mathbf{f}^{(0)} = \mathbf{f}$. For each reduction step $i = 1, 2, \ldots, k-1$, we have

$$\begin{bmatrix} \mathbf{D}^{(i)} & -\mathbf{T}^{(i)} & & \\ -\mathbf{T}^{(i)} & \mathbf{D}^{(i)} & \ddots & \\ & \ddots & \ddots & -\mathbf{T}^{(i)} \\ & & -\mathbf{T}^{(i)} & \mathbf{D}^{(i)} \end{bmatrix} \begin{bmatrix} \mathbf{u}_1^{(i)} \\ \mathbf{u}_2^{(i)} \\ \vdots \\ \mathbf{u}_{2^{k-i}-1}^{(i)} \end{bmatrix} = \begin{bmatrix} \mathbf{f}_1^{(i)} \\ \mathbf{f}_2^{(i)} \\ \vdots \\ \mathbf{f}_{2^{k-i}-1}^{(i)} \end{bmatrix}, \tag{3.42}$$

where

$$\begin{aligned} \mathbf{T}^{(i)} &= \left(\mathbf{T}^{(i-1)}\right)^2 \left(\mathbf{D}^{(i-1)}\right)^{-1}, \\ \mathbf{D}^{(i)} &= \mathbf{D}^{(i-1)} - 2\left(\mathbf{T}^{(i-1)}\right)^2 \left(\mathbf{D}^{(i-1)}\right)^{-1}, \text{ and} \\ \mathbf{f}_j^{(i)} &= \mathbf{f}_{2j}^{(i-1)} + \mathbf{T}^{(i-1)} \left(\mathbf{D}^{(i-1)}\right)^{-1} \left(\mathbf{f}_{2j-1}^{(i-1)} + \mathbf{f}_{2j+1}^{(i-1)}\right). \end{aligned} \tag{3.43}$$

The adopted back substitution formula reads as follows:

$$\mathbf{u}_j^{(i)} = \begin{cases} \left(\mathbf{D}^{(i)}\right)^{-1} \left(\mathbf{f}_j^{(i)} + \mathbf{T}^{(i)} \left(\mathbf{u}_{(j-1)/2}^{(i+1)} + \mathbf{u}_{(j-1)/2+1}^{(i+1)}\right)\right), & \text{when } j \notin 2\mathbb{N}, \\ \mathbf{u}_{j/2}^{(i+1)}, & \text{when } j \in 2\mathbb{N}, \end{cases} \tag{3.44}$$

where $j = 1, 2, \ldots, 2^{k-i} - 1$, and $\mathbf{u}_0^{(i+1)} = \mathbf{u}_{2^{k-r-1}}^{(i+1)} = \mathbf{0}$. Similar but more generalized reduction and back substitution formulas appeared in [91]. As shown in [91], the above formulas are well-defined, that is, $\left(\mathbf{D}^{(i)}\right)^{-1}$, $i = 1, 2, \ldots, k-1$ exist if $\mathbf{D}^{-1}$ exists, and the coefficient matrix in (3.41) is strictly diagonally dominant by rows.

3.4.2 Partial fraction variant

Although the formulas (3.43) and (3.44) could be applied directly, the approach has several drawbacks. First, even if the diagonal block $\mathbf{D}$ is tridiagonal, the recursively defined matrices $\mathbf{D}^{(i)}$ and $\mathbf{T}^{(i)}$ can fill up quickly. This is very expensive in terms of arithmetical complexity and memory requirements. Second, as with the scalar CR method, the resulting method cannot be parallelized effectively.

The first step towards improving the formulation is to define the *matrix rational polynomials* $\alpha^{(i)}(\mathbf{D})$, $\beta^{(i)}(\mathbf{D})$, $i = 0, 1, \ldots, k-1$, by starting with

$$\alpha^{(0)}(\mathbf{D}) = \mathbf{D}, \ \beta^{(0)}(\mathbf{D}) = \mathbf{I}_{n_2}, \tag{3.45}$$

and then

$$\begin{aligned} \alpha^{(i)}(\mathbf{D}) &= \left(\alpha^{(i-1)}(\mathbf{D})\right)^2 - 2\mathbf{I}_{n_2}, \text{ and} \\ \beta^{(i)}(\mathbf{D}) &= \beta^{(i-1)}(\mathbf{D}) \left(\alpha^{(i-1)}(\mathbf{D})\right)^{-1}. \end{aligned} \tag{3.46}$$

Now, the following results from [144] allow us to express the matrices $\mathbf{D}^{(i)}$ and $\mathbf{T}^{(i)}$ in an alternative form:

Lemma 3.4 *The recursively defined matrices* $\mathbf{D}^{(i)}$ *and* $\mathbf{T}^{(i)}$ *can expressed as*

$$\mathbf{D}^{(i)} = \beta^{(i)}(\mathbf{D})\alpha^{(i)}(\mathbf{D}) \;\; and \;\; \mathbf{T}^{(i)} = \beta^{(i)}(\mathbf{D}).$$

Lemma 3.5 *The recursively defined matrix rational polynomials* $\alpha^{(i)}(\mathbf{D})$ *and* $\beta^{(i)}(\mathbf{D})$ *can be factorized as*

$$\alpha^{(i)}(\mathbf{D}) = \prod_{j=1}^{2^i} (\mathbf{D} - \theta(j,i)\mathbf{I}_{n_2}) \;\; and \;\; \beta^{(i)}(\mathbf{D}) = \prod_{j=1}^{2^i-1} (\mathbf{D} - \phi(j,i)\mathbf{I}_{n_2})^{-1},$$

where

$$\theta(j,i) = 2\cos\left(\frac{2j-1}{2^{i+1}}\pi\right) \;\; and \;\; \phi(j,i) = 2\cos\left(\frac{j}{2^i}\pi\right).$$

Thus, the reduction and back substitution formulas (3.43) and (3.44) reduce to a series of multiplications with tridiagonal matrices and their inverses.

The second step is to apply the partial fraction technique introduced in [161]. The basic idea is given by the following lemma:

Lemma 3.6 (Partial fraction technique) *Let* $p(x)$ *and* $q(x)$ *be two polynomials with the following properties:*

1. *p and q are relatively prime,*
2. $\deg(p) < \deg(q) = n$*, and*
3. *the roots,* $\alpha_1, \alpha_2, \ldots, \alpha_n$*, of q are distinct.*

Then, we have:

$$\frac{p(x)}{q(x)} = \sum_{j=1}^{n} \frac{c_j}{x - \alpha_j},$$

where

$$c_j = \frac{p(\alpha_j)}{q'(\alpha_j)}.$$

The application of Lemma 3.6 leads to [144]:

$$\mathbf{T}^{(i)}\left(\mathbf{D}^{(i)}\right)^{-1} = 2^{-i}\sum_{j=1}^{2^i}(-1)^{j-1}\sin\left(\frac{2j-1}{2^{i+1}}\pi\right)(\mathbf{D} - \theta(j,i)\mathbf{I}_{n_2})^{-1} \tag{3.47}$$

and

$$\left(\mathbf{D}^{(i)}\right)^{-1} = 2^{-i}\sum_{j=1}^{2^i}(\mathbf{D} - \theta(j,i)\mathbf{I}_{n_2})^{-1}. \tag{3.48}$$

Thus, the reduction and back substitution formulas (3.43) and (3.44) reduce even further to a solution of a set of tridiagonal linear systems. The arithmetical complexity of the final partial fraction variant is $\mathcal{O}(n_1 n_2 \log n_1)$ when the Thomas method is used to solve the arising tridiagonal subproblems.

3.5 PSCR method

The PSCR method was introduced (in a radix-2 form) in the 1980s by Vassilevski [165, 166] and Kuznetsov [106]. The method shares many similarities with the BCR methods, such as the one depicted in Subsection 3.4.2. However, instead of using matrix rational polynomials and partial fractions, the method relies on a so-called partial solution technique [10, 107]. This technique can be applied very effectively in situations where the right-hand side vector contains only a few non-zero elements and we only are interested in certain components of the solution vector. A more generalized radix-q algorithm was presented later in [108]. Parallel CPU implementations were considered in [1, 137, 145].

3.5.1 Projection matrix formulation

This subsection describes the radix-q PSCR method using *orthogonal projection matrices* similar to [145]. For the sake of simplicity and without major loss of generality, we assume from now on that we are dealing with a linear system where the coefficient matrix is of the form (3.1) and $n_1 = q^k - 1$ for some integers q and k. The case (3.2) requires some minor changes to the formulas and, in the end, reduces to a set of subproblems with coefficient matrices of the form (3.1).

Let us first define the sets $J_0, J_1, \dots, J_k \subset \mathbb{N}$ and $K_1, K_2, \dots, K_k \subset \mathbb{N}$ as

$$\begin{aligned} J_0 &= \{1, 2, 3, \dots, n_1\}, \\ J_i &= \left\{ j \cdot q^i : j = 1, 2, \dots, q^{k-i} - 1 \right\}, i = 1, 2, \dots, k-1, \\ J_k &= \emptyset \end{aligned} \tag{3.49}$$

and

$$K_i = \bigcup_{j \in J_i} \{j-1\} \cup \{j+1\},\ i = 1, 2, \dots, k. \tag{3.50}$$

With these sets we can define orthogonal projection matrices

$$\tilde{\mathbf{P}}^{(i)} = \operatorname{diag}\{p_1^{(i)}, p_2^{(i)}, \dots, p_{n_1}^{(i)}\} \in \mathbb{K}^{n_1},\ i = 0, 1, \dots, k \tag{3.51}$$

and

$$\tilde{\mathbf{T}}^{(i)} = \operatorname{diag}\{t_1^{(i)}, t_2^{(i)}, \dots, t_{n_1}^{(i)}\} \in \mathbb{K}^{n_1},\ i = 1, 2, \dots, k, \tag{3.52}$$

with

$$p_j^{(i)} = \begin{cases} 1, & j \notin J_i, \\ 0, & j \in J_i \end{cases} \quad \text{and} \quad t_j^{(i)} = \begin{cases} 1, & j \in K_i, \\ 0, & j \notin K_i. \end{cases} \tag{3.53}$$

From these, we get

$$\mathbf{P}^{(i)} = \tilde{\mathbf{P}}^{(i)} \otimes \mathbf{I}_{n_2},\ i = 0, 1, \dots, k \tag{3.54}$$

and

$$\mathbf{T}^{(i)} = \tilde{\mathbf{T}}^{(i)} \otimes \mathbf{I}_{n_2}, \ i = 1, 2, \ldots, k. \tag{3.55}$$

If the projected matrix $\mathbf{P}^{(i)}\mathbf{A}\mathbf{P}^{(i)}$ is nonsingular in subspace $\mathrm{Im}(\mathbf{P}^{(i)})$ for all $i = 1, 2, \ldots, k$, then the linear system $\mathbf{A}\mathbf{u} = \mathbf{f}$ can be solved using the following algorithm:

Algorithm 3.1

```
Set $\mathbf{f}^{(1)} = \mathbf{f}$.
for $i = 1, 2, \ldots, k-1$ do
    Solve the vector $\mathbf{v}^{(i)}$ from
        $\mathbf{P}^{(i)}\mathbf{A}\mathbf{P}^{(i)}\mathbf{v}^{(i)} = \mathbf{P}^{(i)}\mathbf{f}^{(i)}$.    (3.56)
    Compute
        $\mathbf{f}^{(i+1)} = \mathbf{f}^{(i)} - \mathbf{A}\mathbf{P}^{(i)}\mathbf{v}^{(i)}$.    (3.57)
end for
Set $\mathbf{u}^{(k+1)} = \mathbf{0}$.
for $i = k, k-1, \ldots, 1$ do
    Solve the vector $\mathbf{u}^{(i)}$ from
        $\mathbf{P}^{(i)}\mathbf{A}\mathbf{P}^{(i)}\mathbf{u}^{(i)} = \mathbf{P}^{(i)}\mathbf{f}^{(i)} - \mathbf{P}^{(i)}\mathbf{A}\left(\mathbf{I}_{n_1n_2} - \mathbf{P}^{(i)}\right)\mathbf{u}^{(i+1)}$.    (3.58)
    Compute
        $\left(\mathbf{I}_{n_1n_2} - \mathbf{P}^{(i)}\right)\mathbf{u}^{(i)} = \left(\mathbf{I}_{n_1n_2} - \mathbf{P}^{(i)}\right)\mathbf{u}^{(i+1)}$.    (3.59)
end for
Set $\mathbf{u} = \mathbf{u}^{(1)}$.
```

The reason Algorithm 3.1 is effective at solving the system $\mathbf{A}\mathbf{u} = \mathbf{f}$ unravels once we observe the following results from [145]:

Lemma 3.7 *In (3.56)–(3.57), for all $i = 1, 2, \ldots, k-1$,*

$$\mathbf{P}^{(i)}\mathbf{f}^{(i)} \in \mathrm{Im}\left(\mathbf{P}^{(i)}\left(\mathbf{I}_{n_1n_2} - \mathbf{P}^{(i-1)}\right)\right)$$

and

$$\mathbf{f}^{(i+1)} = \left(\mathbf{I}_{n_1n_2} - \mathbf{P}^{(i)}\right)\left(\mathbf{f}^{(i)} - \mathbf{A}\mathbf{T}^{(i)}\mathbf{v}^{(i)}\right).$$

Lemma 3.8 *In (3.58)–(3.59), for $i = k, k-1, \ldots, 1$,*

$$\mathbf{P}^{(i)}\mathbf{f}^{(i)} - \mathbf{P}^{(i)}\mathbf{A}\left(\mathbf{I}_{n_1n_2} - \mathbf{P}^{(i)}\right)\mathbf{u}^{(i+1)} \in \mathrm{Im}\left(\mathbf{P}^{(i)}\left(\mathbf{I}_{n_1n_2} - \mathbf{P}^{(i-1)}\right)\right) \cup \mathrm{Im}\left(\mathbf{P}^{(i)}\right).$$

In addition, the solution $u^{(i)}$ is needed in the image subspace of $\mathbf{P}^{(i)}\left(\mathbf{I}_{n_1n_2} - \mathbf{P}^{(i-1)}\right)$ for $i = k, k-1, \ldots, 1$.

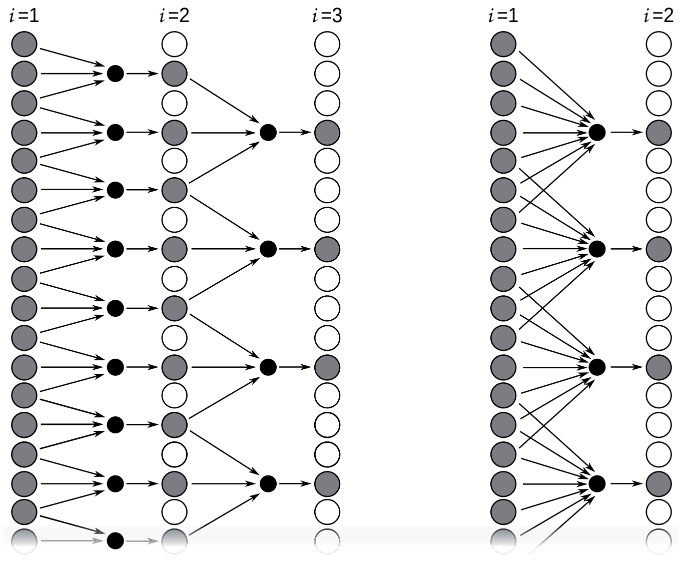


FIGURE 12 An example of how the reduction stages of a radix-2 PSCR method (on the left) and a radix-4 PSCR method (on the right) access the memory when the data is kept in place.

Lemmas 3.7 and 3.8 imply that only a very sparse set of solution components are actually required in the update formulas (3.57) and (3.58), and the right-hand side vectors in the linear systems (3.56) and (3.58) contain only a few non-zero elements.

The choice of the integer q and the sets $J_0, J_1, \ldots, J_k$ determine the radix number of the algorithm. A radix-q method reduces the system size by a factor of q in each reduction step. Thus, a radix-4 method requires only half as many reduction steps as a radix-2 method. A higher radix number would potentially benefit multi-core CPUs and GPUs because the number of consecutive tasks will be reduced. However, the arithmetical complexity estimate shown in [145] indicates that the optimal value for q is somewhere in the range of 4–6. The system size can be arbitrary, but the sets $J_0, J_1, \ldots, J_k$ and $K_1, K_2, \ldots, K_k$ must then be chosen differently in that case. Figure 12 illustrates how the reduction stages of a radix-2 PSCR method and a radix-4 PSCR method access the memory when the data is kept in place.

Remark 3.4 *A radix-q adaptation of the Schur's complement formulation discussed in Remark 3.1 actually leads to the same subproblems during the first reduction step and the last back substitution step. Considering the connection highlighted later in Remark 3.6, it is very likely that an alternative formulation for the radix-q PSCR method could be obtained using the Schur's complement approach.*

3.5.2 Partial solution technique

The subproblems resulting from (3.56)–(3.59) are actually of the form $\tilde{\mathbf{A}}\mathbf{v} = \mathbf{g}$ with

$$\tilde{\mathbf{A}} = \tilde{\mathbf{A}}_1 \otimes \mathbf{M}_2 + \tilde{\mathbf{A}}_1 \otimes \mathbf{A}_2 + c(\tilde{\mathbf{M}}_1 \otimes \mathbf{M}_2), \tag{3.60}$$

where $\tilde{\mathbf{A}}_1 \in \mathbb{K}^{m\times m}$ and $\tilde{\mathbf{M}}_1 \in \mathbb{K}^{m\times m}$ are non-zero diagonal blocks from projected matrices $\tilde{\mathbf{P}}^{(i)}\mathbf{A}_1\tilde{\mathbf{P}}^{(i)}$ and $\tilde{\mathbf{P}}^{(i)}\mathbf{M}_1\tilde{\mathbf{P}}^{(i)}$, respectively. Furthermore, Lemmas 3.7 and 3.8 imply that we are actually interested in computing only a vector $\mathbf{Rv}$ and $\mathbf{g} \in \mathrm{Im}(\mathbf{Q})$, where the orthogonal projection matrices $\mathbf{R} \in \mathbb{K}^{mn_2\times mn_2}$ and $\mathbf{Q} \in \mathbb{K}^{mn_2\times mn_2}$ define the required solution components and the non-zero components of the right-hand side vector, respectively. Naturally, $\mathbf{R} = \tilde{\mathbf{R}} \otimes \mathbf{I}_{n_2}$ and $\mathbf{Q} = \tilde{\mathbf{Q}} \otimes \mathbf{I}_{n_2}$ for suitable orthogonal projection matrices $\tilde{\mathbf{R}} \in \mathbb{K}^{m\times m}$ and $\tilde{\mathbf{Q}} \in \mathbb{K}^{m\times m}$.

Let us consider the following *generalized eigenvalue problem*:

$$\tilde{\mathbf{A}}_1\mathbf{w}_j = \lambda_j\tilde{\mathbf{M}}_1\mathbf{w}_j. \tag{3.61}$$

Because the matrix $\tilde{\mathbf{A}}_1$ is symmetric, we can now assume that the eigenvectors $\mathbf{w}_1, \mathbf{w}_2, \ldots, \mathbf{w}_m \in \mathbb{K}^m$ are $\tilde{\mathbf{M}}_1$-orthonormal. Thus, the matrices

$$\mathbf{W} = [\mathbf{w}_1\mathbf{w}_2 \ldots \mathbf{w}_m] \ \text{ and } \ \mathbf{\Lambda} = \mathrm{diag}\,\{\lambda_1, \lambda_2, \ldots, \lambda_m\} \tag{3.62}$$

have the properties

$$\mathbf{W}^T\tilde{\mathbf{A}}_1\mathbf{W} = \mathbf{\Lambda} \ \text{ and } \ \mathbf{W}^T\tilde{\mathbf{M}}_1\mathbf{W} = \mathbf{I}_m. \tag{3.63}$$

The vector $\mathbf{Rv}$ can be computed effectively using the following result from [10, 107]:

Theorem 3.1 (Partial solution technique) *The projected matrix $\mathbf{R}\tilde{\mathbf{A}}^{-1}\mathbf{Q}$ can be decomposed as*

$$(\tilde{\mathbf{R}}\mathbf{W} \otimes \mathbf{I}_{n_2})((\mathbf{\Lambda} + c\mathbf{I}_m) \otimes \mathbf{M}_2 + \mathbf{I}_m \otimes \mathbf{A}_2)^{-1}(\mathbf{W}^T\tilde{\mathbf{Q}} \otimes \mathbf{I}_{n_2}).$$

The above result follows from Lemmas 3.1, 3.2, and 3.3 and implies that each partial solution reduces into a set of tridiagonal linear systems. The arithmetical complexity of the final PSCR algorithm is $\mathcal{O}(n_1n_2\log n_1)$ when the Thomas method is used to solve the arising tridiagonal subproblems.

Remark 3.5 *More explicit formulas for the radix-q PSCR method can be found in the included article [PIII].*

Remark 3.6 *As noted in [144] and the included article [PII], the radix-2 or radix-4 variants of the PSCR method are equivalent to the radix-2 and radix-4 BCR methods discussed in Subsection 3.4.2 and in the included article [PII], respectively, when applied to a system of the form*

$$(\mathbf{A} \otimes \mathbf{I}_{n_2} + \mathbf{I}_{q^k-1} \otimes (\mathbf{D} - 2\mathbf{I}_{n_2}))\mathbf{u} = \mathbf{f}, \tag{3.64}$$

where $\mathbf{D} \in \mathbb{R}^{n_2\times n_2}$ and $\mathbf{A} = \mathrm{tridiag}\{-1, 2, -1\} \in \mathbb{R}^{(q^k-1)\times(q^k-1)}$ for some positive integer k. Above, $q \in \{2, 4\}$.

Remark 3.7 *Numerous similar methods have been derived by using the properties of the Kronecker matrix tensor product. Reference [20] contains a comprehensive survey of these so-called matrix decomposition algorithms.*

4 AUGMENTED LAGRANGIAN METHODS

This chapter provides a brief introduction to the decomposition principle, the AL technique, and the *alternating direction method of multipliers* (ADMM) following a similar presentation, as in [78]. Together, these techniques allow us to decompose a complex minimization problem into a set of simpler subproblems through the introduction of auxiliary variables. These auxiliary variables are decoupled by adding additional constraints to the problem in order keep the new constrained minimization problem equivalent to the original minimization problem. The resulting constrained minimization problem is then associated with a suitable *augmented Lagrangian functional* (ALF) whose saddle-point corresponds to the solution of the original minimization problem. The resulting solution algorithms are modular, and each subproblem can be treated separately using methods that are best suited for the particular subproblem. These techniques have a well-established role in analyzing optimization problems and deriving solution algorithms for them [19, 71, 76, 78, 101].

4.1 Preliminaries

Before the basics of the AL methods can be covered, we must give a few definitions:

Definition 4.1 (Proper functional) *An extended valued functional G from a function space V into $\bar{\mathbb{R}}$ is said to be proper if $G(v) > -\infty$ for all $v \in V$ and $G(v_0) < +\infty$ for at least one $v_0 \in V$.*

Definition 4.2 (Convex functional) *An extended valued functional G from a function space V into $\bar{\mathbb{R}}$ is said to be convex if for every $v_1, v_2 \in V$ and $t \in [0, 1]$*

$$G(tv_1 + (1-t)v_2) \leq tG(v_1) + (1-t)G(v_2).$$

Definition 4.3 (Strictly convex functional) *An extended valued functional G from a function space V into $\bar{\mathbb{R}}$ is said to be strictly convex if for every $v_1, v_2 \in V$, $v_1 \neq v_2$, and $t \in [0,1]$*

$$G(tv_1 + (1-t)v_2) < tG(v_1) + (1-t)G(v_2).$$

Definition 4.4 (Uniformly convex functional) *An extended valued functional G from a Hilbert space V into $\bar{\mathbb{R}}$ is said to be uniformly convex if $\delta_G(t) > 0$ for all $t > 0$, where the modulus of convexity is defined as*

$$\delta_G(t) = \inf\left\{\frac{1}{2}G(v_1) + \frac{1}{2}G(v_2) - G\left(\frac{v_1+v_2}{2}\right) : \|v_1 - v_2\|_V = t,\ v_1, v_2 \in V\right\}.$$

Definition 4.5 (Coercive functional) *An extended valued functional G from a Hilbert space V into $\bar{\mathbb{R}}$ is said to be coercive if $\lim_{\|v\|_V \to +\infty} G(v) = +\infty$.*

Definition 4.6 (Lower semicontinuous functional) *An extended valued functional G from a function space V into $\bar{\mathbb{R}}$ is said to be lower semicontinuous at $v_0 \in V$ if*

- *for every $\varepsilon > 0$ there exists a neighborhood $U \subset V$ of v_0 such that $G(v) \geq G(v_0) - \varepsilon$ for all $v \in V$ when $G(v_0) < +\infty$, and*
- *$G(v) \to +\infty$ as $v \to v_0$ when $G(v_0) = +\infty$.*

Definition 4.7 (Domain) *The domain of an extended valued functional G from a function space V into $\bar{\mathbb{R}}$ is defined as*

$$\mathrm{dom}(G) = \{v \in V : -\infty < G(v) < +\infty\}.$$

4.2 Decomposition principle

Let us assume that we have

- Hilbert spaces V and H,
- a linear operator $B \in \mathcal{L}(V, H)$,
- a convex, proper, lower semicontinuous functional F from H into $\mathbb{R} \cup \{+\infty\}$, and
- a convex, proper, lower semicontinuous functional G from V into $\mathbb{R} \cup \{+\infty\}$,

with the property

$$\mathrm{dom}(G) \cap \mathrm{dom}(F \circ B) \neq \emptyset. \tag{4.1}$$

We are interested on decomposing the following *minimization problem*:

$$\begin{cases} u \in V, \\ \mathcal{J}(u) \leq \mathcal{J}(v) \quad \forall v \in V, \end{cases} \tag{4.2}$$

where

$$\mathcal{J}(v) = F(Bv) + G(v). \tag{4.3}$$

If the functional $\mathcal{J}$ is coercive, then the minimization problem (4.2) admits a solution, and the solution is unique if the functional $\mathcal{J}$ is strictly convex [78].

We begin by defining a subset $W \subset V \times H$ as

$$W = \{(v, q) \in V \times H : Bv = q\} \tag{4.4}$$

and a new objective functional $j : W \to \mathbb{R} \cup \{+\infty\}$ as

$$j(v, q) = F(q) + G(v). \tag{4.5}$$

Now, the minimization problem (4.2) can be decomposed to the following *constrained two-variable minimization problem*:

$$\begin{cases} (u, p) \in W, \\ j(u, p) \leq j(v, q) \ \ \forall (v, q) \in W. \end{cases} \tag{4.6}$$

Above, p is the new *auxiliary variable*.

Clearly, if we obtain a solution (u, p) to the constrained two-variable minimization problem (4.6), then u is also a solution to the original minimization problem (4.2). While these two minimization problems are mathematically equivalent, the constrained two-variable minimization problem is often easier to solve numerically. This decomposition principle can be applied more than once, in which case multiple auxiliary variables and constraints are introduced.

4.3 Augmented Lagrangian functional

The constrained minimization problem (4.6) could be tackled using the following well-known methods:

Method of Lagrange multipliers: The constrained minimization problem (4.6) is associated with a *Lagrange functional* $\mathcal{L} : V \times H \times H \to \bar{\mathbb{R}}$:

$$\mathcal{L}(v, q; \mu) = F(q) + G(v) + (\mu, Bv - q)_H. \tag{4.7}$$

Above, $\mu \in H$ is called a *Lagrange multiplier*. A solution to the minimization problem (4.6) is then obtained by finding a *saddle-point*

$$\omega = (u, p; \lambda) \in V \times H \times H \tag{4.8}$$

for the Lagrange functional $\mathcal{L}$, that is,

$$\mathcal{L}(u, p; \mu) \leq \mathcal{L}(u, p; \lambda) \leq \mathcal{L}(v, q; \lambda) \tag{4.9}$$

for all $(v, q; \mu) \in V \times H \times H$.

Penalty methods: An additional penalty term is introduced to the objective functional j. A *quadratic penalty term* would lead to

$$j_r(v,q) = F(q) + G(v) + \frac{r}{2}\|Bv - q\|_H^2, \tag{4.10}$$

where $r > 0$. A solution to the minimization problem (4.6) is then obtained by iteratively increasing the value of the *penalty coefficient* r. The solution of the previous iteration is used as an initial guess for the next iteration.

These two techniques can be combined into AL methods. The constrained minimization problem (4.6) is associated with the following ALF $\mathcal{L}_r : V \times H \times H \to \bar{\mathbb{R}}$:

$$\mathcal{L}_r(v,q;\mu) = F(q) + G(v) + \frac{r}{2}\|Bv - q\|_H^2 + (\mu, Bv - q)_H, \tag{4.11}$$

where $r > 0$ and $\mu \in H$. The name augmented Lagrangian comes from the fact that the functional $\mathcal{L}_r$ is basically a Lagrange functional associated with the constrained minimization problem (4.6) with an additional quadratic penalty term $\frac{r}{2}\|Bv - q\|_H^2$.

It can be easily shown (see, e.g., [71, 78]) that if we can solve the saddle-point problem

$$\begin{cases} (u,p;\lambda) \in V \times H \times H, \\ \mathcal{L}_r(u,p;\mu) \leq \mathcal{L}_r(u,p;\lambda) \leq \mathcal{L}_r(v,q;\lambda) \;\; \forall (v,q;\mu) \in V \times H \times H, \end{cases} \tag{4.12}$$

then (u,p) is a solution to the constrained minimization problem (4.6). Furthermore, u is a solution to the original minimization problem (4.2).

When compared to ordinary penalization methods, the AL formulation benefits from the fact that the exact solution to (4.6) can be obtained without making the penalty parameter r tend to infinity. The introduction of more than one auxiliary variable leads to additional Lagrange multipliers and penalty terms.

4.4 Alternating direction method of multipliers

A wide variety of methods could be applied to the saddle-point problem (4.12), but the following ADMM method is advocated in the included articles [PIV, PV]:

Algorithm 4.1
```
*Initialize* $p^{(0)}$ *and* $\lambda^{(0)}$.
***for*** $i = 1,2,\ldots$ ***do***
    $u^{(i)} = \arg\min_{v \in V} \mathcal{L}_r(v, p^{(i-1)}; \lambda^{(i-1)})$.
    $p^{(i)} = \arg\min_{q \in H} \mathcal{L}_r(u^{(i)}, q; \lambda^{(i-1)})$.
    $\lambda^{(i)} = \lambda^{(i-1)} + \rho^{(i)}(Bu^{(i)} - p^{(i)})$.
***end for***
```

Above, $\rho^{(i)} > 0, i = 1, 2, \dots$.

Algorithm 4.1 is commonly referred to as ALG-2 (see, e.g., [15, 71, 75, 78]). This alternating direction approach effectively splits the original minimization problem into two subproblems that are then solved sequentially. If the decomposition is done cleverly enough, then this *operator splitting* technique can greatly improve the effectiveness of the resulting solution algorithm, as each subproblem can be solved using a dedicated solver.

The following convergence result holds for Algorithm 4.1:

Theorem 4.1 *Suppose that the following conditions apply:*

- *The ALF $\mathcal{L}_r$ has a saddle-point $(u, p; \lambda) \in V \times H \times H$.*
- *The ALF $\mathcal{L}_r(v, q; \mu)$*
  - *is coercive over $V \times H$ for any fixed μ,*
  - *is proper on V for any fixed q and μ, and*
  - *is proper on H for any fixed v and μ.*
- *Either*
  - *F is uniformly convex on the bounded sets of H with B injective and with $Im(B)$ closed, or*
  - *G is uniformly convex on the bounded sets of V.*
- *The parameters $\rho^{(1)}, \rho^{(2)}, \dots$ satisfy*

$$0 < \rho^{(i)} = \rho < \frac{1 + \sqrt{5}}{2}\, r.$$

Then,

- *the sequence $(u^{(i)}, p^{(i)}; \lambda^{(i)})$ is well defined,*
- *$F(p^{(i)}) + G(u^{(i)}) \rightarrow F(p) + G(u)$,*
- *$u^{(i)} \rightarrow u$ strongly in V,*
- *$p^{(i)} \rightarrow p$ strongly in H, and*
- *$\lambda^{(i)} \rightarrow \lambda$ weakly in H.*

The above result is from [78]. A proof can be found in the same reference.

Remark 4.1 *As noted in [78], if the spaces V and H are finite-dimensional, then the conditions become much weaker because the saddle-point $(u, p; \lambda)$ always exists and the uniform convexity of the functionals F and G is not required. However, the functionals F and G are still required to be convex.*

5 IMAGE DENOISING

This chapter deals with image denoising. The chapter begins by providing a brief introduction to image denoising and commonly used image denoising techniques. The chapter then proceeds to define the L1MC image denoising model and to summarize a few previously developed solution algorithms.

5.1 Formal problem formulation and preliminaries

A digital image is usually conceived as a discrete mapping

$$\hat{v} : \hat{\Omega} \to U^d, \tag{5.1}$$

where $\hat{\Omega} = \{1, 2, \dots, W\} \times \{1, 2, \dots, H\}$, $U \subset \mathbb{Z}$, and $d \in \{1, 3\}$. The case $d = 1$ corresponds to a *grayscale image* and $d = 3$ corresponds to a *color image*. From now on, it is assumed that $d = 1$. Usually, $U = \{0, 1, \dots, 255\}$, where 0 represents black and 255 represents white in grayscale images.

In this dissertation, an image is actually conceived as a member of a suitable *function space* V. More specifically, the image is represented by a function $v : \Omega \to \mathbb{R}$, where Ω is a rectangular bounded domain of $\mathbb{R}^2$. The function space V can be, for example, a piecewise linear finite element space, in which case the function v is a piecewise linear interpolation of the discrete mapping $\hat{v}$.

In a general case, image denoising is a process of finding a solution $u \in V$ to an *inverse problem*

$$Ku + g = f, \tag{5.2}$$

where the function $f \in V$ is the *noisy image* and the operator $K : V \to V$ and the function $g \in V$ model the *non-additive* and *additive* parts of the noise, respectively [82, 102]. This is a particularly challenging problem because the inverse operator K^{-1} may not exist and the function g is usually unknown. However, in many cases it is possible to obtain a kind of "pseudoinverse" for the operator K, and a

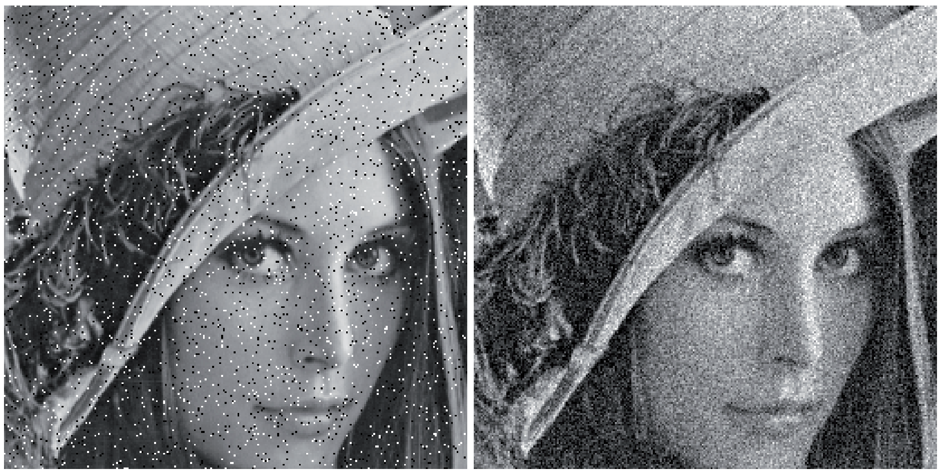


FIGURE 13 Two examples of noisy images: "salt and pepper" noise (on the left) and Gaussian noise (on the right).

sufficient amount of information is known about the function g. Thus, the solution u, or a close representative of it, can be obtained. The articles included in this dissertation deal only with additive noise, and from now on it is assumed that $K = I$. In a general case, the operator K could model, for example, motion blur phenomena [54].

In most cases, the additive noise g that naturally occurs in images can be placed in either of the following categories:

"Salt and pepper" / impulse noise: Only a small minority of pixels are affected, but the colors of the affected pixels can be very different from their true values and their surrounding pixels. Salt and pepper noise usually manifests as dark or white dots in the image. A partially faulty sensor and dust are two common sources of this type of noise.

Gaussian noise: Virtually all pixels are affected, but the colors of the affected pixels do not (usually) differ very much from their true values and their surrounding pixels. Usually, the histogram that arises when the values of the added noise are plotted against the frequency with which they occur follows a normal distribution.

Figure 13 shows examples of these two noise types. The test images considered in the included articles [PIV] and [PV] were of the second type.

5.2 Filter and frequency domain-based methods

Linear filter methods are based around the idea of convolving the noisy image with a suitable (radial) mask. This mask may be a simple low-pass filter or a smoothing operator, such as a Gaussian kernel over the neighboring pixels (a.k.a *Gaussian* *smoothing*) shown here:

$$u(\mathbf{x}) = \int_{\Omega} G_{\mathbf{x},\sigma}(\mathbf{t} - \mathbf{x}) f(\mathbf{t}) d\mathbf{t}, \tag{5.3}$$

for all $\mathbf{x} \in \Omega$ with

$$G_{\mathbf{x},\sigma}(t) = A_{\mathbf{x}} \exp\left(-\frac{\|\mathbf{x}\|^2}{4\sigma^2}\right), \tag{5.4}$$

where the *Gaussian kernel* $G_{\mathbf{x},\sigma}$ has standard deviation σ and $A_{\mathbf{x}} \in \mathbb{R}$ is selected such that

$$\int_{\Omega} G_{\mathbf{x},\sigma}(\mathbf{t} - \mathbf{x}) d\mathbf{t} = 1. \tag{5.5}$$

Although these types of methods are simple to implement and fast, they perform poorly on the singular parts of the image, that is, the parts that contain sharp transitions (edges) and texture. *Nonlinear filter methods* may utilize, for example, median information (see, e.g., [14, 65, 82, 122]). Methods based on the weighted averages are generally considered to be best suited for noisy images that contain Gaussian noise, while filter methods based on medians are generally considered to be best suited for noisy images that contain impulse noise.

The denoising can also be performed in the frequency domain using Fourier transformations or wavelet functions (see, e.g., [9, 120]). *Wavelet methods* (see, e.g., [12, 62, 63, 61, 147, 172]) are based around the idea of decomposing the image into the wavelet basis. The denoising is accomplished by shrinking or otherwise modifying the transform coefficients. *Wavelet threshold methods* (see, e.g., [63]) rely on the idea that the actual characteristics of the image are usually represented by the wavelets with large coefficients, while the noise is generally distributed across the wavelets with small coefficients. Thus, canceling the wavelets with small coefficients would remove the noise but keep the actual characteristics of the image intact.

One of the simplest approaches is to cancel all wavelets with coefficients smaller than a given threshold. Unfortunately, these so-called *hard thresholding methods* are likely to cause unwanted oscillation (a.k.a, wavelet outliers [64]) in the denoised image. *Soft thresholding methods* [61], on the other hand, utilize a continuous threshold operators. Simple wavelet-based methods assume that the wavelet coefficients are independent. In addition, the chosen threshold does not always match the actual distributions of the image and the noise. More advanced wavelet models (see, e.g., [51, 69, 138, 149, 153]) try to solve these problems by using estimators based on Bayesian theory.

5.3 Variational-based methods

Variational-based image denoising methods treat the image as a discretely differentiable function and utilize the derivative information in the denoising process.

One of the earliest and most famous of these methods was based on *anisotropic diffusion* and was introduced in the late 1980s by Perona and Malik [135, 136]. Their idea was to improve upon the Gaussian smoothing method by convolving the noisy image f at $\mathbf{x} \in \Omega$ only in the direction orthogonal to $\nabla f(\mathbf{x})$. This choice reduced the blurring effect commonly associated with the Gaussian smoothing method and thus significantly improved the edge-preserving properties of the method.

Many variational-based methods since then have been based on solving a minimization problem of the form

$$\begin{cases} u \in V, \\ \mathcal{J}(u) \leq \mathcal{J}(v) \quad \forall v \in V. \end{cases} \tag{5.6}$$

Typically,

$$\mathcal{J}(v) = \varepsilon \mathcal{J}_r(v) + \mathcal{J}_f(v), \tag{5.7}$$

where $\mathcal{J}_r$ is the so-called *regularization term*, the purpose of which is to measure how much noise the image v contains, and $\mathcal{J}_f$ is the so-called *fidelity term* whose role is to fit the denoised image u to the noisy data f. The *parameter* $\varepsilon > 0$ defines how these two terms are balanced out.

Remark 5.1 *Sometimes the minimization problem is initially formulated as*

$$u = \arg\min_{v \in V} \mathcal{J}_r(v) \ \ s.t. \ \ \|f - v\|^2 = \sigma^2 \tag{5.8}$$

or

$$u = \arg\min_{v \in V} \mathcal{J}_r(v) \ \ s.t. \ \ \|f - v\|^2 \leq \sigma^2, \tag{5.9}$$

where σ is the variance of the noise function g in (5.2). However, the exact value of σ is usually unknown. Thus, minimization problems of the form (5.8) and (5.9) are often solved approximately using Tikhonov regularization, which leads to minimization problems that are of the form (5.6)–(5.7) with $\mathcal{J}_f(v) = \frac{1}{2}\|f - v\|^2$.

5.3.1 Total variation minimization model

Total variation minimization model (TV model) introduced by Rudin, Osher, and Fatemi (a.k.a ROF model) [146] has dominated the variational-based image denoising scene during the last two decades. The model incorporates so-called total variation norm into the regularization term $\mathcal{J}_r$:

Definition 5.1 (Total variation semi-norm) *Let Ω be an open subset of $\mathbb{R}^n$ and let $C_c^k(\Omega, \mathbb{R}^n)$ be the space of C^k functions with compact support in Ω with values in $\mathbb{R}^n$. The total variation of a function $v \in L^1(\Omega)$ is defined as*

$$\mathrm{TV}_\Omega(v) = \sup \left\{ \int_\Omega v \, \nabla \cdot \phi \, dx : \phi \in C_c^1(\Omega, \mathbb{R}^n), \|\phi\|_{L^\infty(\Omega)} \leq 1 \right\}.$$

In a discrete case, we can write

$$\mathrm{TV}_\Omega(v) = \int_\Omega \|\nabla v\| \, dx. \tag{5.10}$$

The TV model starts from the premise that the solution u consists of connected sets (i.e., objects) and smooth contours (i.e., edges). The solution is assumed to be smooth inside the connected sets but to contain jumps near the contours. On this basis, the function space V was chosen to be the space $\mathrm{BV}(\Omega)$ consisting of integrable functions with finite TV (BV space).

Definition 5.2 (Space of functions of bounded variation) *The space of functions of bounded variation (BV functions) is defined as*

$$\mathrm{BV}(\Omega) = \left\{ v \in L^1(\Omega) : \mathrm{TV}_\Omega(v) < +\infty \right\}.$$

The resulting Tikhonov regularized model is of the form

$$\mathcal{J}(v) = \varepsilon \, \mathrm{TV}_\Omega(v) + \frac{1}{2} \int_\Omega |f - v|^2 \, dx. \tag{5.11}$$

The functional (5.11) is strictly convex and lower semicontinuous. Thus, the minimum exists and is unique [45]. More importantly, the AL techniques discussed in Chapter 4 can be applied to the model (see, e.g., [36, 45, 53, 126, 170, 175]). The so-called *split Bregman iteration*-based solvers have also received some attention [80, 130]. Split Bregman solvers are very similar to the AL-based approaches as they too transform the minimization problem to a series of subproblems through the introduction of auxiliary variables. Each subproblem is defined by using the Bregman distance [38]. Some attempts have been made to solve the problem using *dual methods* [44, 48]. As noted in [175], the split Bregman iteration and these two dual methods can be seen as different procedures to solve the system resulting from a particular AL formulation. Effective solvers have also been presented in [119, 169]. A recent and comprehensive analysis of the TV model can be found in [92].

While the TV model has many good characteristics, such as the ability to retain the edges, it also has some undesirable traits. For example, the method cannot preserve image contrast, the corners get easily smeared, and the so-called *staircase effect* turns smooth surfaces into piecewise constant regions [13, 46, 60, 123, 142]. In addition, please see the numerical experiments presented in [45, 47, 99, 111].

Several *modifications* have been proposed over the years:

– Modify the *Euler-Lagrange* (EL) equation associated with the problem, for example, by multiplying it by $\|\nabla u\|$. This has been found to accelerate the minimization process and dampen the staircase effect [121].
– Modify the regularization term such that

$$\mathcal{J}_r(v) = \int_\Omega \|\nabla v\|^{P(\|\nabla v\|)} \, dx, \tag{5.12}$$

where the function $P : \mathbb{R} \to [1,2]$ is selected such that $P(\|\nabla v\|) \approx 1$ when $\|\nabla v\|$ is large and $P(\|\nabla v\|) \approx 2$ when $\|\nabla v\|$ is small [33], the point being that $\int_\Omega \|\nabla v\| \; dx$ is better at preserving edges and $\int_\Omega \|\nabla v\|^2 \; dx$ is better at preserving the other features of the image [45]. Similar approaches have been compared and analyzed in [113]. In particular, the active-set methods have received moderate attention in this context [99, 110, 111, 113], and a semi-adaptive approach was presented in [112].

– Modify the fidelity term such that

$$\mathcal{J}_f(v) = \frac{1}{s} \int_\Omega |f - v|^s \; dx, \tag{5.13}$$

where $0 < s < +\infty$ [152]. The L^1-fitting in particular has been studied (in this and in other contexts) in [77, 109] because it is better at tolerating outliers [96, 127, 129, 139, 140].

– Variants for vector valued images (i.e., color images) [32, 39, 49, 175].

Different *iterative approaches* have also received some attention:

– Initially apply the TV model to a noisy image f to form a decomposition $f = u_0 + g_0$. Then apply the model recursively to the residual noise g_0 using a smaller parameter ε to form a second decomposition $g_0 = u_1 + g_1$. Finally, after k recursion steps, we get the following decomposition of f:

$$\begin{aligned} f &= u_0 + g_0 \\ f &= u_0 + u_1 + g_1 \\ f &= \dots \\ f &= u_0 + u_1 + u_2 + \cdots + u_k + g_k, \end{aligned} \tag{5.14}$$

where $g_j = u_{j+1} + g_{j+1}$. The denoised image is obtained by $u = \sum_{j=0}^{k} u_j$ [162].

– Initially apply the TV model to a noisy image f to form a decomposition $f = u_0 + g_0$. Then apply the model recursively to the function $f + g_0$ to form a second decomposition $f + g_0 = u_1 + g_1$ and repeat the process for the function $f + g_1$. Finally, after k recursion steps, we get the decomposition $f + g_{k-1} = u_k + g_k$ [131].

Finally, several *alternative models* have been studied:

– *Total generalized variation* (TGV) model [37] with

$$\begin{aligned} \mathrm{TGV}_\Omega^{k,\boldsymbol{\alpha}}(v) = \sup \Bigg\{ \int_\Omega v \, \mathrm{div}^k \phi \; dx : \phi \in C_c^k(\Omega, \mathrm{Sym}^k(\mathbb{R}^n)), \\ \|\mathrm{div}^k \phi\|_{L^\infty(\Omega)} \leq \alpha_l, l = 0, 1, \dots, k-1 \Bigg\} \end{aligned} \tag{5.15}$$

as a regularization term. Above, div^k is the so-called k-divergence, $\mathrm{Sym}^k(\mathbb{R}^n)$ denotes the space of symmetric tensors of order k with arguments in $\mathbb{R}^n$, and $\lambda_l > 0$ for all $l = 0, 1, \dots, k-1$.

- Higher order generalizations with $H1$-space [133] and an additional elliptic operator [47].
- Non-local means models [73, 116] that involve all first-order variations and and models based on diffusion tensors [58].
- The higher order model suggested in [118] is relevant to what follows in the next subsections. The model uses the following objective functional:

$$\mathcal{J}(v) = \varepsilon \int_\Omega \left| \nabla \cdot \frac{\nabla v}{\|\nabla v\|} \right| dx + \frac{1}{2} \int_\Omega |f - v|^2 \, dx. \tag{5.16}$$

- *Euler's elastica* [4, 5, 50, 124] will be mentioned in passing:

$$\mathcal{J}(v) = \varepsilon \int_\Omega \left[a + b \left(\nabla \cdot \frac{\nabla v}{\|\nabla v\|} \right)^2 \right] \|\nabla v\| dx + \frac{1}{2} \int_\Omega |f - v|^2 \, dx, \tag{5.17}$$

  where $a > 0$ and $b > 0$.
- Other higher order models; see, for example, [18, 83, 84, 117, 178, 179].

See, for example, Reference [42] for a more comprehensive review of different image denoising methods.

Remark 5.2 *Actually, the BV-regularization has many uses outside the field of image denoising as it can be applied to other ill-posed problems as well (see, e.g., [2, 100]). Staying in the overall context of image processing, the BV-regularization has been used, for example, in image flow [8, 13] and denoising-deblurring [167] applications.*

5.3.2 L^1-mean curvature model

Let us for a moment think of image $v : \Omega \to \mathbb{R}$ as a surface in $\Omega \times \mathbb{R}$. More formally, let the function $F_v : \Omega \times \mathbb{R} \to \mathbb{R}$ be defined as

$$F_v(x_1, x_2, x_3) = v(x_1, x_2) - x_3. \tag{5.18}$$

The surface we are interested on is specified by the equation

$$F_v(x_1, x_2, x_3) = 0, \tag{5.19}$$

that is, the equation (5.19) defines the *graph* of the image v. In the model suggested by Zhu and Chan [181], the L^1-norm of the *mean curvature* (MC) of the surface (5.19) is incorporated into the regularization term $\mathcal{J}_r$. In that case, the MC is calculated as [132]

$$\kappa(F_v) = \frac{1}{2} \nabla \cdot \left(\frac{\nabla F_v}{\|\nabla F_v\|} \right) = \frac{1}{2} \nabla \cdot \left(\frac{\nabla v}{\sqrt{1 + \|\nabla v\|^2}} \right), \tag{5.20}$$

and thus the objective functional used in their L1MC model is of the form

$$\mathcal{J}(v) = \varepsilon \int_\Omega \left| \nabla \cdot \left(\frac{\nabla v}{\sqrt{1 + \|\nabla v\|^2}} \right) \right| dx + \frac{1}{2} \int_\Omega |f - v|^2 \, dx. \tag{5.21}$$

The authors in [181] suggest that taking $V = \mathrm{BV}(\Omega)$ could be a suitable choice for their model.

Because the unit normal vectors of the surface (5.19) are given by

$$\frac{\nabla F_v}{\|\nabla F_v\|}, \tag{5.22}$$

the L^1-norm of the MC, in essence, measures how much variance exist among the unit normal vectors of the surface. Thus, incorporating the MC term into the regularization term makes sense intuitively because the unit normal vectors of a noisy surface are likely to have more variance among them. Moreover, as partially proved in [181], the L^1-norm of the MC does not penalize sharp transitions of the surface. Thus, the model preserves the image contrast. In addition, partial proofs provided in [181] suggest that the model can keep sharp edges and object corners. Thus, this L1MC model could be a suitable solution to the problems associated with the TV model.

The L1MC model (5.21) can be generalized as follows:

$$\mathcal{J}(v) = \varepsilon \int_\Omega \Phi(\kappa_{v,\beta})\, dx + \frac{1}{s}\int_\Omega |f - v|^s dx, \tag{5.23}$$

where $0 < s < +\infty$, $\Phi : \mathbb{R} \to \mathbb{R}$ is a measurable function, and

$$\kappa_{v,\beta} = \nabla\cdot \left(\frac{\nabla v}{\sqrt{\|\nabla v\|^2 + \beta}} \right), \tag{5.24}$$

with $\beta \geq 0$. This model is referred to from now on as the *generalized mean curvature* (GMC) model. If $\Phi(x) = |x|$ and $s = 2$, then the case $\beta = 1$ corresponds to the L1MC model (5.21) and the case $\beta = 0$ corresponds to the model (5.16). However, the model (5.16) is usually regularized in a such a way that $0 < \beta \ll 1$. Thus, the methods developed for the model (5.16) and its alternatives could be generalized to the L1MC model (5.21).

5.3.3 Solution algorithms for the L^1-mean curvature model

If the function Φ is differentiable and $s = 2$, then the EL equation associated with the GMC model (5.23) is

$$\begin{aligned} \Psi(u) = \varepsilon\nabla\cdot \big(D_1(u)\nabla\Phi'(\kappa_{u,\beta}) - D_2(u)\nabla u\big) + u - f = 0 \ \text{ in } \Omega, \\ \nabla u \cdot \mathbf{n} = 0 \ \text{ on } \partial\Omega, \end{aligned} \tag{5.25}$$

where $\mathbf{n}$ denotes the outward unit normal to $\partial\Omega$,

$$D_1(v) = \frac{1}{\|\nabla v\|_\beta}, \ \text{ and } \ D_2(v) = \frac{\nabla v \cdot \nabla\Phi'(\kappa_{v,\beta})}{(\|\nabla v\|_\beta)^3}, \tag{5.26}$$

with $\|\mathbf{x}\|_\beta = \sqrt{\|\mathbf{x}\|^2 + \beta}$ [41, 181] .

In [181], the L1MC model (5.21) was regularized such that

$$\Phi(x) = \begin{cases} x^2, & |x| \leq 1, \\ |x|, & |x| > 1. \end{cases} \tag{5.27}$$

The authors then transformed the EL equation (5.25) into a parabolic form

$$\frac{\partial u}{\partial t} = \Psi(u) \tag{5.28}$$

and solved the resulting time-dependent problem using an *explicit Euler formula*

$$u^{(i+1)} = u^{(i)} + \triangle t \Psi(u^{(i)}). \tag{5.29}$$

As noted in [41], the magnitudes of the terms D_1 and D_2 may be very far apart, and thus the resulting discrete operator can be very unbalanced. As a result, the time step $\triangle t$ needs to be extremely small for stability reasons, and the method is extremely slow to converge.

A more advanced *AL-based solution algorithm* was later introduced in [182]. The authors extended the solution algorithm developed for the Euler's elastica model (5.17) presented in [163] to the L1MC model (5.21). They decomposed the original minimization problem as

$$(u, \mathbf{p}_1, \mathbf{p}_2, \mathbf{p}_3, \phi) = \arg\min_{(v, \mathbf{q}_1, \mathbf{q}_2, \mathbf{q}_3, \varphi) \in W} \varepsilon \int_\Omega |\varphi| dx + \frac{1}{2} \int_\Omega |f - v|^2 dx, \tag{5.30}$$

where, formally,

$$\begin{aligned} W = \Big\{ & (v, \mathbf{q}_1, \mathbf{q}_2, \mathbf{q}_3, \varphi) \in V \times (L^2(\Omega))^3 \times (L^2(\Omega))^3 \times H_3(\Omega; \mathrm{div}) \times \\ & L^2(\Omega) : \mathbf{q}_1 = (\nabla u, 1), \mathbf{q}_2 = \frac{\mathbf{q}_1}{\|\mathbf{q}_1\|}, \mathbf{q}_3 = \mathbf{q}_2, \varphi = \nabla \cdot \mathbf{q}_3 \Big\}. \end{aligned} \tag{5.31}$$

Above,

$$H_3(\Omega; \mathrm{div}) = \left\{ \mathbf{q} \in (L^2(\Omega))^3 : \nabla \cdot \mathbf{q} \in L^2(\Omega) \right\}. \tag{5.32}$$

They then associated the decomposed minimization problem (5.30) with the following (non-standard) ALF:

$$\begin{aligned} & \mathcal{L}_{\mathbf{r}}(v, \mathbf{q}_1, \mathbf{q}_2, \mathbf{q}_3, \varphi; \boldsymbol{\mu}_1, \mu_2, \boldsymbol{\mu}_3, \mu_4) = \\ & \quad \varepsilon \int_\Omega |\varphi| dx + \frac{1}{2} \int_\Omega |f - v|^2 dx \\ & \quad + \frac{r_1}{2} \int_\Omega \|\mathbf{q}_1 - (\nabla v, 1)\|^2 dx + \int_\Omega \boldsymbol{\mu}_1 \cdot (\mathbf{q}_1 - (\nabla v, 1)) dx \\ & \quad + r_2 \int_\Omega (\|\mathbf{q}_1\| - \mathbf{q}_1 \cdot \mathbf{q}_2) dx + \int_\Omega \mu_2 (\|\mathbf{q}_1\| - \mathbf{q}_1 \cdot \mathbf{q}_2) dx \\ & \quad + \frac{r_3}{2} \int_\Omega \|\mathbf{q}_3 - \mathbf{q}_2\|^2 dx + \int_\Omega \boldsymbol{\mu}_3 \cdot (\mathbf{q}_3 - \mathbf{q}_2) dx \\ & \quad + \frac{r_4}{2} \int_\Omega |\varphi - \partial_{x_1} (\mathbf{q}_3)_1 - \partial_{x_2} (\mathbf{q}_3)_2|^2 dx \\ & \quad + \int_\Omega \mu_4 (\varphi - \partial_{x_1} (\mathbf{q}_3)_1 - \partial_{x_2} (\mathbf{q}_3)_2) dx + \delta_{\mathcal{R}}(\mathbf{q}_2), \end{aligned} \tag{5.33}$$

where $r_i > 0$ for all $i = 1,2,3,4$, $\boldsymbol{\mu}_1, \boldsymbol{\mu}_3 \in (L^2(\Omega))^3$, $\mu_2, \mu_4 \in L^2(\Omega)$, and

$$\delta_{\mathcal{R}}(\mathbf{q}) = \begin{cases} 0, & \mathbf{q} \in \mathcal{R}, \\ +\infty, & \mathbf{q} \notin \mathcal{R}. \end{cases} \tag{5.34}$$

Above, the set $\mathcal{R}$ is defined as

$$\mathcal{R} = \left\{ \mathbf{q} \in (L^2(\Omega))^3 : \|\mathbf{q}\| \le 1 \text{ a.e. in } \Omega \right\}. \tag{5.35}$$

The additional term $\delta_{\mathcal{R}}(\mathbf{q}_2)$ in the ALF (5.33) ensures that the term $\|\mathbf{q}_1\| - \mathbf{q}_1 \cdot \mathbf{q}_2$ is positive almost everywhere in Ω.

Although the objective functional (5.21) is not convex and thus the assumptions enumerated in Theorem 4.1 are not satisfied, the related saddle-point problem was solved using an Algorithm 4.1 style ADMM method. This led to four auxiliary variables and five subproblems, two of which were solved using the fast Fourier transformation method. The authors provided closed-form solutions for the three remaining subproblems. The authors concluded that the proposed method was much faster than the method depicted in [181].

5.3.4 Solution algorithms for the generalized mean curvature model

The three articles that will be discussed in this subsection introduce solution algorithms for the GMC model (5.23) in the case $s = 2$. The numerical results presented in these three articles take $\Phi(x) = x^2$, and thus, the results presented in these three articles are not directly comparable with the results presented in [182] and the included articles [PIV] and [PV]. However, there is no obvious reason why the methods could not be generalized to the L1MC model (5.21).

The solution algorithm presented in [41] relies on a *geometric multigrid algorithm* (see, e.g., [40, 52, 164, 173]) to solve the EL equation (5.25). The authors initially considered a Vogel and Oman style fixed-point iteration scheme [168] as a smoother operator but soon concluded the resulting method was unstable with the GMC model. The authors then proceeded with a so-called *convexity-splitting technique* [67, 68] that leads to a semi-implicit time-marching scheme where the convex part is treated implicitly and the non-convex part is treated explicitly. The technique also introduces an additional differential operator $\mathcal{N}$, the purpose of which is to improve the stability properties of the method. In a general case, the scheme is of the form:

$$\begin{aligned} -\gamma\mathcal{N}(u^{(i+1)}) - \varepsilon\nabla\cdot \left(D_2(u^{(i+1)})\nabla u^{(i+1)} \right) + u^{(i+1)} = \\ -\gamma\mathcal{N}(u^{(i)}) - \varepsilon\nabla\cdot \left(D_1(u^{(i)})\nabla \Phi'(\kappa_{u^{(i)},\beta}) \right) + f, \end{aligned} \tag{5.36}$$

where the functionals D_1 and D_2 are defined in (5.25), and $\gamma > 0$ is a new parameter introduced by the convexity-splitting technique. After considering multiple options, the authors chose the differential operator $\mathcal{N}$ as

$$\mathcal{N}(v) = \nabla\cdot \frac{\nabla v}{\|\nabla v\|}. \tag{5.37}$$

To solve the semi-implicit equation (5.36), the authors first considered a nonlinear Gauss-Seidel-Picard method and a global linearizion approach. After performing *local Fourier analyses* (see, e.g., [164]) and numerical simulations for both alternatives, the authors ended up with an *adaptive smoother* based on the global linearizion approach. The authors concluded their geometric multigrid algorithm combined with the adaptive fixed-point iteration smoother is fast and robust but is subject to certain restrictions regarding the parameter β. The authors noted that $\beta = 10^{-2}$ provides a good balance between the quality of the results and the performance.

In [154], the GMC model (5.23) with $\Phi(x) = x^2/2$ ($\frac{1}{2}$ added for notational reasons) is at first decomposed as follows:

$$(u, \mathbf{p}) = \arg\min_{(v,\mathbf{q})\in W} \frac{\varepsilon}{2}\int_\Omega |\nabla\cdot \mathbf{q}|^2\, dx + \frac{1}{2}\int_\Omega |f - v|^2\, dx, \tag{5.38}$$

where

$$W = \left\{ (v, \mathbf{q}) \in V \times H_2(\Omega; \mathrm{div}) : \mathbf{q} = \frac{\nabla v}{\|\nabla v\|_\beta} \right\}. \tag{5.39}$$

Above,

$$H_2(\Omega; \mathrm{div}) = \left\{ \mathbf{q} \in (L^2(\Omega))^2 : \nabla\cdot \mathbf{q} \in L^2(\Omega) \right\}. \tag{5.40}$$

This yielded a *quadratic penalty method* with the following objective functional:

$$\begin{aligned} j_r(v, \mathbf{q}) = \frac{\varepsilon}{2}\int_\Omega |\nabla\cdot \mathbf{q}|^2\, dx + \frac{1}{2}\int_\Omega |f - v|^2\, dx \\ + \frac{r}{2}\int_\Omega \left\| \nabla v - \mathbf{q}\|\nabla v\|_\beta \right\|^2\, dx. \end{aligned} \tag{5.41}$$

The authors then derived the related EL equations for (5.41):

$$\begin{cases} u - f - r\,\nabla\cdot \left(\nabla u - \|\nabla u\|_\beta \mathbf{p} - \dfrac{\nabla u \cdot \mathbf{p}}{\|\nabla u\|_\beta} \nabla u + (\mathbf{p}\cdot\mathbf{p})\nabla u \right) = 0, \\ r\,\|\nabla u\|_\beta u_{x_1} - \varepsilon\, \partial_{x_1}(\nabla\cdot \mathbf{p}) + r\,\|\nabla u\|_\beta^2\, \mathbf{p}_1 = 0, \\ r\,\|\nabla u\|_\beta u_{x_2} - \varepsilon\, \partial_{x_2}(\nabla\cdot \mathbf{p}) + r\,\|\nabla u\|_\beta^2\, \mathbf{p}_2 = 0, \end{cases} \tag{5.42}$$

with boundary conditions

$$\begin{cases} \nabla\cdot \mathbf{p} = 0, \\ \nabla u \cdot \mathbf{n} = 0, \\ \mathbf{p}\cdot \mathbf{n} = 0, \end{cases} \tag{5.43}$$

where $\mathbf{n}$ denotes the outward unit normal to $\partial\Omega$. Next, the authors proceeded to solve the EL equations (5.42) using a *nonlinear multigrid algorithm*. As with [154], the authors concluded that simple fixed-point schemes are not suitable for the EL equations (5.42). Thus, the authors adopted the convexity-splitting approach and tried to solve the related semi-implicit equations using two *Gauss-Seidel type approaches*. The authors performed local Fourier analyzes and numerical simulations for both approaches and concluded that the obtained smoother operators are efficient for solving the GMC model.

Towards the end of the article, the authors performed extensive comparisons between their nonlinear multigrid algorithm, their adaptation of the AL-based method from [163, 175, 182], and the aforementioned geometric multigrid algorithm [41]. The authors also included the TV (5.11) and TGV (5.15) models in their comparisons. In the end, the authors concluded that their method delivers better quality results and uses less regularization and nonlinearity than the method presented in [41]. In addition, they concluded that the quality of the results produced by their method is comparable to the results produced by the TGV model. The method seemed to perform well, even when $\beta = 10^{-4}$.

The two previous articles made use of the multigrid method and achieved relatively good results. The solution algorithm presented in [177] represents a slightly different approach based on a so-called *homotopy method* [3, 115, 171] and a *relaxed fixed-point iteration*. At first, the authors rewrote the EL equation (5.25) as

$$p(\mathcal{N}_\beta(u), u) = 0, \tag{5.44}$$

where

$$p(\mathcal{N}, v) = 2\varepsilon \mathcal{N} \left(-\nabla \cdot \frac{\nabla v}{\|\nabla v\|_\beta} \right) + v - f, \tag{5.45}$$

and the differential operator $\mathcal{N}_\beta$ is defined as

$$\mathcal{N}_\beta(v) = -\nabla \cdot \left(\frac{1}{\|\nabla v\|_\beta} \left(\mathbf{I}_2 - \frac{\nabla v \nabla v^T}{\|\nabla v\|_\beta^2} \right) \nabla \right). \tag{5.46}$$

The authors then derived a fixed-point method based on the idea of freezing the differential operator $\mathcal{N}_\beta$ for each step and then updating the operator accordingly:

Algorithm 5.1

```
Initialize $u^{(0)} = f$.
Initialize the operator as $\mathcal{N}_\beta^{(0)} = \mathcal{N}_\beta(u^{(0)})$.
for $n = 1, 2, \ldots$ do
    Solve $u^{(i)} \in V$ from $p(\mathcal{N}_\beta^{(i-1)}, u^{(i)}) = 0$.
    Update the operator as $\mathcal{N}_\beta^{(i)} = \mathcal{N}_\beta(u^{(i)})$.
end for
```

Unfortunately, the differential operator $\mathcal{N}_\beta$ is highly singular for small β, which makes the part of solving $u^{(i)}$ in Algorithm 5.1 very expensive. The authors noticed that they could obtain a relaxed fixed-point method by replacing the differential operator $\mathcal{N}_\beta$ with the operator

$$\mathcal{M}_\beta(v) = -\nabla \cdot \left(\frac{1}{\|\nabla v\|_\beta} \nabla \right). \tag{5.47}$$

In addition, the authors advocated a particular *primal dual method* for the solution of $u^{(i)}$. In summary, the modified algorithm is of the form:

Algorithm 5.2

```
  Initialize $u^{(0)} = f$
  Initialize the operator as $\mathcal{M}_\beta^{(0)} = \mathcal{M}_\beta(u^{(0)})$.
  for $n = 1, 2, \ldots$ do
      Solve $u^{(i)} \in V$ from
          $\begin{cases} -2\,\varepsilon\,\mathcal{M}_\beta^{(i-1)} \nabla\cdot \mathbf{w} + u^{(i)} - f = 0 \\ \mathbf{w}\|\nabla u^{(i)}\|_\beta - \nabla u^{(i)} = \mathbf{0}, \end{cases}$
      where $\mathbf{w} : \Omega \to \mathbb{R}^2$ is a differentiable function.
      Update the operator as $\mathcal{M}_\beta^{(i)} = \mathcal{M}_\beta(u^{(i)})$.
  end for
```

In order to even further improve the convergence properties of their method for small β, the authors adopted the homotopy method. The basic idea is to adaptively decrease the value of β by following a predetermined curve. This approach significantly improved the effectiveness of the method, even in the case $\beta = 10^{-6}$.

Remark 5.3 *An alternative approach based on the so-called Bernstein's problem for minimizing the MC of the image was provided in [81, pp. 155 – 159]. The author reported that task of minimizing the MC energy is equivalent to making the image more piecewise linear. Thus, the minimization can be accomplished without explicitly computing the MC. The numerical results indicate that this new approach is three or four orders of magnitude faster than the methods presented in this subsection and in the previous subsection. However, based on preliminary numerical experiments conducted by the author of this dissertation, it appears that this new method does not preserve edges very well and is unable to maintain the image contrast. Thus, the real significance of this new approach remains unclear.*

6 INCLUDED ARTICLES AND RESEARCH CONTRIBUTION

This chapter discusses the included articles in greater detail and highlights the major research contributions of each article. The fast direct solver and GPU computing-related articles [PI], [PII], and [PIII] are treated first, followed by the image denoising and GPU computing-related articles [PIV] and [PV].

The included articles [PI] and [PII] dealt with a special form of separable block tridiagonal linear system

$$\begin{bmatrix} \mathbf{D} & -\mathbf{I}_{n_2} & & \\ -\mathbf{I}_{n_2} & \mathbf{D} & \ddots & \\ & \ddots & \ddots & -\mathbf{I}_{n_2} \\ & & -\mathbf{I} & \mathbf{D} \end{bmatrix} \begin{bmatrix} \mathbf{u}_1 \\ \mathbf{u}_2 \\ \vdots \\ \mathbf{u}_{n_1} \end{bmatrix} = \begin{bmatrix} \mathbf{f}_1 \\ \mathbf{f}_2 \\ \vdots \\ \mathbf{f}_{n_1} \end{bmatrix}, \tag{6.1}$$

where $\mathbf{D} \in \mathbb{R}^{n_2 \times n_2}$, $\mathbf{u}_i, \mathbf{f}_i \in \mathbb{R}^{n_2}$, and $n_1 = 2^{k_1} - 1$ for some positive integer k_1. As noted earlier in Subsection 3.2.1, if $\mathbf{D} = \text{tridiag}\{-1, 4, -1\}$, then this system corresponds to a particular type of two-dimensional Poisson boundary value problem.

6.1 Article [PI]: Fast Poisson solvers for graphics processing units

M. Myllykoski, T. Rossi, and J. Toivanen. Fast Poisson solvers for graphics processing units. In P. Manninen and P. Öster, editors, *Applied Parallel and Scientific Computing*, volume 7782 of *Lecture Notes in Computer Science*, pages 265–279, Springer Berlin Heidelberg: Berlin, Germany, 2013.

In addition to the above limitations, the included article [PI] also assumed that $\mathbf{D} = \text{tridiag}\{-1, 4, -1\}$ (or $\mathbf{B} = \text{tridiag}\{-1, 6, -1\} \in \mathbb{R}^{n_3 \times n_3}$, $n_3 = 2^{k_3} - 1$, $k_3 \in \mathbb{N}$, in (3.12)), and $n_2 = 2^{k_2} - 1$ for some positive integer k_2. The article described GPU implementations of two BCR methods. The first implementation is based on the radix-2 BCR method discussed in Subsection 3.4.2, and the

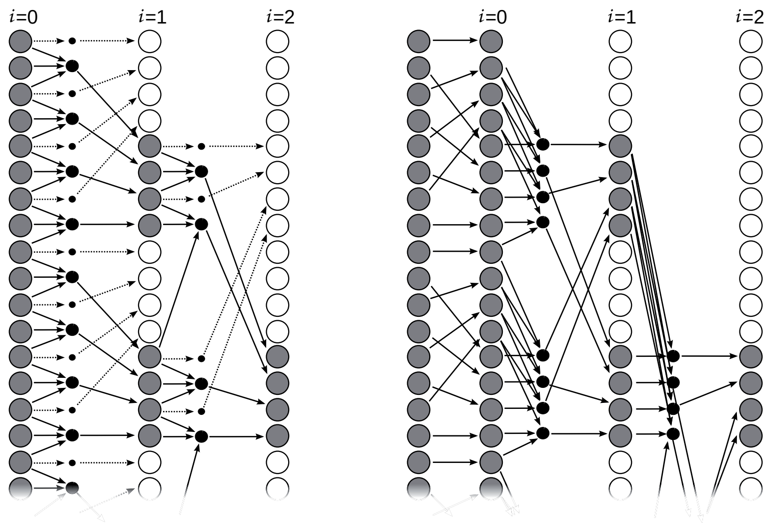


FIGURE 14 Examples of how the global memory (on the left) and the local memory (on the right) data permutation schemes work when the work-group size is four.

second implementation is based on a radix-4 variant of the PSCR method discussed in Section 3.5. The radix-4 algorithm was obtained by explicitly calculating the eigenvalues and eigenvector components associated with the partial solution technique (Theorem 3.1). The resulting radix-4 algorithm is very similar to the radix-2 algorithm but generates a fewer number of tridiagonal linear systems.

The GPU implementations were compared against each other and against equivalent *multithreaded* CPU implementations. Up to sixfold speedups were observed when a Nvidia GeForce GTX580 GPU was compared against a quad-core Intel Core i7 CPU. The numerical experiments were performed using double-precision floating-point arithmetic. The article then investigated the impact of increasing the radix number of the algorithm by comparing the radix-2 and radix-4 methods to each other and concluded that the radix-4 method was more suitable for GPU computation. The article concluded that BCR type methods can offer a sufficient amount of fine-grained parallelism for GPU computation when they are combined with the CR method.

The article introduced a few new ideas related to CR-based tridiagonal linear system solvers. More specifically, the article introduced a permutation scheme that can be used in combination with the CR method and the global memory. This permutation scheme eliminated the memory access pattern fragmentation problem discussed in Subsection 3.3.1 at the cost of some additional global memory traffic. The article also suggested a similar permutation scheme to be used with the local memory in situations where the reduced system contains more even-numbered rows than there are work-items in the work-group. Figure 14 shows examples of how the data permutation schemes operate.

The article provides partial answers to the research questions **RQ1** and **RQ2**. In summary, the primary research contributions of the article were:

1. The article described efficient GPU implementations of two BCR methods.
2. The article showed that GPUs can provide demonstrable benefits in the context of BCR-type fast direct solvers.
3. The article introduced a few new ideas related to tridiagonal linear systems solvers.
4. The article presented evidence that the BCR methods with a higher radix numbers are more suitable for GPU computation.

6.2 Article [PII]: A parallel radix-4 block cyclic reduction algorithm

M. Myllykoski and T. Rossi. A parallel radix-4 block cyclic reduction algorithm. *Numerical Linear Algebra with Applications*, 21(4):540–556, 2014.

The included article [PII] presented an alternative way to derive a radix-4 BCR method for the linear system (6.1). This was accomplished by modifying the radix-2 BCR formulas discussed in Subsection 3.4.1. To put it briefly, two radix-2 reduction steps were combined into a one radix-4 reduction step. Similar but slightly more complicated steps were taken in order to achieve a radix-4 back substitution step. A parallel variant was obtained by using Lemmas 3.4 and 3.5 together with the partial fraction technique (Lemma 3.6). The article demonstrated that the parallel variant of the radix-4 method generates a smaller number of subproblems when compared to the parallel variant of the radix-2 method discussed in Subsection 3.4.2. In that sense, the radix-4 method is less computationally expensive than the radix-2 method. The radix-4 method was also demonstrated to be numerically stable and equivalent to the radix-4 PSCR method in certain special circumstances. The theoretical results were confirmed by numerical experiments.

The article provides partial answers to the research question **RQ1**. In summary, the primary research contributions of the article were:

1. The article described an alternative way to derive a radix-4 BCR method.
2. The article demonstrated that the number of tridiagonal subproblems is reduced when the radix number is increased.
3. The article presented the results of numerical experiments showing that the new radix-4 method is actually faster than the radix-2 method.
4. The article showed that the new method is numerically stable.
5. The article highlighted the equivalence between the new method and the radix-4 PSCR method.

6.3 Article [PIII]: On solving separable block tridiagonal linear systems using a GPU implementation of radix-4 PSCR method

M. Myllykoski, T. Rossi, and J. Toivanen. On solving separable block tridiagonal linear systems using a GPU implementation of radix-4 PSCR method. Submitted to SIAM Journal on Scientific Computing.

The included article [PIII] presented a generalized GPU implementation of the radix-4 PSCR method discussed in Section 3.5. The implementation can be applied to real and complex valued problems of arbitrary size with coefficient matrices of the form (3.1) or (3.2). The article can be considered as a natural extension of the included article [PI]. However, the relaxation of the requirement related to the problem size posed in the included article [PI] complicates the implementation considerably.

The implementation utilizes an updated version of the tridiagonal solver used in [PI]. It uses the CR method and the same permutation schemes but includes additional PCR and Thomas stages, similar to [57, 105]. However, the numerical experiments presented in the article suggested that the Thomas step was actually unnecessary, and it was disabled in most numerical experiments. The performance of the implementation was further improved by using the *Newton-Raphson division algorithm* [70] and an initial guess that leads to full double-precision accuracy with only four iterations, each of which requires two *fused multiply-add* instructions [134]. Additional numerical experiments indicated that this approach more than doubled the division operation throughput on a Nvidia Tesla K40c GPU.

Up to 16-fold speedups were observed when Nvidia GeForce GTX580 and Nvidia Tesla K40c GPUs were compared against a *single-threaded* CPU implementation run on an Intel Xeon CPU. The obtained floating-point performance was extensively analyzed using the *roofline performance analysis model* [174], which takes into account the available (off-chip) memory bandwidth. The obtained models indicated that the implementation was otherwise optimal, but the hybrid tridiagonal solver did not perform quite as well as expected. Various local memory-related limitations were offered as an explanation. The numerical experiments were performed using double-precision floating-point arithmetic.

The article provides partial answers to the research questions **RQ1** and **RQ2**. In summary, the primary research contributions of the article were:

1. The article described an efficient and generalized GPU implementation of the radix-4 PSCR method.
2. The article showed that GPUs can provide demonstrable benefits in the context of more generalized BCR-type fast direct solvers.
3. The article presented extensive roofline analyzes of the presented numerical results, including separate analyzes of the tridiagonal system solver.
4. The article showed that the Newton-Raphson division algorithm can improve the division operation throughput on some GPUs.

6.4 Article [PIV]: A new augmented Lagrangian approach for L^1-mean curvature image denoising

M. Myllykoski, R. Glowinski, T. Kärkkäinen, and T. Rossi. A new augmented Lagrangian approach for L^1-mean curvature image denoising. *SIAM Journal on Imaging Sciences*, 8(1):95–125, 2015.

The included article [PIV] introduced a new AL-based solution algorithm for the L1MC image denoising model discussed in Subsection 5.3.2. The solution algorithm differs from the existing approach presented in [182] to the extent that it uses a different ALF and mixed finite elements [34] instead of finite differences.

The original minimization problem with $\mathcal{J}$ defined by (5.21) was decomposed as

$$(u, \mathbf{p}_1, \mathbf{p}_2, \mathbf{p}_3, \phi) = \arg\min_{(v,\mathbf{q}_1,\mathbf{q}_2,\mathbf{q}_3,\varphi)\in W} \varepsilon \int_\Omega |\varphi| dx + \frac{1}{2}\int_\Omega |f - v|^2 dx, \tag{6.2}$$

where, formally,

$$W = \Bigg\{ (v, \mathbf{q}_1, \mathbf{q}_2, \mathbf{q}_3, \varphi) \in V \times (L^2(\Omega))^2 \times (L^2(\Omega))^2 \times H_2(\Omega; \mathrm{div}) \times L^2(\Omega) \\ : \mathbf{q}_1 = \nabla v,\ \mathbf{q}_2 = \frac{\mathbf{q}_1}{\sqrt{1 + \|\mathbf{q}_1\|^2}},\ \mathbf{q}_3 = \mathbf{q}_2 \text{ and } \varphi = \nabla\cdot \mathbf{q}_3 \Bigg\}. \tag{6.3}$$

Above, $V = H^2(\Omega)$ and $H_2(\Omega; \mathrm{div})$ is defined in (5.40). The non-convex term

$$\mathbf{q}_2 = \frac{\mathbf{q}_1}{\sqrt{1 + \|\mathbf{q}_1\|^2}} \tag{6.4}$$

was handled using a *projection* in

$$\mathbf{E}_{12} = \left\{ (\mathbf{q}_1, \mathbf{q}_2) \in \left(L^2(\Omega)\right)^{2\times 2} : \mathbf{q}_2 = \frac{\mathbf{q}_1}{\sqrt{1 + \|\mathbf{q}_1\|^2}} \right\}. \tag{6.5}$$

Thus, the non-convex term (6.4) does not appear in the resulting ALF, which is the form:

$$\begin{aligned} \mathcal{L}_\mathbf{r}(v, \mathbf{q}_1, \mathbf{q}_2, \mathbf{q}_3, \varphi; \boldsymbol{\mu}_1, \boldsymbol{\mu}_2, \mu_3) = {} & \varepsilon \int_\Omega |\varphi| dx + \frac{1}{2}\int_\Omega |f - v|^2 dx \\ & + \frac{r_1}{2}\int_\Omega \|\nabla v - \mathbf{q}_1\|^2 dx + \int_\Omega \boldsymbol{\mu}_1 \cdot (\nabla v - \mathbf{q}_1) dx \\ & + \frac{r_2}{2}\int_\Omega \|\mathbf{q}_2 - \mathbf{q}_3\|^2 dx + \int_\Omega \boldsymbol{\mu}_2 \cdot (\mathbf{q}_2 - \mathbf{q}_3) dx \\ & + \frac{r_3}{2}\int_\Omega |\nabla\cdot \mathbf{q}_3 - \varphi|^2 dx + \int_\Omega \mu_3(\nabla\cdot \mathbf{q}_3 - \varphi) dx, \end{aligned} \tag{6.6}$$

where $r_i > 0$ for all $i = 1, 2, 3$, $\boldsymbol{\mu}_1, \boldsymbol{\mu}_2 \in (L^2(\Omega))^2$, $\mu_3 \in L^2(\Omega)$, and $(\mathbf{q}_1, \mathbf{q}_2) \in \mathbf{E}_{12}$.

The resulting saddle-point problem was solved using an Algorithm 4.1 style ADMM method. Because the ALF (6.6) contains five variables and two of the auxiliary variables are coupled together in the space $\mathbf{E}_{12}$, the application of Algorithm 4.1 leads to four subproblems that are solved sequentially:

1. a smooth but nonlinear and non-convex minimization problem in $\mathbb{R}^2$ (involves the variables $\mathbf{q}_1$ and $\mathbf{q}_2$),
2. a vector valued linear minimization problem with a positive definite and symmetric coefficient matrix (involves the variable $\mathbf{q}_3$),
3. a purely explicit pointwise treated minimization problem (involves the variable φ), and
4. a scalar valued linear minimization problem with a separable, positive definite and symmetric coefficient matrix (involves the variable v).

The fourth arising subproblem can be solved using the PSCR method discussed in Section 3.5.

The non-convexity of the objective functional (5.21) is effectively transferred to the first subproblem, which is initially solved using Newton's method. The obtained solution candidate is tested against an explicit relation, and if this test fails, the method then proceeds to a one-dimensional version of the subproblem. In this way, the global minimum can be found more reliably, and as a result, edges can be recovered more easily. Numerical experiments indicated that the algorithm is capable of removing considerable amounts of noise with a reasonable number of iterations. When compared to the solution algorithm depicted in [182], this solution algorithm has the benefit of having one less Lagrange multiplier, which makes it easier to adjust of the penalty coefficients. In addition, the equality constraints in the ALF (6.6) are all linear and result in different (and simpler) subproblems.

The article provides a answer to the research question **RQ3**. In summary, the primary research contributions of the article were:

1. The article presented a new AL-based solution algorithm for the L1MC image denoising model that has fewer tunable parameters than the previous method discussed in [182].
2. The article presented the results of numerical experiments demonstrating the performance of the method.
3. The article showed that the method is capable of removing considerable amounts of noise and can preserve edges very well.

6.5 Article [PV]: A GPU-accelerated augmented Lagrangian based L^1-mean curvature image denoising algorithm implementation

M. Myllykoski, R. Glowinski, T. Kärkkäinen, and T. Rossi. A GPU-accelerated augmented Lagrangian based L^1-mean curvature image denoising algorithm implementation. In M. Gavrilova, V. Skala, editors, *WSCG 2015: 23rd International Conference in Central Europe on Computer Graphics, Visualization and Computer Vision'2015: Full Papers Proceedings*, pages 119–128, Vaclav Skala – Union Agency, 2015.

The included article [PV] describes a GPU implementation of the solution algorithm depicted in [PIV]. Because two of the arising subproblems can be solved without inter-process communication, GPU-acceleration brought significant performance benefits because the non-convex term in the original objective functional is treated by one that these two subproblems. The remaining subproblems were solved using the radix-4 PSCR GPU implementation depicted in [PIII] and a conjugate gradient method which can be implemented relatively efficiently on GPUs [6, 35, 90]. The article also presented an analysis of how certain data structures should be stored in the global memory.

The new GPU implementation was compared against a *single-threaded* CPU implementation. Up to 33-fold speedups were observed when Nvidia GeForce GTX580 and Nvidia Tesla K40c GPUs were compared against an Intel Xeon CPU. The first subproblem was solved up to 76 times faster. More global memory intensive subproblems did not benefit quite as much from the GPU-acceleration. The numerical experiments were performed using double-precision floating-point arithmetic.

The article provides a answer to the research question **RQ4**. In summary, the primary research contributions of the article were:

1. The article described an efficient GPU implementation of the L1MC image denoising algorithm described in the included article [PIV].
2. The article showed that GPUs can provide demonstrable benefits in the context of the higher order variational-based image denoising algorithms.
3. The article presented an analysis of how certain data structures should be stored in the global memory.

7 CONCLUSION

This dissertation had three main themes: BCR type fast direct solvers, GPU computation, and L1MC-based image denoising. The BCR methods have many uses, ranging from solving a simple Poisson boundary value problem in a rectangle to solving a set of subproblems generated by a different numerical method. GPUs can provide significant performance benefits for the aforementioned BCR methods. Image denoising, on the other hand, is a suitable application area for the GPU implementations of the BCR methods.

A total of five articles were included in this article-type dissertation. The first three articles dealt with BCR methods. The first article presented GPU implementations of two simplified BCR methods and compared the implementations against similar CPU implementations. The second article introduced a new way of deriving a GPU-friendly radix-4 BCR method. The third article presented a GPU implementation of a radix-4 PSCR method. Reasonable speedups were obtained in the first and the third articles, although the generalized implementation presented in the third article did not perform quite as well as expected on newer GPUs. The implementation would benefit from a more GPU-friendly tridiagonal solver. Otherwise, these articles showed that GPUs can provide demonstrable benefits in the context of BCR type fast direct solvers.

The two remaining articles focused on the L1MC model. The fourth article introduced a new AL-based solution algorithm and demonstrated the performance of the proposed algorithm by numerical means. The fifth article presented an efficient GPU implementation of the method. Significant performance benefits were obtained, although some parts of the implementation did not benefit quite as much as expected. Otherwise, these articles show that GPUs can provide demonstrable benefits in the context of higher order image denoising models and ADMM methods.

8 FUTURE WORK

In future, the generalized radix-4 PSCR implementations described in the included article [PIII] could be rewritten using CUDA [128], or the implementation could have a support for both frameworks. This would allow the implementation to access more memory because Nvidia's current OpenCL implementation supports only up to 4 GB of global memory. Furthermore, AMD/ATI GPUs should be tested with the implementation, and support for multiple GPUs would be useful. The implementation was written with a multi-GPU support in mind, but some functionality is missing.

The AL-based L1MC image denoising algorithm described in the included article [PIV] could be generalized. Adopting the method to the GMC model (5.23) would require modifications to three subproblems:

Arbitrary function Φ: The third subproblem must be modified. The case $\Phi(x) = x^2$ yields a closed-form solution for the subproblem and requires only minor modifications.

Arbitrary parameter β: The first subproblem must be modified. The required modifications are trivial, but the behavior of the whole algorithm changes drastically in some cases.

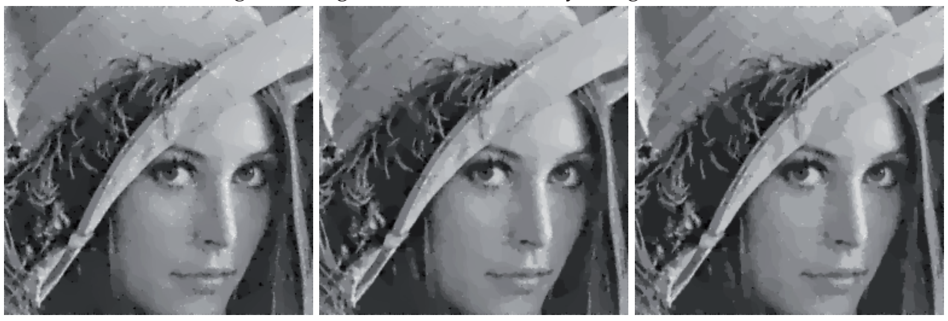
Arbitrary parameter s: The fourth subproblem must be modified. The subproblem becomes nonlinear, which means that the PSCR method cannot be applied directly. However, an operator splitting-based time-marching technique was proposed by Professor Roland Glowinski, and this technique would utilize the PSCR method.

8.1 Preliminary results

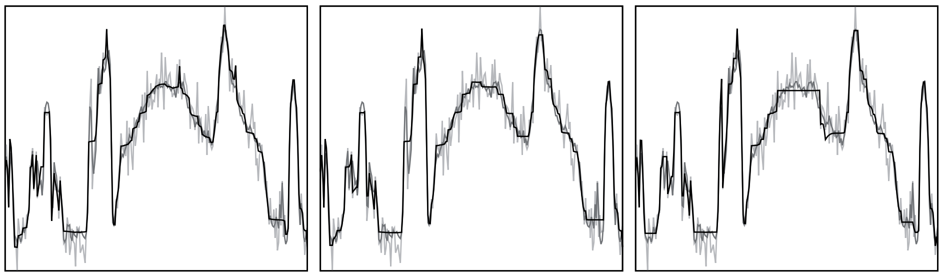
Figures 15 and 17 show some preliminary results in the case $\Phi(x) = |x|$ and $s = 2$. These proof-of-concept style results show that the method at least converges and produces visually pleasing results. The method actually seems to converge slightly faster when the value of the parameter β was reduced. Making the parameter β smaller also seems to reduce the number of artifacts in the result (the narrow peaks) but makes the result more piecewise constant. The original test image was scaled to the range $[0, 1]$, and the noisy input image contained normally distributed zero-mean noise with the standard deviation $\sigma = 0.05$. The parameters were: $h = 0.005$ (mesh refinement), $r_0 = 0.001$, $\varepsilon = r_0 h$, $r_1 = 50 r_0 h$, $r_2 = 5 r_0$, and $r_3 = 5 r_0 h^2$.

Figures 16 and 18 show similar preliminary results in the case $\Phi(x) = x^2$ and $s = 2$. The method converges, but the convergence rate was negatively effected when the value of the parameter β was reduced. However, the method could be competitive with the other methods discussed in Subsection 5.3.4 after a proper adjustment of the parameters. Again, making the parameter β smaller makes the result more piecewise constant. The parameters were: $h = 0.005$, $r_0 = 0.001$, $\varepsilon = r_0 h^2$, $r_1 = 50 r_0 h$, $r_2 = 5 r_0$, and $r_3 = 5 r_0 h^2$.

(a) Original image. (b) Noisy image.



(c) Result, $\beta = 10^0$. (d) Result, $\beta = 10^{-3}$. (e) Result, $\beta = 10^{-6}$.

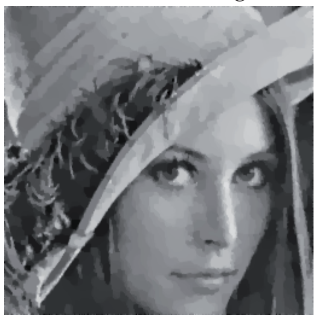


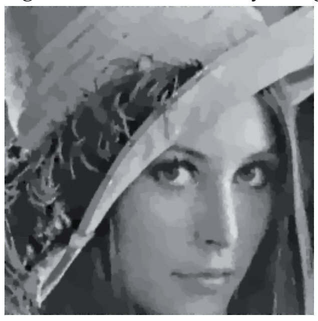
(f) Cross-section, $\beta = 10^0$. (g) Cross-section, $\beta = 10^{-3}$. (h) Cross-section, $\beta = 10^{-6}$.

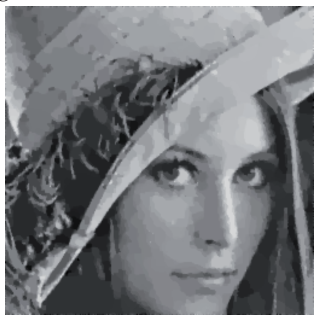
FIGURE 15 The results obtained with the GMC model ($\Phi(x) = |x|$ and $s = 2$).

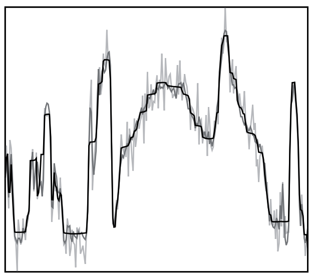
(a) Original image.

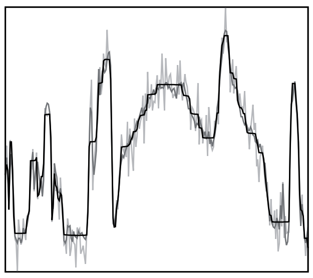
(b) Noisy image.


(c) Result, $\beta = 10^0$.


(d) Result, $\beta = 10^{-3}$.


(e) Result, $\beta = 10^{-6}$.


(f) Cross-section, $\beta = 10^0$.


(g) Cross-section, $\beta = 10^{-3}$.

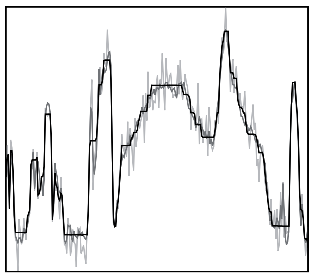

(h) Cross-section, $\beta = 10^{-6}$.

FIGURE 16 The results obtained with the GMC model ($\Phi(x) = x^2$ and $s = 2$.)

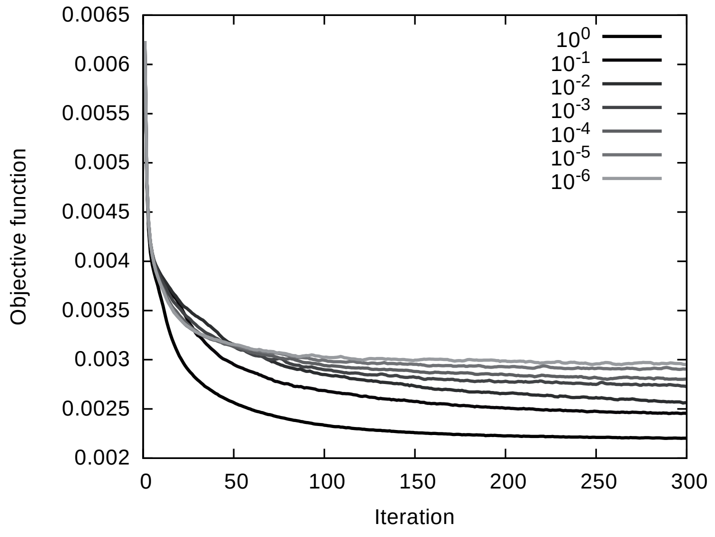


FIGURE 17 Values of the objective functional (5.23) ($\Phi(x) = |x|$, $s = 2$) among the iterations when the value of the parameter β is varied.

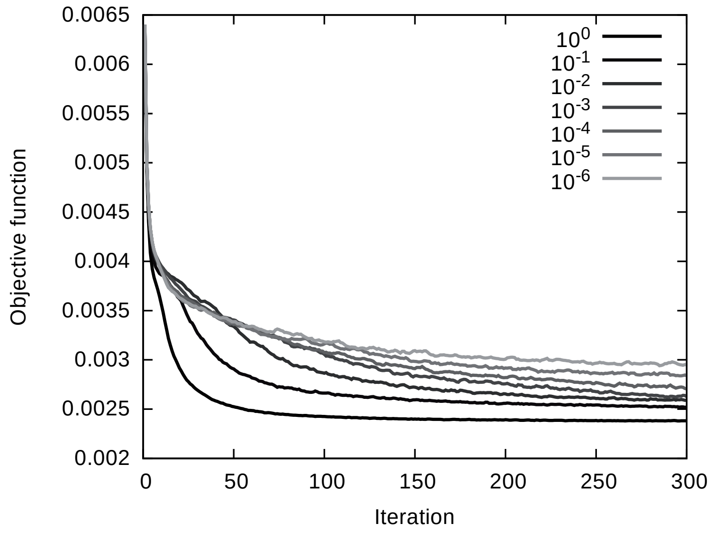


FIGURE 18 Values of the objective functional (5.23) ($\Phi(x) = x^2$, $s = 2$) among the iterations when the value of the parameter β is varied.

YHTEENVETO (FINNISH SUMMARY)

Tällä väitöskirjalla, *GPU-kiihdytetyt nopeat suorat ratkaisijat ja niiden sovellukset häiriön poistamisessa valokuvista*, on kolme pääasiallista teemaa: lohkosyklinen reduktio (*block cyclic reduction*, BCR) -pohjaiset nopeat suorat ratkaisijat, näytönohjainlaskenta ja keskimääräisen kaarevuuden (*mean curvature*) L^1-normiin perustuvat häiriönpoistomenetelmät. BCR-menetelmillä on monia käyttökohteita kuten esimerkiksi yksinkertaisen Poissonin reuna-arvotehtävän ratkaiseminen suorakulmiossa. Lisäksi menetelmille soveltuvia osatehtäviä syntyy monimutkaisempien numeeristen menetelmien yhteydessä. Modernit ohjelmoitavat näytönohjaimet (GPU) kykenevät tarjoamaan huomattavasti enemmän liukulukulaskentatehoa kuin perinteiset keskusprosessorit. Täten GPU:den hyödyntämisestä saattaisi olla hyötyä myöskin BCR-menetelmien yhteydessä. Häiriönpoistomenetelmät puolestaan tarjoavat erinomaisen sovellusalueen BCR-menetelmien näytönohjainpohjaisille toteutuksille.

Tähän artikkelikokoelmaan sisältyy yhteensä viisi tieteellistä artikkelia. Kolme ensimmäistä artikkelia käsittelivät BCR-menetelmiä. Ensimmäinen artikkeli esitteli kahden yksinkertaistetun BCR-menetelmän näytönohjainpohjaiset toteutukset. Toinen artikkeli esitteli uuden tavan johtaa näytönohjaimelle soveltuva 4-kantainen BCR-menetelmä. Kolmas artikkeli esitteli yleistetyn BCR-menetelmän näytönohjainpohjaisen toteutuksen. Ensimmäisessä ja kolmannessa artikkelissa raportoitiin kohtalaisen hyviä suorituskykylisäyksiä, mutta kolmannessa artikkelissa esitelty yleistetty toteutus ei suoriutunut täysin odotuksien mukaisesti uudemmilla näytönohjaimilla. Paremmin näytönohjaimelle soveltuva tridiagonaaliratkaisija saattaisi parantaa toteutuksen suorituskykyä.

Kaksi jälkimmäistä artikkelia käsittelivät keskimääräisen kaarevuuden L^1-normiin perustuvaa häiriönpoistomallia. Neljäs artikkeli esitteli laajennettuun Lagrangen funktionaaliin perustuvan menetelmän mallin ratkaisemiseksi ja osoitti menetelmän toimivuuden numeerisin keinoin. Viides artikkeli esitteli näytönohjainpohjaisen toteutuksen kyseisestä menetelmästä. Näytönohjainpohjaisen toteutuksen avulla päästiin merkittäviin nopeushyötyihin, mutta osa toteutuksen osatehtäväratkaisijoista ei hyötynyt aivan niin paljon kuin oli odotettavissa.

Väitöskirjaan liitetyt julkaisut osoittavat, että GPU:sta on oleellista hyötyä BCR-menetelmien, korkeamman asteen häiriönpoistomallien ja laajennettuun Lagrangen funktionaaliin perustuvien menetelmien yhteydessä.

REFERENCES

[1] A. A. Abakumov, A. Yu. Yeremin, and Yu. A. Kuznetsov. Efficient fast direct method of solving Poisson's equation on a parallelepiped and its implementation in an array processor. *Soviet Journal of Numerical Analysis and Mathematical Modelling*, 3:1–20, 1988.

[2] R. Acar and C. R. Vogel. Analysis of bounded variation penalty methods for ill-posed problems. *Inverse Problems*, 10(6):1217–1229, 1994.

[3] E. L. Allgower and K. Georg. Numerical path following. In P. G. Ciarlet and J. L. Lions, editors, *Techniques of Scientific Computing (Part 2)*, volume 5 of *Handbook of Numerical Analysis*, pages 3 – 207. Elsevier, 1997.

[4] L. Ambrosio and S. Masnou. A direct variational approach to a problem arising in image reconstruction. *Interfaces and free boundaries*, 5(1):63–81, 2003.

[5] L. Ambrosio and S. Masnou. On a variational problem arising in image reconstruction. In P. Colli, C. Verdi, and A. Visintin, editors, *Free Boundary Problems*, volume 147 of *ISNM International Series of Numerical Mathematics*, pages 17–26. Birkhäuser Basel, 2004.

[6] M. Ament, G. Knittel, D. Weiskopf, and W. Strasser. A parallel preconditioned conjugate gradient solver for the Poisson problem on a multi-GPU platform. In *18th Euromicro International Conference on Parallel, Distributed and Network-Based Processing (PDP)*, pages 583–592. IEEE, 2010.

[7] P. Amodio and M. Paprzycki. A cyclic reduction approach to the numerical solution of boundary value ODEs. *SIAM Journal on Scientific Computing*, 18(1):56–68, 1997.

[8] F. Andreu, C. Ballester, V. Caselles, and J. M. Mazón. Minimizing total variation flow. *Comptes Rendus de l'Académie des Sciences - Series I - Mathematics*, 331(11):867 – 872, 2000.

[9] A. Z. Averbuch, P. Neittaanmäki, and V. A. Zheludev. *Spline and Spline Wavelet Methods with Applications to Signal and Image Processing, Volume I: Periodic Splines*. Springer Netherlands, 1 edition, 2014.

[10] A. Banegas. Fast Poisson solvers for problems with sparsity. *Mathematics of Computations*, 32:441–446, 1978.

[11] A. Bayliss, M. Gunzburger, and E. Turkel. Boundary conditions for the numerical solution of elliptic equations in exterior regions. *SIAM Journal on Applied Mathematics*, 42(2):430–451, 1982.

[12] M. Belge, M. E. Kilmer, and E. L. Miller. Wavelet domain image restoration with adaptive edge-preserving regularization. *IEEE Transactions on Image Processing*, 9(4):597–608, 2000.

[13] G. Bellettini, V. Caselles, and M. Novaga. The total variation flow in RN. *Journal of Differential Equations*, 184(2):475 – 525, 2002.

[14] A. Ben Hamza and H. Krim. Image denoising: a nonlinear robust statistical approach. *IEEE Transactions on Signal Processing*, 49(12):3045–3054, 2001.

[15] J. D. Benamou and Y. Brenier. A computational fluid mechanics solution to the Monge-Kantorovich mass transfer problem. *Numerische Mathematik*, 84:375–393, 2000.

[16] J. Berenger. A perfectly matched layer for the absorption of electromagnetic waves. *Journal of Computational Physics*, 114(2):185 – 200, 1994.

[17] J. Berenger. Three-dimensional perfectly matched layer for the absorption of electromagnetic waves. *Journal of Computational Physics*, 127(2):363 – 379, 1996.

[18] A. L. Bertozzi and J. B. Greer. Low-curvature image simplifiers: Global regularity of smooth solutions and Laplacian limiting schemes. *Communications on Pure and Applied Mathematics*, 57(6):764–790, 2004.

[19] D. P. Bertsekas. *Constrained Optimization and Lagrange Multiplier Methods*. Academic Press, 1982.

[20] B. Bialecki, G. Fairweather, and A. Karageorghis. Matrix decomposition algorithms for elliptic boundary value problems: a survey. *Numerical Algorithms*, 56:253–295, 2011.

[21] D. A. Bini, G. Fiorentino, L. Gemignani, and B. Meini. Effective fast algorithms for polynomial spectral factorization. *Numerical Algorithms*, 34(2–4):217–227, 2003.

[22] D. A. Bini, L. Gemignani, and B. Meini. Factorization of analytic functions by means of Koenig's theorem and Toeplitz computations. *Numerische Mathematik*, 89(1):49–82, 2001.

[23] D. A. Bini, L. Gemignani, and B. Meini. Computations with infinite Toeplitz matrices and polynomials. *Linear Algebra and its Applications*, 343/344:21–61, 2002. Special issue on structured and infinite systems of linear equations.

[24] D. A. Bini, B. Iannazzo, B. Meini, and F. Poloni. Nonsymmetric algebraic Riccati equations associated with an M-matrix: recent advances and algorithms. In D. A. Bini, B. Meini, V. Ramaswami, M. Remiche, and

P. Taylor, editors, *Numerical Methods for Structured Markov Chains*, number 07461 in Dagstuhl Seminar Proceedings, Dagstuhl, Germany, 2008. Internationales Begegnungs- und Forschungszentrum für Informatik (IBFI), Schloss Dagstuhl, Germany.

[25] D. A. Bini and B. Meini. On cyclic reduction applied to a class of Toeplitz-like matrices arising in queuing problems. In W. J. Stewart, editor, *Computations with Markov Chains*, pages 21–38. Kluwer Academic Publishers, 1995.

[26] D. A. Bini and B. Meini. On the solution of a nonlinear matrix equation arising in queueing problems. *SIAM Journal on Matrix Analysis and Applications*, 17(4):906–926, 1996.

[27] D. A. Bini and B. Meini. Improved cyclic reduction for solving queueing problems. *Numerical Algorithms*, 15(1):57–74, 1997.

[28] D. A. Bini and B. Meini. Effective methods for solving banded Toeplitz systems. *SIAM Journal on Matrix Analysis and Applications*, 20(3):700–719, 1999.

[29] D. A. Bini and B. Meini. The cyclic reduction algorithm: from Poisson equation to stochastic processes and beyond. *Numerical Algorithms*, 51(1):23–60, 2008.

[30] D. A. Bini, B. Meini, and V. Ramaswami. Analyzing M/G/1 paradigms through QBDs: the role of the block structure in computing the matrix G. In G. Latouche and P. Taylor, editors, *Proceedings of the Third Conference on Matrix Analytic Methods*, Advances in Algorithmic Methods for Stochastic Models, pages 73–86, NJ, USA, 2000. Notable Publications.

[31] F. Bleichrodt, R. H. Bisseling, and H. A. Dijkstra. Accelerating a barotropic ocean model using a GPU. *Ocean Modelling*, 41(0):16–21, 2012.

[32] P. Blomgren. *Total variation methods for restoration of vector valued images*. PhD thesis, University of California, Los Angeles, 1998.

[33] P. Blomgren, T. F. Chan, P. Mulet, and C. K. Wong. Total variation image restoration: numerical methods and extensions. In *Proceedings of International Conference on Image Processing*, volume 3, pages 384–387, 1997.

[34] D. Boffi, F. Brezzi, and M. Fortin. *Mixed Finite Element Methods and Applications*. Springer-Verlag Berlin Heidelberg, 2013.

[35] J. Bolz, I. Farmer, E. Grinspun, and Schröoder. Sparse matrix solvers on the GPU: conjugate gradients and multigrid. *ACM Transactions on Graphics*, 22(3):917–924, 2003.

[36] S. Boyd, N. Parikh, E. Chu, B. Peleato, and J. Eckstein. Distributed optimization and statistical learning via the alternating direction method of multipliers. *Foundation and Trends in Machine Learning*, 3(1):1–122, 2011.

[37] K. Bredies, K. Kunisch, and T. Pock. Total generalized variation. *SIAM Journal on Imaging Sciences*, 3(3):492–526, 2010.

[38] L. M. Bregman. The relaxation method of finding the common point of convex sets and its application to the solution of problems in convex programming. *USSR Computational Mathematics and Mathematical Physics*, 7(3):200 – 217, 1967.

[39] X. Bresson and T. F. Chan. Fast dual minimization of the vectorial total variation norm and applications to color image processing. *Inverse Problems and Imaging*, 2(4):455 – 484, 2008.

[40] W. Briggs. *A Multigrid Tutorial*. SIAM, Philadelphia, 1987.

[41] C. Brito-Loeza and K. Chen. Multigrid algorithm for high order denoising. *SIAM Journal on Imaging Sciences*, 3(3):363–389, 2010.

[42] A. Buades, B. Coll, and J. M. Morel. A review of image denoising algorithms, with a new one. *Multiscale Modeling & Simulation*, 4(2):490–530, 2005.

[43] O. Buneman. A compact non-iterative Poisson solver. Technical report 294, Institute for Plasma Research, Stanford University, Stanford, CA, 1969.

[44] A. Chambolle. An algorithm for total variation minimization and applications. *Journal of Mathematical Imaging and Vision*, 20(1-2):89–97, 2004.

[45] A. Chambolle and P. Lions. Image recovery via total variation minimization and related problems. *Numerische Mathematik*, 76(2):167–188, 1997.

[46] T. Chan, S. Esedoglu, and A. Yip. Total variation image restoration: Overview and recent developments. In *Proceedings of the 1997 IEEE International Conference on Image Processing*, volume 3, pages 384–387, 1997.

[47] T. Chan, A. Marquina, and P. Mulet. High-order total variation-based image restoration. *SIAM Journal on Scientific Computing*, 22(2):503–516, 2000.

[48] T. F. Chan, G. H. Golub, and P. Mulet. A nonlinear primal-dual method for total variation-based image restoration. *SIAM Journal on Scientific Computing*, 20(6):1964–1977, 1999.

[49] T. F. Chan, S. H. Kang, and J. Shen. Total variation denoising and enhancement of color images based on the CB and HSV color models. *Journal of Visual Communication and Image Representation*, 12(4):422 – 435, 2001.

[50] T. F. Chan, S. H. Kang, and J. Shen. Euler's elastica and curvature-based inpainting. *SIAM Journal on Applied Mathematics*, 63(2):564–592, 2002.

[51] S. G. Chang, Bin Y., and M. Vetterli. Adaptive wavelet thresholding for image denoising and compression. *IEEE Transactions on Image Processing*, 9(9):1532–1546, 2000.

[52] K. Chen. *Matrix Preconditioning Techniques and Applications*. Cambridge University Press, Cambridge, 2005.

[53] Y. Chen, W. Hager, F. Huang, D. Phan, X. Ye, and W. Yin. Fast algorithms for image reconstruction with application to partially parallel MR imaging. *SIAM Journal on Imaging Sciences*, 5(1):90–118, 2012.

[54] S. Cho, Y. Matsushita, and S. Lee. Removing non-uniform motion blur from images. In *11th International Conference on Computer Vision, ICCV 2007*, pages 1–8, 2007.

[55] S. D. Conte and C. de Boor. *Elementary numerical analysis: an algorithmic approach*. International series in pure and applied mathematics. McGraw-Hill, New York, Montreal, third edition, 1980.

[56] C. M. Costa, G. Haase, M. Liebmann, A. Neic, and G. Plank. Stepping into fully GPU accelerated biomedical applications. In I. Lirkov, S. Margenov, and J. Waśniewski, editors, *Large-Scale Scientific Computing*, volume 8353 of *Lecture Notes in Computer Science*, pages 3–14. Springer Berlin Heidelberg, 2014.

[57] A. Davidson, Y. Zhang, and J. D. Owens. An auto-tuned method for solving large tridiagonal systems on the GPU. In *Proceedings of the 25th IEEE International Parallel and Distributed Processing Symposium*, pages 956–965. IEEE, 2011.

[58] M. Desai, D. Kennedy, R. Mangoubi, J. Shah, C. Karl, N. Markis, and A. Worth. Diffusion tensor model based smoothing. In *IEEE International Symposium on Biomedical Imaging, 2002*, pages 705–708, 2002.

[59] M. A. Diamond and D. L. V. Ferreira. On a cyclic reduction method for the solution of Poisson's equations. *SIAM Journal on Numerical Analysis*, 13(1):54–70, 1976.

[60] D. C. Dobson and F. Santosa. Recovery of blocky images from noisy and blurred data. *SIAM Journal on Applied Mathematics*, 56(4):1181–1198, 1996.

[61] D. L. Donoho. De-noising by soft-thresholding. *IEEE Transactions on Information Theory*, 41(3):613–627, 1995.

[62] D. L. Donoho and I. M. Johnstone. Adapting to unknown smoothness via wavelet shrinkage. *Journal of the American Statistical Association*, 90(432):1200–1224, 1995.

[63] D. L. Donoho and J. M. Johnstone. Ideal spatial adaptation by wavelet shrinkage. *Biometrika*, 81(3):425–455, 1994.

[64] S. Durand and M. Nikolova. Restoration of wavelet coefficients by minimizing a specially designed objective function. In *Proceedings of IEEE Workshop on Variational and Level Set Methods in Computer Vision*, 2003.

[65] H.-L. Eng and K.-K. Ma. Noise adaptive soft-switching median filter. *IEEE Transactions on Image Processing*, 10(2):242–251, 2001.

[66] B. Engquist and A. Majda. Absorbing boundary conditions for the numerical simulation of waves. *Mathematics of Computations*, 31(139):629–651, 1977.

[67] D. J. Eyre. Unconditionally gradient stable time marching the Cahn-Hilliard equation. In *Symposia BB Computational & Mathematical Models of Microstructural Evolution*, volume 529 of *MRS Proceedings*, 1998.

[68] D. J. Eyre. An unconditionally stable one-step scheme for gradient systems. Available online from http://www.math.utah.edu/eyre/research/methods/stable.ps, 1998.

[69] M. A. T. Figueiredo and R. D. Nowak. Wavelet-based image estimation: an empirical Bayes approach using Jeffrey's noninformative prior. *IEEE Transactions on Image Processing*, 10(9):1322–1331, 2001.

[70] M. Flynn. On division by functional iteration. *IEEE Transactions on Computers*, C-19(8):702–706, 1970.

[71] M. Fortin and R. Glowinski. *Augmented Lagrangian Methods: Applications to the Numerical Solution of Boundary Value Problems*. North-Holland, Amsterdam, 1983.

[72] W. Gander and G. H. Golub. Cyclic reduction — history and applications. In *Scientific computing*, pages 73–85. Springer, Singapore, 1997.

[73] G. Gilboa and S. Osher. Nonlocal operators with applications to image processing. *Multiscale Modeling & Simulation*, 7(3):1005–1028, 2009.

[74] D. Givoli. Non-reflecting boundary conditions. *Journal of Computational Physics*, 94(1):1 – 29, 1991.

[75] R. Glowinski. *Numerical Methods for Nonlinear Variational Problems*. Springer, New York, NY, 1984.

[76] R. Glowinski. Finite element methods for imcompressible viscous flow. In *Handbook of Numerical Analysis*, volume IX, pages 3–1176. North-Holland, Amsterdam, 2003.

[77] R. Glowinski, T. Kärkkäinen, T. Valkonen, and A. Ivannikov. Non-smooth sor for l1-fitting: Convergence study and discussion of related issues. *Journal of Scientific Computing*, 37(2):103–138, November 2008.

[78] R. Glowinski and P. Le Tallec. *Augmented Lagrangian and Operator-Splitting Methods in Nonlinear Mechanics*. SIAM, Philadelphia, 1989.

[79] D. Göddeke and R. Strzodka. Cyclic reduction tridiagonal solvers on GPUs applied to mixed precision multigrid. *IEEE Transactions on Parallel and Distributed Systems, Special issue: High performance computing with accelerators*, 22(1):22–32, 2011.

[80] T. Goldstein and S. Osher. The split Bregman method for L1-regularized problems. *SIAM Journal on Imaging Sciences*, 2(2):323–343, 2009.

[81] Y. Gong. *Spectrally regularized surfaces*. PhD thesis, ETH Zurich, Nr. 22616, 2015. Available online from http://dx.doi.org/10.3929/ethz-a-010438292.

[82] R. C. Gonzales and R. E. Woods. *Digital Image Processing*. MA: Addison-Wesley, 1993.

[83] J. B. Greer and A. L. Bertozzi. Traveling wave solutions of fourth order PDEs for image processing. *SIAM Journal on Mathematical Analysis*, 36(1):38–68, 2004.

[84] J. B. Greer, A. L. Bertozzi, and G. Sapiro. Fourth order partial differential equations on general geometries. *Journal of Computational Physics*, 216:216–246, 2006.

[85] C.-H. Guo. Efficient methods for solving a nonsymmetric algebraic Riccati equation arising in stochastic fluid models. *Journal of Computational and Applied Mathematics*, 192(2):353–373, 2006.

[86] G. Haase, M. Liebmann, C. Douglas, and G. Plank. A parallel algebraic multigrid solver on graphics processing units. In W. Zhang, Z. Chen, C. Douglas, and W. Tong, editors, *High Performance Computing and Applications*, volume 5938 of *Lecture Notes in Computer Science*, pages 38–47. Springer Berlin Heidelberg, 2010.

[87] G. Haase, M. Schanz, and S. Vafai. Computing boundary element method's matrices on GPU. In I. Lirkov, S. Margenov, and J. Waśniewski, editors, *Large-Scale Scientific Computing*, volume 7116 of *Lecture Notes in Computer Science*, pages 343–350. Springer Berlin Heidelberg, 2012.

[88] M. Harris, S. Sengupta, and J. D. Owens. Parallel prefix sum (scan) with CUDA. In H. Nguyen, editor, *GPU Gems 3*. Addison-Wesley Professional, 2007.

[89] E. Heikkola, T. Rossi, and J. Toivanen. Fast direct solution of the Helmholtz equation with a perfectly matched layer or an absorbing boundary condition. *International Journal for Numerical Methods in Engineering*, 57(14):2007–2025, 2003.

[90] R. Helfenstein and J. Koko. Parallel preconditioned conjugate gradient algorithm on GPU. *Journal of Computational and Applied Mathematics*, 236(15):3584–3590, 2012.

[91] D. Heller. Some aspects of the cyclic reduction algorithm for block tridiagonal linear systems. *SIAM Journal on Numerical Analysis*, 13:484–496, 1976.

[92] M. Hintermüller, C. N. Rautenberg, and J. Hahn. Functional-analytic and numerical issues in splitting methods for total variation-based image reconstruction. *Inverse Problems*, 30(5):055014, 2014.

[93] R. W. Hockney. A fast direct solution of Poisson's equation using Fourier analysis. *Journal of the ACM*, 12:95–113, 1965.

[94] R. W. Hockney. Potential calculation and some applications. *Methods in Computational Physics*, 9:135–511, 1970.

[95] R. W. Hockney and C. R. Jesshope. *Parallel Computers: Architecture, Programming and Algorithms*. Bristol : Adam Hilger, 1981.

[96] P. J. Huber. *Robust Statistics*. Wiley Series in Probability and Mathematical Statistics. New York: Wiley, 1981.

[97] W.-M. Hwu, editor. *GPU Computing GEMs — Emerald Edition*. Applications of GPU Computing Series. Morgan Kaufmann, 1 edition, 2011.

[98] W.-M. Hwu, editor. *GPU Computing GEMs — Jade Edition*. Applications of GPU Computing Series. Morgan Kaufmann, 1 edition, 2011.

[99] K. Ito and K. Kunisch. An active set strategy based on the augmented lagrangian formulation for image restoration. *Mathematical Modelling and Numerical Analysis*, 33(1):1–21, 1999.

[100] K. Ito and K. Kunisch. Bv-type regularization methods for convoluted objects with edge, flat and grey scales. *Inverse Problems*, 16(4):909, 2000.

[101] K. Ito and K. Kunisch. *Lagrange Multiplier Approach to Variational Problems and Applications*. Society for Industrial and Applied Mathematics, 2008.

[102] A. Jain. *Fundamentals of Digital Image Processing*. New Jersey: Prentice Hall, 1989.

[103] M. Kass, A. Lefohn, and J. Owens. Interactive depth of field using simulated diffusion on a GPU. Available as Pixar Technical Memo 06-01, 2006.

[104] Khronos OpenCL Working Group. OpenCL 1.2 API and C Language Specification, revision 19. Available online from https://www.khronos.org/registry/cl/.

[105] H.-S. Kim, S. Wu, L. Chang, and W. W. Hwu. A scalable tridiagonal solver for GPUs. *2011 International Conference on Parallel Processing (ICPP)*, pages 444–453, 2011.

[106] Yu. A. Kuznetsov. Numerical methods in subspaces. *Vychislitel'-nye Processy i Sistemy II*, 37:265–350, 1985.

[107] Yu. A. Kuznetsov and A. M. Matsokin. On partial solution of systems of linear algebraic equations. *Soviet Journal of Numerical Analysis and Mathematical Modelling*, 4:453–468, 1989.

[108] Yu. A. Kuznetsov and T. Rossi. Fast direct method for solving algebraic systems with separable symmetric band matrices. *East-West Journal of Numerical Mathematics*, 4:53–68, 1996.

[109] T. Kärkkäinen, K. Kunisch, and K. Majava. Denoising of smooth images using l1-fitting. *Computing*, 74(4):353–376, 2005.

[110] T. Kärkkäinen and K. Majava. Nonmonotone and monotone active-set methods for image restoration, part 1: Convergence analysis. *Journal of Optimization Theory and Applications*, 106:61–80, 2000.

[111] T. Kärkkäinen and K. Majava. Nonmonotone and monotone active-set methods for image restoration, part 2: Numerical results. *Journal of Optimization Theory and Applications*, 106:81–105, 2000.

[112] T. Kärkkäinen and K. Majava. Semi-adaptive, convex optimisation methodology for image denoising. *IEE Proceedings on Vision, Image and Signal Processing*, 152(5):553–560, 2005.

[113] T. Kärkkäinen, K. Majava, and M. M. Mäkelä. Comparison of formulations and solution methods for image restoration problems. *Inverse Problems*, 17(6):1977–1995, 2001.

[114] J. Lamas-Rodriguez, F. Arguello, D.B. Heras, and M. Boo. Memory hierarchy optimization for large tridiagonal system solvers on GPU. In *IEEE 10th International Symposium on Parallel and Distributed Processing with Applications (ISPA)*, ASP-DAC '09, pages 87–94, Piscataway, NJ, USA, 2012. IEEE, IEEE Press.

[115] Z. Lin, B. Yu, and D. Zhu. A continuation method for solving fixed points of self-mappings in general nonconvex sets. *Nonlinear Analysis: Theory, Methods & Applications*, 52(3):905 – 915, 2003.

[116] Y. Lou, X. Zhang, S. Osher, and A. Bertozzi. Image recovery via nonlocal operators. *Journal of Scientific Computing*, 42(2):185–197, 2010.

[117] M. Lysaker, A. Lundervold, and X.-C. Tai. Noise removal using fourth-order partial differential equation with applications to medical magnetic resonance images in space and time. *IEEE Transactions on Image Processing*, 12(12):1579–1590, 2003.

[118] M. Lysaker, S. Osher, and X.-C. Tai. Noise removal using smoothed normals and surface fitting. *IEEE Transactions on Image Processing*, 13(10):1345–1357, 2004.

[119] F. Malgouyres. Minimizing the total variation under a general convex constraint for image restoration. *IEEE Transactions on Image Processing*, 11(12):1450 – 1456, 2002.

[120] S. Mallat. *A wavelet tour of signal processing*. Academic Press, 1997.

[121] A. Marquina and S. Osher. Explicit algorithms for a new time dependent model based on level set motion for nonlinear deblurring and noise removal. *SIAM Journal on Scientific Computing*, 22(2):387–405, 2000.

[122] V. P. Melnik, I. Shmulevich, K. Egiazarian, and J. Astola. Block-median pyramidal transform: analysis and denoising applications. *IEEE Transactions on Signal Processing*, 49(2):364–372, 2001.

[123] Y. Meyer. *Oscillating Patterns in Image Processing and Nonlinear Evolution Equations: The Fifteenth Dean Jacqueline B. Lewis Memorial Lectures*. American Mathematical Society, Boston, MA, USA, 2001.

[124] D. Mumford. Elastica and computer vision. In *Algebraic Geometry and Its Applications*, pages 491–506. Springer-Verlag, New York, 1994.

[125] M. Myllykoski. Separoituvien yhtälöryhmien ratkaiseminen ohjelmoitavalla näytönohjaimella (Solving separable linear systems on GPU). Master's thesis, University of Jyväskylä, 2011. In Finnish.

[126] M. Ng, P. Weiss, and X. Yuan. Solving constrained total-variation image restoration and reconstruction problems via alternating direction methods. *SIAM Journal on Scientific Computing*, 32(5):2710–2736, 2010.

[127] M. Nikolova. Minimizers of cost-functions involving nonsmooth data-fidelity terms. application to the processing of outliers. *SIAM Journal on Numerical Analysis*, 40(3):965–994, 2002.

[128] Nvidia Corporation. CUDA C Programming Guide, version 7.0. Available online from http://docs.nvidia.com/cuda/.

[129] E. Ollila. *Sign and rank covariance matrices with applications to multivariate analysis*. PhD thesis, University of Jyväskylä, 2002.

[130] S. Osher, M. Burger, D. Goldfarb, J. Xu, and W. Yin. An iterative regularization method for total variation-based image restoration. *Multiscale Modeling & Simulation*, 4(2):460–489, 2005.

[131] S. Osher, M. Burger, D. Goldfarb, J. Xu, and W. Yin. Using geometry and iterated refinement for inverse problems (1): Total variation based image restoration. Department of Mathematics, UCLA, LA, CA 90095, CAM Report, 4–13.

[132] S. Osher and R. Fedkiw. *Level Set Methods and Dynamic Implicit Surfaces*. Springer-Verlag, New York, 2002.

[133] S. Osher, A. Solé, and L. Vese. Image decomposition and restoration using total variation minimization and the H1. *Multiscale Modeling & Simulation*, 1(3):349–370, 2003.

[134] A. Parker and J. O. Hamblen. Optimal value for the Newton-Raphson division algorithm. *Information Processing Letters*, 42(3):141 – 144, 1992.

[135] P. Perona and J. Malik. Scale space and edge detection using anisotropic diffusion. In *Proceedings of IEEE Workshop on Computer Vision*, pages 16–27, 1987.

[136] P. Perona and J. Malik. Scale-space and edge detection using anisotropic diffusion. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 12(7):629–639, 1990.

[137] S. Petrova. Parallel implementation of fast elliptic solver. *Parallel Computing*, 23(8):1113–1128, 1997.

[138] J. Portilla, V. Strela, M. J. Wainwright, and E. P. Simoncelli. Image denoising using scale mixtures of gaussians in the wavelet domain. *IEEE Transactions on Image Processing*, 12(11):1338–1351, 2003.

[139] A. Quarteroni and A. Valli. *Numerical Approximation of Partial Differential Equations*, volume 23 of *Springer Series in Computational Mathematics*. Springer-Verlag Berlin Heidelberg, 1994.

[140] C. R. Rao. Methodology based on the l1-norm, in statistical inference. *Sankhya: The Indian Journal of Statistics*, 50:289–313, 1988.

[141] L. Reichel. The ordering of tridiagonal matrices in the cyclic reduction method for Poisson's equation. *Numerische Mathematik*, 56:215–227, 1989.

[142] W. Ring. Structural properties of solutions to total variation regularization problems. *Mathematical Modelling and Numerical Analysis*, 34(4):799–810, 2000.

[143] T. Rossi. *Fictitious domain methods with separable preconditioners*. PhD thesis, University of Jyväskylä, 1995.

[144] T. Rossi and J. Toivanen. A nonstandard cyclic reduction method, its variants and stability. *SIAM Journal on Matrix Analysis and Applications*, 20:628–645, 1999.

[145] T. Rossi and J. Toivanen. A parallel fast direct solver for block tridiagonal systems with separable matrices of arbitrary dimension. *SIAM Journal on Scientific Computing*, 20:1778–1796, 1999.

[146] L. I. Rudin, S. Osher, and E. Fatemi. Nonlinear total variation based noise removal algorithms. *Physica D*, 60(1-4):259–268, 1992.

[147] J. Scharcanski, C. R. Jung, and R. T. Clarke. Adaptive image denoising using scale and space consistency. *IEEE Transactions on Image Processing*, 11(9):1092–1101, 2002.

[148] H. Schwandt. Truncated interval arithmetic block cyclic reduction. *Applied Numerical Mathematics*, 5(6):495–527, 1989.

[149] L. Sendur and I.W. Selesnick. Bivariate shrinkage functions for wavelet-based denoising exploiting interscale dependency. *IEEE Transactions on Signal Processing*, 50(11):2744–2756, 2002.

[150] S. Sengupta, M. Harris, Y. Zhang, and J. D. Owens. Scan primitives for gpu computing. In *Proceedings of the 22Nd ACM SIGGRAPH/EUROGRAPHICS Symposium on Graphics Hardware*, GH '07, pages 97–106, Aire-la-Ville, Switzerland, 2007. Eurographics Association.

[151] J. Shi, Y. Cai, W. Hou, L. Ma, S.X. Tan, P.-H. Ho, and X. Wang. GPU friendly fast Poisson solver for structured power grid network analysis. In *Design Automation Conference, 2009. DAC '09. 46th ACM/IEEE*, pages 178–183, 2009.

[152] Y. Shi, L.-L. Wang, and X.-C. Tai. Geometry of total variation regularized L^p-model. *Journal of Computational and Applied Mathematics*, 236:2223–2234, 2012.

[153] E. Simoncelli. Bayesian denoising of visual images in the wavelet domain. In P. Müller and B. Vidakovic, editors, *Bayesian Inference in Wavelet-Based Models*, volume 141 of *Lecture Notes in Statistics*, pages 291–308. Springer New York, 1999.

[154] L. Sun and K. Chen. A new iterative algorithm for mean curvature-based variational image denoising. *BIT Numerical Mathematics*, 54(2):523–553, 2014.

[155] P. N. Swarztrauber. The methods of cyclic reduction, Fourier analysis and the FACR algorithm for the discrete solution of Poisson's equation on a rectangle. *SIAM Review*, 19:490–501, 1970.

[156] P. N. Swarztrauber. A direct method for the discrete solution of separable elliptic equations. *SIAM Journal on Numerical Analysis*, 11(6):1136–1150, 1974.

[157] P. N. Swarztrauber. Approximation cyclic reduction for solving Poisson's equation. *SIAM Journal on Scientific and Statistical Computing*, 8(3):199–209, 1987.

[158] P. N. Swarztrauber and R. A. Sweet. Vector and parallel methods for the direct solution of Poisson's equation. *Journal of Computational and Applied Mathematics*, 27(1–2):241–263, 1989. Special Issue on Parallel Algorithms for Numerical Linear Algebra.

[159] R. A. Sweet. A generalized cyclic reduction algorithm. *SIAM Journal on Numerical Analysis*, 11(3):506–520, 1974.

[160] R. A. Sweet. A cyclic reduction algorithm for solving block tridiagonal systems of arbitrary dimension. *SIAM Journal on Numerical Analysis*, 14:706–719, 1977.

[161] R. A. Sweet. A parallel and vector variant of the cyclic reduction algorithm. *SIAM Journal on Scientific and Statistical Computing*, 9:761–765, 1988.

[162] E. Tadmor, S. Nezzar, and L. Vese. A multiscale image representation using hierarchical (BV,L2) decompositions. *Multiscale Modeling & Simulation*, 2(4):554–579, 2004.

[163] X.-C. Tai, J. Hahn, and G. J. Chung. A fast algorithm for Euler's elastica model using augmented Lagrangian method. *SIAM Journal on Imaging Sciences*, 4(1):313–344, 2011.

[164] U. Trottenberg, C. Oosterlee, and A. Schuller. *Multigrid*. Academic Press, London, 2001.

[165] P. Vassilevski. Fast algorithm for solving a linear algebraic problem with separable variables. *Comptes rendus de l'Academie bulgare des Sciences*, 37:305–308, 1984.

[166] P Vassilevski. Fast algorithm for solving discrete Poisson equation in a rectangle. *Comptes rendus de l'Academie bulgare des Sciences*, 38:1311–1314, 1985.

[167] L. Vese. A study in the BV space of a denoising-deblurring variational problem. *Applied Mathematics and Optimization*, 44(2):131–161, 2001.

[168] C. R. Vogel and M. E. Oman. Iterative methods for total variation denoising. *SIAM Journal on Scientific Computing*, 17(1):227–238, 1996.

[169] C. R. Vogel and M. E. Oman. Fast, robust total variation-based reconstruction of noisy, blurred images. *IEEE Transactions on Image Processing*, 7(6):813–824, 1998.

[170] Y. Wang, J. Yang, W. Yin, and Y. Zhang. A new alternating minimization algorithm for total variation image reconstruction. *SIAM Journal on Imaging Sciences*, 1(3):248–272, 2008.

[171] L. T. Watson, S. C. Billups, and A. P. Morgan. Algorithm 652: Hompack: A suite of codes for globally convergent homotopy algorithms. *ACM Transactions on Mathematical Software*, 13(3):281–310, 1987.

[172] J. B. Weaver, Y. Xu, D. M. Healy, and L. D. Cromwell. Filtering noise from images with wavelet transforms. *Magnetic Resonance in Medicine*, 21(2):288–295, 1991.

[173] P. Wesseling. *An Introduction to Multigrid Methods*. Wiley, Chichester, 1992.

[174] S. Williams, A. Waterman, and D. Patterson. Roofline: An insightful visual performance model for multicore architectures. *Communications of the ACM*, 52(4):65–76, 2009.

[175] C. Wu and X.-C. Tai. Augmented Lagrangian method, dual methods, and split Bregman iteration for ROF, vectorial TV, and high order models. *SIAM Journal on Imaging Sciences*, 3(3):300–339, 2010.

[176] P. Yalamov and V. Pavlov. Stability of the block cyclic reduction. *Linear Algebra and its Applications*, 249(1–3):341–358, 1996.

[177] F. Yang, K. Chen, B. Yu, and D. Fang. A relaxed fixed point method for a mean curvature-based denoising model. *Optimization Methods and Software*, 29(2):274–285, 2014.

[178] I. Yanovsky, C. L. Guyader, A. Leow, A. Toga, P. Thompson, and L. Vese. Unbiased volumetric registration via nonlinear elastic regularization. In *Proceedings of the 2nd MICCAI Workshop on Mathematical Foundations of Computational Anatomy*, pages 1–12, 2008.

[179] Y.-L. You and M. Kaveh. Fourth-order partial differential equations for noise removal. *IEEE Transactions on Image Processing*, 9(10):1723–1730, 2000.

[180] Y. Zhang, J. Cohen, and J. D. Owens. Fast tridiagonal solvers on the GPU. In *Proceedings of the 15th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming PPoPP 10*, number 1, pages 127–136. ACM Press, 2010.

[181] W. Zhu and T. Chan. Image denoising using mean curvature of image surfaces. *SIAM Journal on Imaging Sciences*, 5(1):1–32, 2012.

[182] W. Zhu, X.-C. Tai, and T. Chan. Augmented Lagrangian method for a mean curvature based image denoising model. *Inverse Problems in Image Processing*, 7(4):1409–1432, 2013.

ORIGINAL PAPERS

PI

FAST POISSON SOLVERS FOR GRAPHICS PROCESSING UNITS

by

Mirko Myllykoski, Tuomo Rossi, and Jari Toivanen 2013

In P. Manninen and P. Öster, editors, Applied Parallel and Scientific Computing, volume 7782 of Lecture Notes in Computer Science, pages 265–279

PII

A PARALLEL RADIX-4 BLOCK CYCLIC REDUCTION ALGORITHM

by

Mirko Myllykoski and Tuomo Rossi 2014

Numerical Linear Algebra with Applications, 21(4):540–556

PIII

ON SOLVING SEPARABLE BLOCK TRIDIAGONAL LINEAR SYSTEMS USING A GPU IMPLEMENTATION OF RADIX-4 PSCR METHOD

by

Mirko Myllykoski, Tuomo Rossi, and Jari Toivanen

Submitted to SIAM Journal on Scientific Computing

ON SOLVING SEPARABLE BLOCK TRIDIAGONAL LINEAR SYSTEMS USING A GPU IMPLEMENTATION OF RADIX-4 PSCR METHOD

M. MYLLYKOSKI[†][§][¶], T. ROSSI[†], AND J. TOIVANEN[†][‡]

Abstract. Partial solution variant of the cyclic reduction (PSCR) method is a direct solver which can be applied to certain separable block tridiagonal linear systems. Such linear systems arise, e.g., from the Poisson and Helmholtz equations discretized with (mass lumped) linear finite-elements or bilinear (or trilinear) finite-elements. Furthermore, the domain has to be rectangular and the mesh orthogonal. This paper presents a generalized graphics processing unit (GPU) implementation of the radix-4 PSCR method and numerical results obtained while testing the implementation using the two aforementioned numerical problems. The numerical results indicate up to 16-fold speedups when compared to an equivalent CPU implementation utilizing a single CPU core. Attained floating point performance is analyzed using the roofline performance analysis model. Both the off-chip and on-chip memory accesses are covered by the analysis. Obtained roofline models show that the attained floating point performance is mainly limited by the available off-chip memory bandwidth and the effectiveness of the tridiagonal solver used to solve the arising tridiagonal subproblems. The performance was accelerated using a Newton-Raphson division algorithm and rudimentary parameter optimization.

Key words. fast direct solver, GPU computing, partial solution technique, PSCR method, roofline model, separable block tridiagonal linear system

AMS subject classifications. 35J05, 65F05, 65F50, 65N30, 65Y05

1. Introduction.

Separable block tridiagonal linear systems appear in many practical applications. In such a system the coefficient matrix can be presented in a separable form using the Kronecker matrix tensor product. An example is a Poisson equation with Dirichlet boundary conditions discretized in a rectangular domain using an orthogonal finite-element mesh and bilinear (or trilinear) finite-elements. A similarly treated Helmholtz equation either with absorbing boundary conditions (ABC) [3, 15, 17] or a perfectly matched layer (PML) [4, 5], among others, will also lead to a suitable linear system [19]. Such systems also appear as subproblems in a variety of situations. For example, a similarly discretized diffusion equation with a suitable implicit time-stepping scheme will lead to the solution of a separable block tridiagonal linear system on each time step.

Numerous effective direct methods have been derived by employing many useful properties of the Kronecker matrix tensor forms. A comprehensive survey of these so-called matrix decomposition algorithms (MDAs) can be found in [6]. A MDA operates by reducing the linear system into a set of smaller sub-systems which can be solved independently of each other. The solution of the original linear system is then obtained by reverting the reduction operation. MDAs are similar to the well-known method of separation of variables. Many MDAs utilize the fast Fourier transformation (FFT) method while performing the reduction operation. One of the first and most important of these is so-called Fourier analysis method introduced by Hockney [21] in 1965.

[†]Department of Mathematical Information Technology, University of Jyväskylä, P.O. Box 35 (Agora), FI-40014 University of Jyväskylä, Finland.

[‡]Department of Aeronautics & Astronautics, Stanford University, Stanford, CA 94305, USA

[§]The research of the first author was supported by the Academy of Finland, grant #252549.

[¶]Email: mirko.myllykoski@jyu.fi

Cyclic reduction methods are a well-known class of numerical algorithms which can be applied, among other things, to tridiagonal and block tridiagonal linear systems. A traditional cyclic reduction type method (see, e.g., [10, 20, 21, 34, 40, 41]) operates in two stages. The first stage generates a sequence of sub-systems by recursively eliminating odd numbered (block) rows from the system. As a result, the size of each sub-system is approximately half of the size of the previous sub-system. This means that the reduction factor, or the radix number, is two. The reduction operation often takes advantage of properties of the coefficient matrix. For example, in certain cases, the sub-systems can be represented using matrix rational polynomials which considerably simplifies computations and reduces amount of memory needed perform the computations. The sub-systems are solved in reverse order during the back substitution stage of the algorithm. A recent survey of the cyclic reduction methods and their applications can be found in [7].

This paper deals with a direct method called partial solution variant of the cyclic reduction (PSCR). It is a cyclic reduction type method with MDA-type features for separable block tridiagonal linear systems. The initial work on the (radix-2) PSCR method was done in the 80's by Vassilevski [42, 43] and Kuznetsov [26]. The method uses so-called partial solution technique [2, 27], which can be applied effectively to a separable linear system when a sparse set of the solution components is required and the right-hand side vector has only a few non-zero elements. The technique is a special form of MDA. A more generalized radix-q algorithm was formulated later in [28] and a parallel radix-4 CPU implementation was presented in [35]. Parallel implementations were also considered earlier in [1] and [33]. Here the radix number q means that the system size is reduced by a factor of q on each reduction step. The usual formulation of the PSCR method only distantly resembles the formulation of a traditional block cyclic reduction type method. However, in certain special cases, equivalent methods can be derived using a more traditional matrix rational polynomial approach [30, 34]. If the factor matrices are tridiagonal, then the arithmetical complexity of the method is $\mathcal{O}(n_1 n_2 \log n_1)$ for $n_1 n_2 \times n_1 n_2$ two-dimensional problems and $\mathcal{O}(n_1 n_2 n_3 \log n_1 \log n_2)$ for $n_1 n_2 n_3 \times n_1 n_2 n_3$ three-dimensional problems.

While scalar cyclic reduction (CR) method [21] and its parallel variant (parallel cyclic reduction, PCR) [22] have become very popular methods for the solution of tridiagonal linear systems on graphics processing units (GPUs) (see, e.g., [14, 18, 23, 24, 29, 37, 45]), the block cyclic reduction type methods have been somewhat less popular topic. Comparisons between CPU and GPU implementations of simplified radix-2 [34] and radix-4 [30] methods can be found in [31]. The paper concluded that the block cyclic reduction type methods considered in the paper are suitable for GPU computation and that the radix-4 method is better able to utilize the GPU's parallel computing resources than the radix-2 method. Obtained numerical results indicated up to 6-fold speed increase for two-dimensional Poisson equations and up to 3-fold speed increase for three-dimensional Poisson equations. The implementations were compared against equivalent multi-threaded CPU implementations run on a quad-core CPU.

This paper presents a generalized GPU implementation of the radix-4 PSCR method. In a sense, this paper can be seen as a natural follow-up to [30]. The implementation presented in this paper can be applied to real and complex valued problems. The problem can be two- or three-dimensional and the factor matrices are assumed to be tridiagonal and symmetric. As an additional, formal requirement, certain factor matrices are assumed to be positive definite. However, the GPU implementation presented in this paper and the CPU implementation presented in [35] (see [19]) are applicable in some cases even when this condition is not strictly satisfied. The implementation is tested using a Poisson equation with Dirichlet boundary conditions and a Helmholtz equation with second-order ABC. In the both cases the domain is rectangular and the discretization is performed using an orthogonal mesh. In the case of the Poisson equation, the finite-element space consists of (mass lumped) piecewise linear element, and in the case of the Helmholtz equation, the finite-element space consist of bilinear (or trilinear) elements. The obtained numerical results are analyzed using roofline performance analysis model [44]. The model takes into account the available off-chip memory bandwidth and, thus, provides a very extensive picture of how effectively the computational and memory resources are utilized.

The rest of this paper is organized as follows: Section 2 describes the Kronecker matrix tensor product, the partial solution technique, and the PSCR method. Section 3 gives a brief introduction to GPU computing and describes the GPU implementation. Section 4 presents the numerical results. The roofline models are presented in Section 5. The final conclusions are given in Section 6.

2. Radix-q PSCR method. In order to describe the PSCR method, we first define the Kronecker matrix tensor product for matrices $B \in \mathbb{K}^{n_1 \times m_1}$ and $C \in \mathbb{K}^{n_2 \times m_2}$, as follows

$$B \otimes C = \begin{bmatrix} b_{1,1}C & b_{1,2}C & \dots & b_{1,m_1}C \\ b_{2,1}C & b_{2,2}C & \dots & b_{2,m_1}C \\ \vdots & \vdots & \ddots & \vdots \\ b_{n_1,1}C & b_{n_1,2}C & \dots & b_{n_1,m_1}C \end{bmatrix} \in \mathbb{K}^{n_1 n_2 \times m_1 m_2}, \tag{2.1}$$

where the field $\mathbb{K}$ can be $\mathbb{R}$ or $\mathbb{C}$. The product has two properties which are the basis of many matrix decomposition algorithms: First, let $D \in \mathbb{K}^{m_1 \times n_1}$ and $E \in \mathbb{K}^{m_2 \times n_2}$. Then,

$$(B \otimes C)(D \otimes E) = (BD \otimes CE) \in \mathbb{K}^{n_1 n_2 \times n_1 n_2}. \tag{2.2}$$

Second, let $D \in \mathbb{K}^{n_1 \times n_1}$ and $E \in \mathbb{K}^{n_2 \times n_2}$. If the matrices D and E are nonsingular, then product matrix $D \otimes C \in \mathbb{K}^{n_1 n_2 \times n_1 n_2}$ is also nonsingular and

$$(D \otimes E)^{-1} = D^{-1} \otimes E^{-1} \in \mathbb{K}^{n_1 n_2 \times n_1 n_2}. \tag{2.3}$$

These two results can be derived from the definition of the Kronecker matrix tensor product.

Generally speaking, the PSCR method considered in this paper can be applied to a linear system $Au = f$, with

$$A = A_1 \otimes M_2 + M_1 \otimes A_2 + c(M_1 \otimes M_2), \tag{2.4}$$

where the factor matrices $A_1 \in \mathbb{K}^{n_1 \times n_1}$ and $M_1 \in \mathbb{K}^{n_1 \times n_1}$ are symmetric and tridiagonal, $A_2, M_2 \in \mathbb{K}^{n_2 \times n_2}$, and $c \in \mathbb{K}$. Thus, the coefficient matrix A is symmetric and block tridiagonal. This corresponds to a two-dimensional problem. The method can also be applied to three-dimensional problems in which case the coefficient matrix A is of the form

$$A_1 \otimes M_2 \otimes M_3 + M_1 \otimes A_2 \otimes M_3 + M_1 \otimes M_2 \otimes A_3 + c(M_1 \otimes M_2 \otimes M_3), \tag{2.5}$$

where the factor matrices A_2 and M_2 are symmetric and tridiagonal, and $A_3, M_3 \in \mathbb{K}^{n_3 \times n_3}$. The PSCR method reduces a three-dimensional problem into a set of two-dimensional problems where the coefficient matrices are of the form

$$A_2 \otimes M_3 + M_2 \otimes A_3 + (c + \lambda)(M_2 \otimes M_3), \tag{2.6}$$

with $\lambda \in \mathbb{K}$. The PSCR can be applied recursively to solve problems with (2.6).

2.1. Basic algorithm. This and the next subsection describe the radix-q PSCR method using projection matrices similarly to [35]. Let sets $J_0, J_1, \dots, J_k \subset \mathbb{N}$ determine (indirectly) which block rows are eliminated during each reduction step. The exact formulation of the PSCR method depends on how these sets are chosen. In the case of the radix-q PSCR method, $k = \lfloor \log_q n_1 \rfloor + 1$ and the sets fulfill the following conditions:

(i) $J_0 = \{1, 2, 3, \dots, n_1\}$, $J_k = \emptyset$.
(ii) $J_k \subset J_{k-1} \subset \dots \subset J_1 \subset J_0$.
(iii) Let $\hat{J}_i = J_i \cup \{0, n_1 + 1\}$, $(\hat{j}_1^{(i)}, \hat{j}_2^{(i)}, \dots)$ be the elements of the set $\hat{J}_i$, in ascending order, and

$$D_l^{(i)} = \{j \in J_{i-1} : \hat{j}_l^{(i)} < j < \hat{j}_{l+1}^{(i)}\}.$$

Then $\#D_l^{(i)} \leq q - 1$ for all $i = 1, 2, \dots, k$ and $l = 1, 2, \dots, \#J_i + 1$.

Above $\#J_i$ and $\#D_l^{(i)}$ are the cardinalities of the sets J_i and $D_l^{(i)}$, respectively. The consequence of the third condition is that the rows, that are to be eliminated during a reduction step, are distributed into groups of a size of no more than $q - 1$. Examples of the sets $J_1, J_2, \dots, J_k$ are given in Figure 2.1 and in Section 2.3.

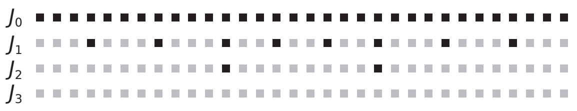


FIG. 2.1. *An example of the sets J_0, J_1, J_2, J_3 when $q = 4$ and $n_1 = 32$.*

With the sets $J_0, J_1, \dots, J_k$, we can define projection matrices

$$\tilde{P}^{(i)} = \operatorname{diag}\{p_1^{(i)}, p_2^{(i)}, \dots, p_{n_1}^{(i)}\} \in \mathbb{K}^{n_1}, \ i = 1, 2, \dots, k, \tag{2.7}$$

with

$$p_j^{(i)} = \begin{cases} 1, & j \notin J_i, \\ 0, & j \in J_i. \end{cases} \tag{2.8}$$

Based on these, we can define a second set of projection matrices: $P^{(i)} = \tilde{P}^{(i)} \otimes I_{n_2}$, $i = 1, 2, \dots, k$.

Under the assumption that projected matrix $P^{(i)} A P^{(i)}$ is nonsingular in subspace $\operatorname{Im}(P^{(i)})$ for all $i = 1, 2, \dots, k$, the system $Au = f$ can be solved in two stages:

(i) Let $f^{(1)} = f$. Then, for $i = 1, 2, \dots, k - 1$: Solve the vector $v^{(i)}$ from

$$P^{(i)} A P^{(i)} v^{(i)} = P^{(i)} f^{(i)}$$

and compute

$$f^{(i+1)} = f^{(i)} - A P^{(i)} v^{(i)}.$$

(ii) Let $u^{(k+1)} = 0$. Then, for $i = k, k-1, \ldots, 1$: Solve the vector $u^{(i)}$ from

$$P^{(i)} A P^{(i)} u^{(i)} = P^{(i)} f^{(i)} - P^{(i)} A (I - P^{(i)}) u^{(i+1)}$$

and compute

$$(I - P^{(i)}) u^{(i)} = (I - P^{(i)}) u^{(i+1)}.$$

Finally, $u = u^{(1)}$.

The rationale behind the aforementioned recursive technique becomes clear after the following observations: Due to the special block tridiagonal structure of the coefficient matrix A, only a sparse set of the solution components are actually required in the update formulas, and the right-hand side vectors have only a few non-zero elements. In this situation the partial solution technique [2, 27], which is described in the next subsection, can be applied very effectively.

2.2. Partial solution technique. Let us focus on a separable linear system $\tilde{A}v = g$ with

$$\tilde{A} = \tilde{A}_1 \otimes M_2 + \tilde{A}_1 \otimes A_2 + c(\tilde{M}_1 \otimes M_2), \tag{2.9}$$

where $\tilde{A}_1, \tilde{M}_1 \in \mathbb{K}^{m \times m}$ are tridiagonal and symmetric. Let us have projection matrices $\tilde{R} \in \mathbb{K}^{m \times m}$ and $\tilde{Q} \in \mathbb{K}^{m \times m}$ defining the required solution components and the non-zero components of the right-hand side vector, respectively. Based on these projection matrices, we construct two additional projection matrices: $R = \tilde{R} \otimes I_{n_2}$ and $Q = \tilde{Q} \otimes I_{n_2}$.

Instead of solving the whole vector v, we are going to solve only vector Rv. In addition, it is assumed that $g \in \mathrm{Im}(Q)$. The vector Rv can be solved effectively by using the following formula:

$$R\tilde{A}^{-1}Q = (\tilde{R}W \otimes I_{n_2})((\Lambda + cI_m) \otimes M_2 + I_m \otimes A_2)^{-1}(W^T\tilde{Q} \otimes I_{n_2}), \tag{2.10}$$

where the diagonal matrix $\Lambda \in \mathbb{K}^{m \times m}$ and the matrix $W \in \mathbb{K}^{m \times m}$ have the properties

$$W^T \tilde{A}_1 W = \Lambda \ \text{ and } \ W^T \tilde{M}_1 W = I_m. \tag{2.11}$$

In other words,

$$W = [w_1 w_2 \ldots w_m] \ \text{ and } \ \Lambda = \mathrm{diag}\{\lambda_1, \lambda_2, \ldots, \lambda_m\} \tag{2.12}$$

contain the eigenvalues $\lambda_1, \lambda_2, \ldots, \lambda_m \in \mathbb{K}$ and the $\tilde{M}_1$-orthonormal eigenvectors $w_1, w_2, \ldots, w_m \in \mathbb{K}^m$, which satisfy the generalized eigenvalue problem

$$\tilde{A}_1 w_j = \lambda_j \tilde{M}_1 w_j. \tag{2.13}$$

The aforementioned formula (2.10) follows from the properties (2.2) and (2.3).

In the case of the radix-q PSCR method, the projection matrices $\tilde{R}$ and $\tilde{Q}$ contain not more than $q+1$ non-zero elements, which means that only certain components of the eigenvectors are required to compute (2.10). If the factor matrices A_2 and M_2 are tridiagonal, then the arithmetical complexity of the partial solution technique is $\mathcal{O}(mn_2)$, the most expensive operation being the solution of m tridiagonal linear systems.

2.3. Explicit formulas. This subsection gives explicit formulas in the special case when $n_1 = q^k - 1$ for some positive integer k. The method can be generalized easily for arbitrarily n_1 but due to lengthy formulas we limit ourselves to presenting the method in this special case. However, the implementation presented in this paper is applicable for general n_1. We begin by defining the sets $J_0, J_1, \dots, J_k \subset \mathbb{N}$ as

$$\begin{aligned} &J_0 = \{1, 2, 3, \dots, n_1\}, \\ &J_i = \left\{ j \cdot q^i : j = 1, 2, \dots, q^{k-i} - 1 \right\}, i = 1, 2, \dots, k-1 \text{ and} \\ &J_k = \emptyset. \end{aligned} \tag{2.14}$$

Clearly these sets fulfill the conditions enumerated in Section 2.1.

In summary, the radix-q PSCR method proceeds for a two-dimensional problem as follows: First, we solve a set of generalized eigenvalue problems

$$\tilde{A}_j^{(i)} w_{j,l}^{(i)} = \lambda_{j,l}^{(i)} \tilde{M}_j^{(i)} w_{j,l}^{(i)}, \;\; i = 1, 2, \dots, k, \; l = 1, 2, \dots, m_i, \tag{2.15}$$

where $\tilde{A}_1^{(i)}, \tilde{A}_2^{(i)}, \dots, \tilde{A}_{q^{k-i}}^{(i)} \in \mathbb{K}^{m_i \times m_i}$ and $\tilde{M}_1^{(i)}, \tilde{M}_2^{(i)}, \dots, \tilde{M}_{q^{k-i}}^{(i)} \in \mathbb{K}^{m_i \times m_i}$ are the non-zero diagonal blocks from projected matrices $\tilde{P}^{(i)} A_1 \tilde{P}^{(i)}$ and $\tilde{P}^{(i)} M_1 \tilde{P}^{(i)}$, respectively, in order starting from the upper left corner, and $m_i = q^i - 1$. This step can be considered as an initialization stage.

Next, let $f^{(1)} = f$. We recursively compute vectors $f^{(2)} \in \mathbb{K}^{(q^{k-1}-1)n_2}, f^{(3)} \in \mathbb{K}^{(q^{k-2}-1)n_2}, \dots, f^{(k)} \in \mathbb{K}^{(q-1)n_2}$ by using the formula

$$f_j^{(i+1)} = f_{qj}^{(i)} - \hat{T}_j^{(i)} \sum_{l=1}^{m_i} (w_{j,l}^{(i)})_{m_i} v_{j,l}^{(i)} - \check{T}_j^{(i)} \sum_{l=1}^{m_i} (w_{j+1,l}^{(i)})_1 v_{j+1,l}^{(i)} \in \mathbb{K}^{n_2}, \tag{2.16}$$

where

$$\begin{aligned} \hat{T}_j^{(i)} &= ((A_1)_{jq^i, jq^i-1} + c\,(M_1)_{jq^i, jq^i-1}) M_2 + (M_1)_{jq^i, jq^i-1} A_2, \\ \check{T}_j^{(i)} &= ((A_1)_{jq^i, jq^i+1} + c\,(M_1)_{jq^i, jq^i+1}) M_2 + (M_1)_{jq^i, jq^i+1} A_2. \end{aligned}$$

The vectors $v_{j,l}^{(i)} \in \mathbb{K}^{n_2}$ are solved from

$$\left(A_2 + (\lambda_{j,l}^{(i)} + c) M_2 \right) v_{j,l}^{(i)} = \sum_{s=1}^{q-1} (w_{j,l}^{(i)})_{sq^{i-1}} f_{(j-1)q+s}^{(i)}. \tag{2.17}$$

Next, we recursively compute vectors $u^{(k)} \in \mathbb{K}^{(q-1)n_2}, u^{(k-1)} \in \mathbb{K}^{(q^2-1)n_2}, \dots, u^{(1)} \in \mathbb{K}^{(q^k-1)n_2}$ by using the formula

$$\begin{aligned} u_{qd+j}^{(i)} &= \sum_{l=1}^{m_i} (w_{d+1,l}^{(i)})_{jq^{i-1}} y_{d,l}^{(i)} \in \mathbb{K}^{n_2}, \; j = 1, 2, \dots, q-1, \\ u_{qd+q}^{(i)} &= u_{d+1}^{(i+1)} \in \mathbb{K}^{n_2}, \; d < q^{k-i} - 1, \end{aligned} \tag{2.18}$$

where $d = 0, 1, \dots, q^{k-i} - 1$. The vectors $y_{d,l}^{(i)} \in \mathbb{K}^{n_2}$ are solved from

$$\begin{aligned} \left(A_2 + (\lambda_{d+1,l}^{(i)} + c) M_2 \right) y_{d,l}^{(i)} = \sum_{s=1}^{q-1} (w_{d+1,l}^{(i)})_{sq^{i-1}} f_{qd+s}^{(i)} - \\ \hat{K}_d^{(i)} (w_{d+1,l}^{(i)})_1 u_d^{(i+1)} - \check{K}_{d+1}^{(i)} (w_{d+1,l}^{(i)})_{m_i} u_{d+1}^{(i+1)}, \end{aligned} \tag{2.19}$$

where

$$\hat{K}_d^{(i)} = ((A_1)_{dq^i+1,dq^i} + c\,(M_1)_{dq^i+1,dq^i})M_2 + (M_1)_{dq^i+1,dq^i}A_2,$$
$$\check{K}_d^{(i)} = ((A_1)_{dq^i-1,dq^i} + c\,(M_1)_{dq^i-1,dq^i})M_2 + (M_1)_{dq^i-1,dq^i}A_2,$$

$\hat{K}_0^{(i)} = \hat{K}_{q^{k-i}}^{(i)} = 0$, and $u_0^{(i+1)} = u_{q^{k-i}}^{(i+1)} = 0$. Finally, $u = u^{(1)}$.

In the case of a three-dimensional problem the formulas are essentially the same. The numerical stability of the PSCR method depend largely on the properties of the generalized eigenvalue problems (2.15) associated with the partial solutions.

3. Implementation.

3.1. OpenCL.

Open computing language (OpenCL) is a programming framework which enables programmers to develop portable programs for heterogeneous platforms. One of the main applications of the framework is in utilizing GPUs for general purpose computation. Other similar frameworks also do exist, perhaps the most popular being called compute unified device architecture (CUDA). CUDA can be used only with Nvidia's GPUs while OpenCL is intended to be platform independent. The GPU-related part of OpenCL and CUDA resemble each other in many respects, but there are some major differences between the terminologies used. Since an OpenCL implementation is presented in this paper, the OpenCL terminology is used throughout this paper. OpenCL specifications do not include much relevant information on GPU hardware. That knowledge is, however, essential for high performance implementation on a GPU. For that reason, some additional information on Nvidia's current hardware is provided.

A contemporary high-end GPU contains thousands of processing elements (cores) which are grouped into multiple computing units. On the logical level all processing elements are full-fledged cores, but on the hardware level the processing elements are linked to each other in many ways. The processing elements inside the same computing unit share certain resources such as a register file, schedulers, special function units, and caches. Because multiple processing elements share the same scheduler, the code is actually executed in a synchronous manner. The hardware handles branches by going through all necessary execution paths and disabling those processing elements that do not contribute to the final result.

GPU has two primary memory spaces:

Global memory is a large but relative slow (off-chip) memory space, which can be accessed by all processing elements. Achieved memory bandwidth depends largely on the access pattern. Usually this means that the processing elements, that are executing the same memory access command synchronously, should access memory locations that are located inside the same memory block. These kind of coherent memory requests can be carried out collectively while scattered memory requests lead to several separate memory transactions.

Local memory is a fast (on-chip) memory space which is commonly used when multiple processing elements, that belong to the same computing unit, want to share data. This memory is often divided into 32-bit or 64-bit memory banks organized such a way that successive 32-bit (or 64-bit) words map to successive memory banks. Each memory bank is capable of serving only one memory request simultaneously. This limits the number of effective access patterns as simultaneous memory requests, that point to the same memory bank, cause a memory bank conflict and are processed sequentially.

In addition, there are two additional memory spaces called constant memory and private memory, but they are not used (explicitly) in our GPU implementation.

A processing element is capable of executing multiple work-items (threads) concurrently. Having extra active work-items is actually beneficial as the computing pipeline can be kept occupied and the memory latencies can be seemingly hidden. Nvidia uses the term warp to describe a set of work-items that are executed together in a synchronized manner. The warp size is 32 work-items. The work-items are grouped into multiple work groups, which are then mapped to the computing units. GPU-side code execution begins when a special kind of subroutine called kernel is launched. Every work-item starts from the same location in the code, but each work-item is given a unique index number which makes branching possible. The work groups are also indexed. The work-items, that belong to the same work group, can share a portion of the local memory and can use synchronization commands which guarantee memory coherence between the work-items.

3.2. General notes. The GPU implementation is based on the radix-4 PSCR method. The radix-4 variant was chosen because it is relatively simple to implement but it is still nearly optimal in the sense of arithmetical complexity [35]. Based on the numerical results presented in [31], it is also likely that a radix-4 method will outperform a radix-2 method on a GPU, especially when the problem size is small.

The GPU implementation can be applied to problems where the factor matrices A_1, A_2, M_1, and M_2 are tridiagonal and symmetric. For three-dimensional problem, the factor matrices A_3 and M_3 are also assumed to be tridiagonal and symmetric. The field $\mathbb{K}$ can be either $\mathbb{R}$ or $\mathbb{C}$. The system size can be arbitrary as long as the GPU has enough global memory to accommodate the right-hand side vector, the factor matrices, the eigenvalues, the required eigenvector components, guiding information, and workspace buffers.

The generalized eigenvalue problems are solved using the CPU. This is not a major limitation because the PSCR method is usually used when one must solve a large set of problems with (nearly) identical coefficient matrices. If the factor matrix M_1 (or M_2) is not diagonal, then the generalized eigenvalue problem is preprocessed with Crawford's algorithm [13] leading to an ordinary eigenvalue problem with the same eigenvalues. The eigenvalues are then solved using the LR-algorithm [36] coupled with Wilkinson's shift. The eigenvectors are solved using the inverse iteration method after the eigenvalues have been computed. Crawford's algorithm uses the Cholesky decomposition which assumes that the factor matrix M_1 (and M_2) is positive definite. However, this condition does not need to be always satisfied.

OpenCL does not have a native support for complex numbers. In the GPU implementation presented in this paper, a complex number is presented using a vector of length two. Multiplications and divisions are implemented using suitable precompiler conditionals/macros. This means that the same codebase can be used for real and complex valued problems. Problems appear when each work-item accesses two consecutive elements of a complex valued vector that is stored in the global memory. If the real and imaginary parts are stored together in the same memory buffer, then each work-item would have to load a 256-bit memory block at once. However, for performance reasons each work-item should access a maximum of 128-bit memory block at once. This problem is avoided by storing the real and imaginary parts separately in the situation where it is beneficial. The same approach is also used with the local memory to avoid memory bank conflicts.

Double precision floating point division is a very expensive operation on a con-

	FMA	division	Newton-Raphson
GTX580	$98.7 \cdot 10^9$ op/s (197 GFlop/s)	$9.55 \cdot 10^9$ op/s	$9.89 \cdot 10^9$ op/s
K40c	$704 \cdot 10^9$ op/s (1408 GFlop/s)	$25.1 \cdot 10^9$ op/s	$55.1 \cdot 10^9$ op/s

Table 3.1
Measured arithmetical operation output on Nvidia Geforce GTX580 and Nvidia Tesla K40c GPUs.

temporary GPU. Based on the measurements shown in Table 3.1, the native double precision floating point division is over 28 times more expensive than double precision fused multibly-add (FMA) operation on a Nvidia Tesla K40c GPU. This motivated us to consider Newton-Raphson division algorithm [16] for computing the inverse of a real number $D \in [1/2, 1)$. The iteration formula is

$$X_{i+1} = X_i + X_i(1 - DX_i) \tag{3.1}$$

and the optimal linear initial approximation of $1/D$ which minimize the maximum relative error in the final result over interval $[1/2, 1]$ is given by

$$X_0 = \frac{48}{17} - \frac{32}{17}D. \tag{3.2}$$

This initial approximation leads to full double precision accuracy with only four iteration [32], which means that the cost of computing the reciprocal is 9 FMA. The scaling into the half-open interval $[1/2, 1)$ and back to the correct range can be performed with five bitwise operations and two integer additions/subtractions. As shown in Table 3.1, on a Nvidia Geforce GTX580 GPU, the Newton-Raphson division slightly outperforms the native floating point division, and on the K40c GPU, the Newton-Raphson division is more than two times faster.

The implementation is divided into three levels: The level 3 is the tridiagonal solver, which is responsible for solving the tridiagonal sub-problems in (2.17) and (2.19). The details are given in the next subsection. The level 2 forms the right-hand side vectors for the aforementioned tridiagonal sub-problems and computes the vectors (2.16) and (2.18). The level 1 is analogous to the level 2 as it performs the same operations to a three-dimensional problem. The only major difference is that the sub-problems are then block tridiagonal.

When the tridiagonal solver is excluded, the implementation consists the following seven kernels:
lx_stage_11 forms the right-hand side vectors for the sub-problems in (2.17).
lx_stage_12a computes the vector sums in (2.16).
lx_stage_y2b helps to compute the vector sums in (2.16) and (2.18).
lx_stage_12c computes the vector $f^{(i+1)}$ in (2.16).
lx_stage_21 forms the right-hand side vectors for the sub-problems in (2.19).
lx_stage_22a computes the vector sums in (2.18).
lx_stage_22c computes the vector $u^{(i)}$ in (2.18).
As the levels 1 and 2 are analogous to each others, they use the same codebase. Level 1 kernels have a prefix l1 and level 2 kernels have a prefix l2.

The purpose of the kernel lx_stage_y2b is to distribute the task of computing the vector sums in (2.16) and (2.18). Each vector sum is divided into multiple partial sums which are then evaluated in parallel by the kernels lx_stage_12a and lx_stage_22a. The same division into partial sums is then repeated recursively by the kernel lx_stage_y2b using a tree-like reduction pattern until the whole vector sum is evaluated.

Many aspects of the implementation are parametrized. For example, each kernel has its own parameter that specifies the size of the work group. The maximum size of each partial vector sum and division algorithm are also set by parameters. In addition, several parameters control the way the tridiagonal solver behaves. Optimal parameters are selected by solving the arising integer programming problems using a (discrete) differential evolution [38, 39] method.

3.3. Tridiagonal solver (level 3). The tridiagonal solver used in this paper is a generalized and extended version of the CR based tridiagonal solver used in [31]. The extended tridiagonal solver is based on the CR, PCR, and Thomas [11] methods. The CR method has been shown to be particularly effective on GPUs due to its simplicity and parallel nature (see, e.g., [14, 18, 23, 24, 29, 37, 45]). The method solves a tridiagonal linear system $Du = f$, $D = \operatorname{tridiag}\{a_i, b_i, c_i\} \in \mathbb{K}^{m \times m}$, by generating a sequence of tridiagonal sub-systems as follows: Let $a_i^{(0)} = a_i$, $b_i^{(0)} = b_i$, $c_i^{(0)} = c_i$, and $f_i^{(0)} = f_i$, $i = 1, 2, \ldots, m$. Now, for $i = 1, 2, \ldots, k-1$, $k = \lfloor \log_2(m) \rfloor + 1$, we generate a sequence of sub-systems by using the formulas

$$\begin{aligned}
\alpha_j^{(i)} &= -a_{2j}^{(i-1)}/b_{2j-1}^{(i-1)},\ \beta_j^{(i)} = -c_{2j}^{(i-1)}/b_{2j+1}^{(i-1)}, \\
a_j^{(i)} &= \alpha_j^{(i)} a_{2j-1}^{(i-1)},\ c_j^{(i)} = \beta_j^{(i)} c_{2j+1}^{(i-1)}, \\
b_j^{(i)} &= b_{2j}^{(i-1)} + \alpha_j^{(i)} c_{2j-1}^{(i-1)} + \beta_j^{(i)} a_{2j+1}^{(i-1)}, \\
f_j^{(i)} &= f_{2j}^{(i-1)} + \alpha_j^{(i)} f_{2j-1}^{(i-1)} + \beta_j^{(i)} f_{2j+1}^{(i-1)},
\end{aligned} \tag{3.3}$$

where $j = 1, 2, \ldots, \lfloor m/2^i \rfloor$, $a_{\lfloor m/2^{i-1} \rfloor+1}^{(i-1)} = b_{\lfloor m/2^{i-1} \rfloor+1}^{(i-1)} = c_{\lfloor m/2^{i-1} \rfloor+1}^{(i-1)} = f_{\lfloor m/2^{i-1} \rfloor+1}^{(i-1)} = 0$, and $c_{\lfloor m/2^{i-1} \rfloor}^{(i-1)} = 0$. Then, for $i = k-1, k-2, \ldots, 0$, we solve the sub-system using the formula

$$u_j^{(i)} = \begin{cases} (f_j^{(i)} - a_j^{(i)} u_{(j-1)/2}^{(i+1)} - c_j^{(i)} u_{(j-1)/2+1}^{(i+1)})/b_j^{(i)}, & j \notin 2\mathbb{N}, \\ u_{j/2}^{(i+1)}, & j \in 2\mathbb{N}, \end{cases} \tag{3.4}$$

where $j = 1, 2, \ldots, \lfloor m/2^i \rfloor$, $a_1^{(i)} = c_{\lfloor m/2^i \rfloor}^{(i)} = 0$, and $u_0^{(i+1)} = u_{\lfloor m/2^{i+1} \rfloor+1}^{(i+1)} = 0$. Finally, $u = u^{(0)}$. The arithmetical complexity of the method is $\mathcal{O}(m)$.

In its basic formulation the CR method suffers from many well-known drawbacks. The most significant of them, when GPUs are concerned, are: (i) The memory access pattern disperses exponentially as the method progresses. This causes problems with the global memory as the processing elements, that are executing the same memory access command synchronously, must access data that does not any longer fit inside a single memory block. And, it causes problems with the local memory because some processing elements, that are executing the same memory access command synchronously, must access data that is stored in the same memory bank. This problem is tackled in this paper by different permutation patterns. (ii) The number of parallel tasks is reduced by a factor of two on each CR recursion step. Thus, some processing elements are left partially uninitialized, i.e., the processing element occupancy is low. This further leads to suboptimal floating point and memory performance. The situation is even worse in the complex valued case since the memory requirement is doubled but the number of parallel tasks stays the same.

The PCR method uses the same reduction formulas as the CR method but the reduction operation is applied to every row in each reduction step. Thus, all sub-systems are the same size and the arithmetical complexity of the method is $\mathcal{O}(m \log m)$. The benefits of using the PCR method are a more GPU-friendly memory access pattern, which does not cause additional memory bank conflicts, and high processing element occupancy. The method is widely used for solving tridiagonal systems on GPUs; see, e.g. [14, 45].

And finally, the Thomas method is a special variant of the well-known LU-decomposition method for tridiagonal matrices. The arithmetical complexity of the method is $\mathcal{O}(m)$ and it is the most effective of all the three mentioned methods. However, this sequential algorithm is not suitable for GPUs on its own. Thus, the method is often combined with the PCR method (see, e.g., [14, 24]). This is possible because a PCR reduction step splits a linear system into two independent sub-systems. After a few reduction steps, the remaining sub-systems can be solved effectively using the Thomas method.

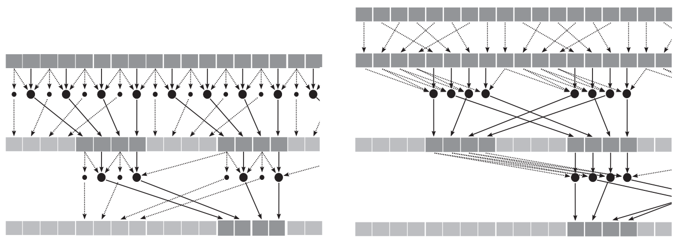


FIG. 3.1. *Examples of the permutation patterns during the stage A (on the left) and stage B (on the right) of the tridiagonal solver. The work group size is four.*

The solver has four main stages: **Stage A** is used when the reduced system does not fit into the allocated local memory buffer. It this case, the coefficient matrix and the right-hand side vector are stored in the global memory and the local memory is used to share odd numbered rows between work-items. The system is divided into sections which are then processed independently of each other. Each section is two times the size of the work group. After the new sub-system has been computed using the CR method, each section is permuted such that all odd numbered rows from the previous sub-system are stored in the upper half of the section and all rows from the new sub-system are stored in the lower half of the section. This division and permutation operation is repeated after each reduction step. Figure 3.1 shows how the reduction stage proceeds when the work group size is four. Each reduction and back substitution step can be performed by using multiple work groups concurrently, thus allowing larger number of work-items to be used per tridiagonal system.

Stage B is used when the reduced system fits into the allocated local memory but the number of remaining even numbered rows is bigger than the size of the used work group. The system is again divided into sections which are then processed independently of each other and each section is two times the size of the work group. Before the first reduction step, each section is permuted such that all odd numbered rows are stored in the upper half of the section and all even numbered rows are stored in the lower half of the section. Then all sections are processed in pairs. After the new sub-system has been computed using the CR method, each section pair is permuted such that all computed rows, that are going to be odd numbered rows during the next reduction step, are stored in the lower half of the first section and all computed rows, that are going to be even numbered rows during the next reduction step, are stored in the lower half of the second section. This division and permutation operation is repeated after each reduction step. This permutation pattern seems to be in some respects similar to the one used in [18]. Figure 3.1 shows how the reduction stage proceeds when the work group size is four.

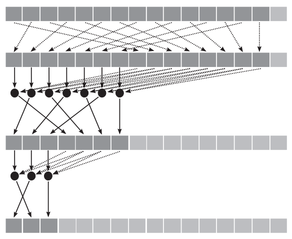


FIG. 3.2. *An example of the permutation pattern during the stage C of the tridiagonal solver. The work group size is eight.*

Stage C is used when the reduced system fits into the allocated local memory buffer and the number of remaining even numbered rows is at most the same as the size of the used work group. Before the first reduction step, the system is permuted such that all odd numbered rows are stored in the upper half of the vector and all even numbered rows are stored in the lower half of the vector. After the new subsystem has been computed using the CR method, each vector is permuted such that all computed rows, that are going to be even numbered rows during the next reduction step, are stored in the first part of the vector, followed by all computed rows, that are going to be odd numbered rows during the next reduction step. This division and permutation operation is repeated after each reduction step. This permutation pattern seems to be identical with the one used in [18].

Stage D performs the remaining system using a PCR-Thomas hybrid method similar to [14, 24]. This stage can be disabled completely or the Thomas-step can be skipped, if desired. In our implementation this is decided by the differential evolution optimization of the run time.

The permutations can be implemented effectively as the coefficient matrix is stored as three vectors and the reversal of the permutation pattern during the back substitution stage is necessary only in the case of the right-hand side vector. The tridiagonal solve presented in [31] consisted simplified versions of the stages A, B, and C.

In total, the tridiagonal solver consists of five kernels. If it is necessary to use the global memory, then the following four kernels are used: the first kernel (l3_gen_glo_sys) is responsible for forming the coefficient matrices into the global memory, the second kernel (l3_a1) performs one (or multiple if only one work group is used per system) reduction step, the third kernel (l3_a2) performs one (or multiple if only one work group is used per system) back substitution step, and the fourth kernel (l3_bcd_cpy_sys) copies the system into the local memory and performs the other stages. If the global memory is not needed, then only one kernel (l3_bcd_gen_sys) is used to performs all necessary operations. The work group sizes, the amount of allocated local memory, switching points between different stages, the number of work groups per system, and many other properties are parametrized and, thus, optimized.

4. Numerical results. The GPU implementation is tested with the following two test problems. The first test problem is a two- or three-dimensional Poisson equation with Dirichlet boundary conditions:

$$\begin{cases} - \triangle u = f, & \text{in } \Omega, \\ u = g, & \text{on } \partial\Omega. \end{cases} \tag{4.1}$$

In a rectangle the discretization using (mass lumped) linear finite-elements on an orthogonal finite-element mesh leads the following factor matrices:

$$\begin{aligned} A_j &= \operatorname{tridiag}\left\{ -\frac{1}{h_{j,l-1}}, \frac{h_{j,l}+h_{j,l+1}}{h_{j,l}h_{j,l+1}}, -\frac{1}{h_{j,l}} \right\} \in \mathbb{R}^{n_j \times n_j}, \\ M_j &= \operatorname{diag}\left\{ \frac{h_{j,l}+h_{j,l+1}}{2} \right\} \in \mathbb{R}^{n_j \times n_j}. \end{aligned} \tag{4.2}$$

Above, $h_{j,l}$ is the lth mesh step in the jth coordinate axis direction. In addition, $c = 0$ in (2.5) or (2.6). Bilinear (or trilinear) finite-elements could be used. In that case, the factor matrix M_j would be tridiagonal which would make the solution slightly slower.

The second test problem is an approximation of a Helmholtz equation

$$\begin{aligned} - \triangle u - \omega^2 u = f, \ \text{in } \mathbb{R}^d, \\ \lim_{r\to\infty} r^{(d-1)/2}\left(\frac{\partial u}{\partial r} - i\omega u \right) = 0, \end{aligned} \tag{4.3}$$

where $d = 2, 3$ and ω is the wave number. The second equation of (4.3) is the Sommerfeld radiation condition posing u to be a radiating solution. Here i denotes the imaginary unit. The unbounded domain $\mathbb{R}^d$ must be truncated to a finite one before finite element solution can be attempted. This means that we must approximate the radiation condition at the truncation boundary. Two popular ways of achieving this are a perfectly matched layer (PML) [4, 5] and an absorbing boundary condition (ABC) [3, 15, 17]. If the truncated domain is rectangular and the problem is discretized using bilinear (or trilinear) finite elements on an orthogonal finite-element mesh, then the resulting coefficient matrices can be presented using the Kronecker matrix tensor product in a form which is suitable for the PSCR method. The factor matrices and other details can be found in [19].

A second-order ABC was chosen for numerical experiments. For the Helmholtz equation, $c = -\omega^2$ in (2.5) or (2.6). The factor matrices are defined as

$$A_j = \begin{bmatrix} b_{j,1} & a_{j,1} & & \\ a_{j,1} & b_{j,2} & \ddots & \\ & \ddots & \ddots & a_{j,n_j-1} \\ & & a_{j,n_j-1} & b_{j,n_j} \end{bmatrix} \in \mathbb{C}^{n_j \times n_j} \tag{4.4}$$

and

$$M_j = \begin{bmatrix} d_{j,1} & c_{j,1} & & \\ c_{j,1} & d_{j,2} & \ddots & \\ & \ddots & \ddots & c_{j,n_j-1} \\ & & c_{j,n_j-1} & d_{j,n_j} \end{bmatrix} \in \mathbb{C}^{n_j \times n_j}, \tag{4.5}$$

n	CPU	GTX580	K40c	K40c/cuFFT
31	0.0002	0.0007 + 0.0001	0.0006 + 0.0001	0.0002
63	0.0005	0.0006 + 0.0001	0.0008 + 0.0001	0.0002
127	0.0019	0.0008 + 0.0002	0.0009 + 0.0002	0.0002
255	0.0073	0.0011 + 0.0005	0.0010 + 0.0006	0.0008
511	0.0344	0.0037 + 0.0021	0.0031 + 0.0021	0.0016
1023	0.1441	0.0176 + 0.0055	0.0126 + 0.0044	0.0080
2047	0.7072	0.0819 + 0.0165	0.0619 + 0.0303	0.0245
4095	2.9455	0.3384 + 0.0922	0.2454 + 0.0794	0.1619
8191	13.991	—	1.2558 + 0.5003	—

TABLE 4.1
Solution and right-hand side vector transfer times (in seconds) for the two-dimensional Poisson equations.

where

$$\begin{aligned} b_{j,l} &= \frac{h_{j,l-1}+h_{j,l}}{h_{j,l-1}h_{j,l}},\ l=2,3,\dots,n_j-1,\\ b_{j,1} &= \frac{1}{h_{j,1}}-\frac{i\omega}{2},\ b_{j,n_j}=\frac{1}{h_{j,n_j-1}}-\frac{i\omega}{2},\\ d_{j,l} &= \frac{h_{j,l-1}+h_{j,l}}{3},\ l=2,3,\dots,n_j-1,\\ d_{j,1} &= \frac{h_{j,1}}{3}+\frac{i}{2\omega},\ d_{j,n_j}=\frac{h_{j,n_j-1}}{3}+\frac{i}{2\omega},\\ a_{j,l} &= -\frac{1}{h_{j,l}},\ \text{ and }\ c_{j,l}=\frac{5h_{j,l}}{6}. \end{aligned} \tag{4.6}$$

4.1. Comparisons. The GPU tests presented in this paper were carried out on a Nvidia GeForce GTX580 GPU with 512 processing elements[1] and a Nvidia Tesla K40c GPU with 2880 single-precision processing elements and 960 double-precision processing elements. The first GPU is a few years old consumer level device which is not meant for general purpose computation. The reason to include this device is to demonstrate that our GPU implementation is capable of utilizing even a consumer level device in such a degree that it significantly outperforms a CPU. The later device is a high-end device aimed for computations with error checking & correction (ECC) memory. It is substantially more capable device than the GeForce-series device as it contains more than five times as many processing elements and more than seven times as much global memory. However, the theoretical peak global memory bandwidth has increased only by 50% when compared to the GTX580 GPU. The Tesla-series device could potentially allow us to solve much larger problems but unfortunately, due to the limitations of Nvidia's OpenCL implementation, it is only possible to access up to 4GB of global memory. GPU driver versions were 340.58 (64-bit Linux) for the GTX580 GPU and 331.75 (64-bit Linux) for the K40c GPU. The radix-4 PSCR CPU code used as a reference implementation is the one presented in [35]. It is written in Fortran and utilizes a single CPU core. CPU tests were carried out using an Intel Xeon E5-2630 v2 (2.60GHz) CPU. All the test were performed using double precision floating point arithmetic and the ECC memory was used when possible.

Tables 4.1 and 4.2 show the results obtained with the CPU and GPU implementations for the two-dimensional test problems. The initialization time is excluded

[1]Double precision performance of the Nvidia Geforce GTX580 GPU is 1/8 of single-precision performance.

n	CPU	GTX580	K40c
31	0.0005	0.0006 + 0.0001	0.0007 + 0.0001
63	0.0009	0.0006 + 0.0001	0.0006 + 0.0001
127	0.0055	0.0009 + 0.0003	0.0008 + 0.0003
255	0.0196	0.0024 + 0.0014	0.0018 + 0.0012
511	0.0972	0.0113 + 0.0036	0.0098 + 0.0038
1023	0.3927	0.0436 + 0.0086	0.0358 + 0.0090
2047	1.8860	0.1946 + 0.0465	0.1600 + 0.0615
4095	8.7141	0.8027 + 0.1851	0.6053 + 0.2487

TABLE 4.2
Solution and right-hand side vector transfer times (in seconds) for the two-dimensional Helmholtz equations.

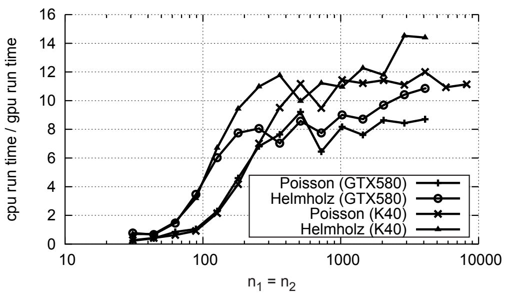


FIG. 4.1. *A run time comparison between the Intel Xeon CPU and the Nvidia GPUs for the two-dimensional problems.*

from the tabulated run times. The initialization time depends on the properties of the generalized eigenvalue problems but generally the initialization time is tens of times longer than the actual solution time. Figure 4.1 shows the speed up when the Nvidia GPUs are used instead of one core of the Intel Xeon CPU. The right-hand side vector transfer time is excluded from the comparison. For the Poisson equation, the GTX580 GPU is up to 9 times faster and the K40c GPU is up to 12 times faster. For the Helmholtz equation, the GTX580 GPU is up to 11 times faster and the K40c GPU is over 14 times faster. The size of the problem determines the number of work-items that can be used which further determines how effectively the GPU can be utilized. In addition, each kernel launch causes some additional overhead which has the largest impact on small problems. These two observations explains why the speedup goes up initially as the problem size increases.

Table 4.1 also includes an additional comparison against a cuFFT-based [12] Poisson solver [8, 9]. The right-hand side vector transfer time is excluded. The results show that the cuFFT-based solver is faster than the PSCR implementation even though it uses a 2D complex-to-complex FFT-transformation instead of a discrete sine transformation. However, the PSCR method can be applied to a much wider class of problems. The cuFFT-based solver requires that the equation has constant coefficients and it is posed with Dirichlet boundary conditions. Furthermore, the discretization grid must be equidistant.

Tables 4.3 and 4.4 show the corresponding results for the three-dimensional test problems. The initialization time is again excluded from the tabulated run times.

n	CPU	GTX580	K40c
31	0.0095	0.0028 + 0.0003	0.0035 + 0.0002
44	0.0266	0.0058 + 0.0009	0.0047 + 0.0012
63	0.0803	0.0114 + 0.0021	0.0084 + 0.0020
89	0.4028	0.0510 + 0.0049	0.0341 + 0.0054
127	1.2771	0.1063 + 0.0085	0.0828 + 0.0090
180	3.7289	0.2925 + 0.0346	0.2598 + 0.0455
255	10.977	0.8434 + 0.0634	0.6665 + 0.1209
361	47.380	—	3.3248 + 0.3576

Table 4.3

Solution and right-hand side vector transfer times times (in seconds) for the three-dimensional Poisson equations.

n	CPU	GTX580	K40c
31	0.0268	0.0067 + 0.0006	0.0038 + 0.0005
44	0.0774	0.0167 + 0.0021	0.0089 + 0.0013
63	0.2372	0.0332 + 0.0025	0.0173 + 0.0024
89	1.2201	0.1572 + 0.0064	0.0897 + 0.0066
127	3.7773	0.3818 + 0.0162	0.2445 + 0.0298
180	11.439	1.2079 + 0.0769	0.7880 + 0.0971
255	33.022	—	2.0755 + 0.2469

Table 4.4

Solution and right-hand side vector transfer times times (in seconds) for the three-dimensional Helmholtz equations.

Unlike in the case of two-dimensional test problems, the initialization time is generally neglectable when compared to the solution time. Figure 4.2 shows the speed up when the Nvidia GPUs are used instead of one core of the Intel Xeon CPU. In this case, the GPUs outperform the CPU even when the problem size is small. For the Poisson equation, the GTX580 GPU is almost 13 times faster and the K40c GPU is over 16 times faster. For the Helmholtz equation, the GTX580 GPU is up to 10 times faster and the K40c GPU is up to 16 times faster.

Figures 4.3 and 4.4 show how the run time was distributed among the kernels on the K40c GPU for the two- and three-dimensional Helmholtz equations. For the small two-dimensional problems, the run time is dominated by the overhead, as expected. For the larger two-dimensional problems, the run time is dominated by the tridiagonal solver. Therefore the effort of improving the implementation should be directed toward the tridiagonal solver. The transition point where the tridiagonal solver begins to use the global memory can again be seen clearly. For the three-dimensional problems, the overhead is small and the run time is again strongly dominated by the tridiagonal solver. The run time breakdowns for the GTX580 GPU and the Poisson equation are very similar with the difference that the overhead constituted a slightly larger portion of the total run time.

The GPU implementation is numerically as accurate as the CPU implementation. The parameter optimization provided some additional benefit usually of the order of 15% when compared to non-optimized generic parameters, which try to maximize GPU utilization by using large work groups and utilizing the local memory as much as possible. The Newton-Raphson division algorithm was used in almost all numerical tests and it increased the performance on average by 3.1% on the GTX580 GPU and 6.8% on the K40c GPU.

5. Roofline models.

The roofline model [44] is a performance analysis tool which takes into account the available off-chip memory bandwidth. The model gives

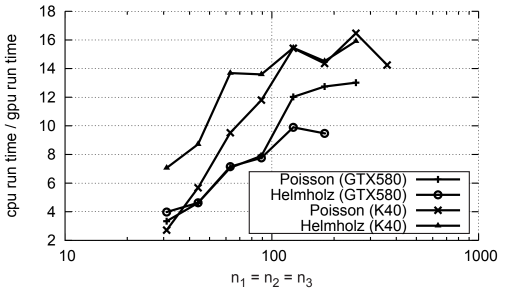


FIG. 4.2. *A run time comparison between the Intel Xeon CPU and the Nvidia GPUs for the three-dimensional test problems.*

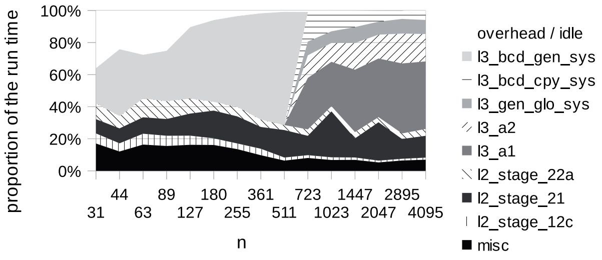


FIG. 4.3. *The run time distribution on the Nvidia Tesla K40c GPU for the two-dimensional Helmholtz equation.*

a much more accurate picture than only counting of the floating point operations (flop) would. This is especially true in the case of GPUs since the theoretical floating point performance can be very high but the actual attainable floating point performance is limited in several ways, including the global memory bandwidth. The roofline model has been previously successfully applied to GPUs; see, e.g., [25].

Basically, the application of the roofline model produces a two-dimensional scatter plot, assigning "operational intensity" to the horizontal axis, and "attained floating point performance" to the vertical axis. Here the operational intensity is defined as

$$\text{operational intensity} = \frac{\text{number of floating point operations}}{\text{number of bytes of (off-chip) memory traffic}}. \tag{5.1}$$

The model includes two device specific upper limits for the attainable floating point performance. The first upper limit is the theoretical peak floating point performance of the device. The second upper limit is determined by the peak (off-chip) memory bandwidth and calculated by

$$\text{peak (off-chip) memory bandwidth} \times \text{operational intensity}. \tag{5.2}$$

The point where these two upper bounds intersect is called device specific balance point. If the computational intensity of an algorithm is lower than the device specific

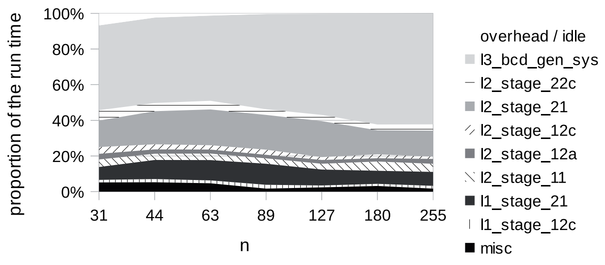


FIG. 4.4. *The run time distribution on the Nvidia Tesla K40c GPU for the three-dimensional Helmholtz equation.*

balance point, then the algorithm is considered to be memory-bound; otherwise the algorithm is considered to be compute-bound. The actual attained floating point performance should be close to the upper limit specified by the computational intensity of the algorithm.

The analysis presented in this paper is based on computing an estimate for the number of floating point and memory operations on a test run by test run basis. These estimates take into account the problem size and the parameters. Each Newton-Raphson division operation counts as 18 flops. The test run specific computational intensities, attained floating point performances, and average memory throughputs were then derived from these estimates.

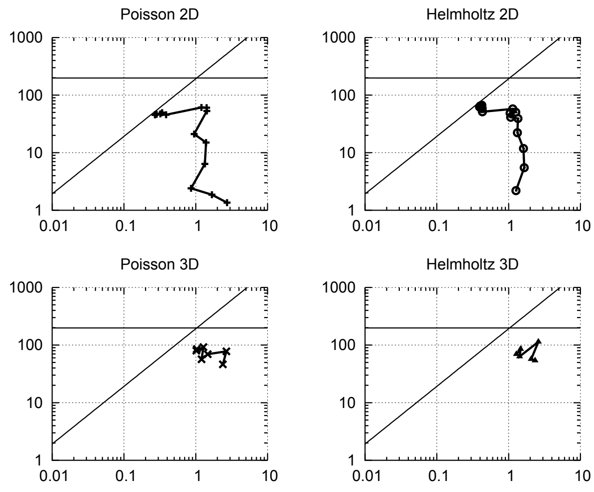


FIG. 5.1. *Global memory roofline models for the Nvidia Geforce GTX580 GPU.*

5.1. Global memory roofline models and throughput. Figure 5.1 shows the global memory roofline models for the GTX580 GPU. Based on the information provided by Nvidia, the theoretical peak floating point performance was estimated to be about 198 GFlop/s and the theoretical peak global memory bandwidth to be about 192 GB/s. This means that the device specific balance point is about 1.03 Flop/Byte. For the two-dimensional problems, the measurement points form two distinct clusters. For the Poisson equation, these clusters are located near 0.30 Flop/Byte and 1.30 Flop/Byte. The first cluster includes the cases in which the tridiagonal sub-problems are solved using the global memory and the second cluster includes the cases where the global memory is not used. This relatively large separation between the cluster is due to the fact that this analysis does not take into account the local memory throughput and, thus, the operational intensity is higher in the cases where the local memory is used more intensively. For the Helmholtz equation, the measurement points form similar cluster near 0.41 Flop/Byte and 1.17 Flop/Byte. In all cases, the models indicate that the algorithm is memory-bound for large problems and compute-bound for small problems. For the three-dimensional Poisson and Helmholtz equations, the clusters are located near 1.52 Flop/Byte and 1.86 Flop/Byte, respectively. These imply that the algorithm is compute-bound.

The roofline model explain the numerical results very well in the case of the large two-dimensional problems. However, it is clear that the attained floating point performance is not even half of the attainable floating point performance in other cases. Additional analyses, in which local memory intensive kernels were excluded from the estimates, produced roofline models where the clusters were located much closer to the global memory bandwidth related upper limit.

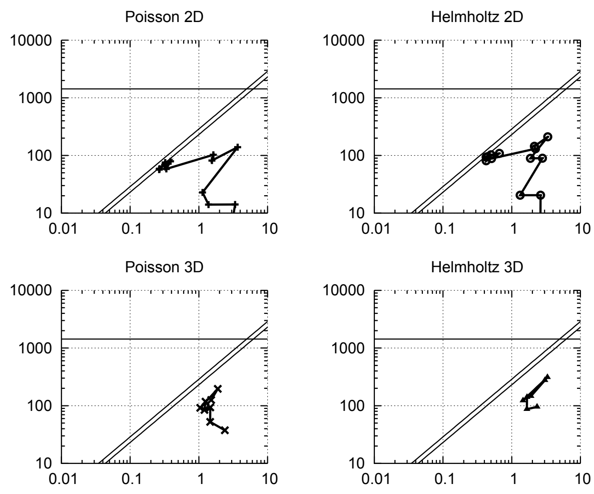


Fig. 5.2. *Global memory roofline models for the Nvidia Tesla K40c GPU.*

Figure 5.2 shows similar global memory roofline models for the K40c GPU. This time the theoretical peak floating point performance is about 1430 GFlop/s and the theoretical peak global memory bandwidth is about 288 GB/s when the ECC is turned off. This means that the device specific balance point is about 4.97 Flop/Byte. If the ECC is turned on, as it was during the test runs, the theoretical peak global memory bandwidth will drop. If it is assumed that this drop is about 20%, then the effective theoretical peak global memory bandwidth would be about 230 GB/s and the effective device specific balance point would be about 6.21 Flop/Byte. Figure 5.2 includes two global memory bandwidth related upper limits, one with ECC turned off and one with ECC turned on. Such a large device specific balance points mean that the task of harnessing all the available computing resources is going to be much harder when compared to the GTX580 GPU.

For the two-dimensional Poisson equation, the clusters are located near 0.30 Flop/Byte and 2.20 Flop/Byte. For the two-dimensional Helmholtz equation, the clusters are located near 0.50 Flop/Byte and 2.47 Flop/Byte. And finally, in the case of the three-dimensional Poisson and Helmholtz equations, the clusters are located near 1.34 Flop/Byte and 2.20 Flop/Byte, respectively. In all cases, the roofline models indicate that the algorithm is heavily memory-bound. This explains, at least in part, why the considerably more powerful computing orientated Tesla-series GPU did not outperform the consumer-level GeForce-series GPU in such a large extend as would have been expected.

When compared to the roofline model in Figure 5.1, it can be seen that some measurement points have shifted to the right. This is mainly due to the fact that the tridiagonal solver begins to utilize the PCR method at much earlier stages and the PCR method has higher arithmetical complexity than CR method. While this increases the overall arithmetical complexity of the implementation, it also streamlines the local memory usage. This benefits devices with high device specific balance points, such as the K40c GPU.

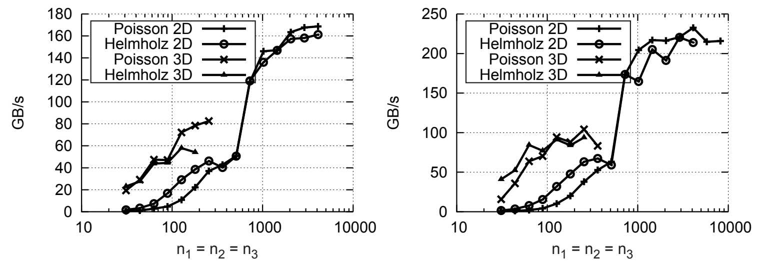


FIG. 5.3. *Average global memory throughputs for the Nvidia Geforce GTX580 GPU (on the left) and the Nvidia Tesla K40c (on the right) GPU.*

Figure 5.3 shows the average global memory throughputs for the GTX580 and K40c GPUs. The transition point where the tridiagonal solver begins to use the global memory can be seen clearly and it appear that the memory channels begin to saturate when the two-dimensional problems become large enough thus implying that the algorithm becomes memory-bound. These graphs are consistent with the global memory roofline models.

5.2. Local memory roofline models. The roofline models in Figures 5.1 and 5.2 provide reasonably accurate analysis when the global memory usage is dominating over the local memory usage. However, the local memory usage is also an important factor in particularly when the problem size is small or the problem is three-dimensional. In order to take this into account, the local memory usage of the kernels l3_bcd_cpy_sys and l3_bcd_gen_sys was analyzed in further detail.

Based on the hardware specifications and the documentation provided by Nvidia, the theoretical peak local memory bandwidth of the GTX580 GPU was estimated to be about

$$\begin{aligned} 16 \text{ computing units} \times \frac{32 \text{ banks}}{\text{computing unit}} \times \frac{4 \text{ bytes / 2 cycles}}{\text{bank}} \times 1.544 \text{ Ghz} \\ \approx 1581 \text{ GB/s}, \end{aligned} \tag{5.3}$$

which means that the device specific balance point is about 0.13 Flop/Byte. Similarly, if we assume that each memory bank has a bandwidth of 64-bits per cycle, as documented, then the theoretical peak local memory bandwidth of the K40c GPU would be about

$$\begin{aligned} 15 \text{ computing units} \times \frac{32 \text{ banks}}{\text{computing unit}} \times \frac{8 \text{ bytes / cycle}}{\text{bank}} \times 0.745 \text{ Ghz} \\ \approx 2861 \text{ GB/s}, \end{aligned} \tag{5.4}$$

which means that the device specific balance point is about 0.50 Flop/Byte. Our additional numerical experiments support these estimates.

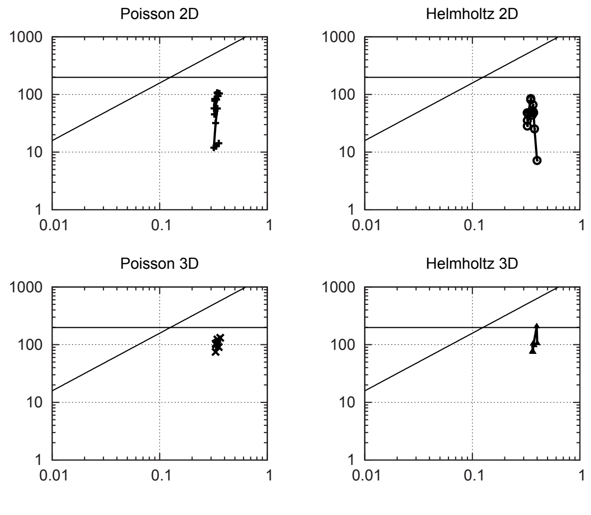


FIG. 5.4. *Local memory roofline models for the Nvidia Geforce GTX580 GPU.*

Figure 5.4 shows the obtained local memory roofline models for the GTX580 GPU. For the two-dimensional Poisson and Helmholtz equations and three-dimensional Poisson and Helmholtz equations the measurement points are located near 0.32 Flop/Byte, 0.36 Flop/Byte, 0.33 Flop/Byte, and 0.38 Flop/Byte, respectively. These imply that the algorithm is compute-bound in all cases. In the best cases, slightly more than half of the attainable floating point performance is actually utilized.

Figure 5.5 shows the obtained local memory roofline models for the K40c GPU. For the two-dimensional Poisson and Helmholtz equations and three-dimensional Poisson and Helmholtz equations the measurement points are located near 0.33 Flop/Byte,

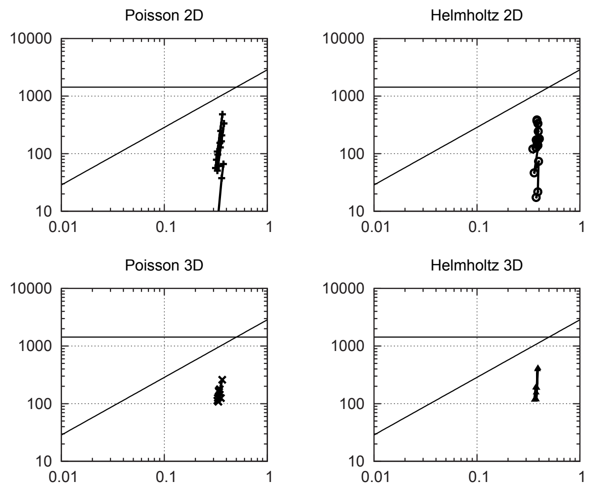


FIG. 5.5. *Local memory roofline models for the Nvidia Tesla K40c GPU.*

0.38 Flop/Byte, 0.33 Flop/Byte, and 0.38 Flop/Byte, respectively. These imply that the algorithm is slightly memory-bound in all cases. In the best cases, almost half of the attainable floating point performance is actually utilized. The significantly higher attained floating point performance when compared to the GTX580 GPU is mainly due to the increased utilization of the PCR method.

The above presented roofline models do not alone explain all the numerical results. Several explanations were considered ranging from memory bank conflicts to potential GPU driver problems and limitations. In the end, we came to the conclusion that the most important issue to consider in this regard is the limited amount of local memory available in contemporary GPUs (48kB), which has the unfortunate side effect of severely restricting the number of work groups that can be executed simultaneously in a computing unit. This leads to to suboptimal local memory performance as based on our measurements, the GTX580 GPU requires at least 256 active work-items and the K40c GPU at least 512 active work-items per computing unit in order to achieve the best local memory bandwidth.

In practice, the CR-PCR-Thomas hybrid tridiagonal solver is unable maintain the required processing element occupancy even though it is possible in principle by utilizing the PCR-Thomas hybrid method sufficiently. This is due to the fact that the increase in arithmetical complexity would negate any gains made. Actually, there was little discernible benefit from using the Thomas stage at all and in most cases only the CR and PCR stages were used. Previous findings suggest that the roofline model overestimates the amount of available local memory bandwidth and as a result, the amount of attainable floating point performance.

6. Conclusions.

This paper presented a generalized GPU implementation of the radix-4 PSCR method and numerical results obtained with the implementation for two test problems. The results indicate up to 16-fold speedups when compared to an equivalent CPU implementation utilizing a single CPU core. The highest speedups were obtained with the three-dimensional test problems. The numerical results indicate that the presented GPU implementation is effective and that GPUs provide demonstrable benefits in the context of the cyclic reduction and MDA methods. The presented roofline models indicate that the performance is, for the most part, limited by the available global memory bandwidth and the effectiveness of the tridiagonal solver used to solve the arising tridiagonal subproblems. The Newton-Raphson division algorithm improved the average performance almost by 7% and the differential evolution parameter optimization by 15%.

REFERENCES

[1] A.A. Abakumov, Yu. A. Yeremin, and Yu. A. Kuznetsov, *Efficient fast direct method of solving Poisson's equation on a parallelepiped and its implementation in an array processor*, Sov. J. Numer. Anal. Math. Model., 3 (1988), pp. 1–20.

[2] A. Banegas, *Fast Poisson solvers for problems with sparsity*, Math. Comput., 32 (1978), pp. 441–446.

[3] A. Bayliss, M. Gunzburger, and E. Turkel, *Boundary conditions for the numerical solution of elliptic equations in exterior regions*, SIAM J. Appl. Math., 42 (1982), pp. 430–451.

[4] J. Berenger, *A perfectly matched layer for the absorption of electromagnetic waves*, J. Comput. Phys., 114 (1994), pp. 185 – 200.

[5] ———, *Three-dimensional perfectly matched layer for the absorption of electromagnetic waves*, J. Comput. Phys., 127 (1996), pp. 363 – 379.

[6] B. Bialecki, G. Fairweather, and A. Karageorghis, *Matrix decomposition algorithms for elliptic boundary value problems: a survey*, Numer. Algorithms, 56 (2011), pp. 253–295.

[7] D. A. Bini and B. Meini, *The cyclic reduction algorithm: from Poisson equation to stochastic processes and beyond*, Numer. Algorithms, 51 (2008), pp. 23–60.

[8] F. Bleichrodt, *CuPoisson*. `https://code.google.com/p/cupoisson/`, 2013.

[9] F. Bleichrodt, R. H. Bisseling, and H. A. Dijkstra, *Accelerating a barotropic ocean model using a GPU*, Ocean Model., 41 (2012), pp. 16–21.

[10] O. Buneman, *A compact non-iterative Poisson solver*, Technical report 294, Institute for Plasma Research, Stanford University, Stanford, CA, 1969.

[11] S. D. Conte and C. de Boor, *Elementary numerical analysis: an algorithmic approach*, International series in pure and applied mathematics, McGraw-Hill, New York, Montreal, third ed., 1980.

[12] NVIDIA Corporation, *The Nvidia CUDA fast Fourier transform library (cuFFT)*. `https://developer.nvidia.com/cuFFT`, 2014.

[13] C. R. Crawford, *Reduction of a band-symmetric generalized eigenvalue problem*, Commun. ACM, 16 (1973), pp. 41–44.

[14] A. Davidson, Y. Zhang, and J. D. Owens, *An auto-tuned method for solving large tridiagonal systems on the GPU*, in Proceedings of the 25th IEEE International Parallel and Distributed Processing Symposium, IEEE, 2011, pp. 956–965.

[15] B. Engquist and A. Majda, *Absorbing boundary conditions for the numerical simulation of waves*, Math. Comput., 31 (1977), pp. 629–651.

[16] M. Flynn, *On division by functional iteration*, IEEE T. Comput., C-19 (1970), pp. 702–706.

[17] D. Givoli, *Non-reflecting boundary conditions*, J. Comput. Phys., 94 (1991), pp. 1 – 29.

[18] D. Göddeke and R. Strzodka, *Cyclic reduction tridiagonal solvers on GPUs applied to mixed precision multigrid*, IEEE T. Parall. Distr., Special issue: High performance computing with accelerators, 22 (2011), pp. 22–32.

[19] E. Heikkola, T. Rossi, and J. Toivanen, *Fast direct solution of the Helmholtz equation with a perfectly matched layer or an absorbing boundary condition*, Inter. J. Numer. Meth. Engrg., 57 (2003), pp. 2007–2025.

[20] D. Heller, *Some aspects of the cyclic reduction algorithm for block tridiagonal linear systems*, SIAM J. Numer. Anal., 13 (1976), pp. 484–496.

[21] R. W. Hockney, *A fast direct solution of Poisson's equation using Fourier analysis*, J. Assoc. Comput. Mach., 12 (1965), pp. 95–113.

[22] R. W. Hockney and C. R. Jesshope, *Parallel computers: architecture, programming and algorithms*, Bristol : Adam Hilger, 1981.

[23] M. Kass, A. Lefohn, and J. Owens, *Interactive depth of field using simulated diffusion on a GPU*. Available as Pixar Technical Memo 06-01, 2006.

[24] H.-S. Kim, S. Wu, L. Chang, and W. W. Hwu, *A scalable tridiagonal solver for GPUs*, 2013 42nd International Conference on Parallel Processing, (2011), pp. 444–453.

[25] K.-H. Kim, K. Kim, and Q.-H. Park, *Performance analysis and optimization of three-dimensional FDTD on GPU using roofline model*, Comput. Phys. Commun., 182 (2011), pp. 1201 – 1207.

[26] Yu. A. Kuznetsov, *Numerical methods in subspaces*, Vychislitel'-nye Processy i Sistemy II, 37 (1985), pp. 265–350.

[27] Yu. A. Kuznetsov and A. M. Matsokin, *On partial solution of systems of linear algebraic equations*, Sov. J. Numer. Anal. Math. Modelling, 4 (1989), pp. 453–468.

[28] Yu. A. Kuznetsov and T. Rossi, *Fast direct method for solving algebraic systems with separable symmetric band matrices*, East-West J. Numer. Math., 4 (1996), pp. 53–68.

[29] J. Lamas-Rodriguez, F. Arguello, D.B. Heras, and M. Boo, *Memory hierarchy optimization for large tridiagonal system solvers on GPU*, in IEEE 10th International Symposium on Parallel and Distributed Processing with Applications (ISPA), ASP-DAC '09, Piscataway, NJ, USA, 2012, IEEE, IEEE Press, pp. 87–94.

[30] M. Myllykoski and T. Rossi, *A parallel radix-4 block cyclic reduction algorithm*, Numer. Linear Algebra Appl., 21 (2014), pp. 540–556.

[31] M. Myllykoski, T. Rossi, and J. Toivanen, *Fast Poisson solvers for graphics processing units*, in Applied Parallel and Scientific Computing, Pekka Manninen and Per Öster, eds., vol. 7782 of Lecture Notes in Computer Science, Springer Berlin Heidelberg, 2013, pp. 265–279.

[32] A. Parker and J.O. Hamblen, *Optimal value for the Newton-Raphson division algorithm*, Inform. Process. Lett., 42 (1992), pp. 141 – 144.

[33] S. Petrova, *Parallel implementation of fast elliptic solver*, Parallel Comput., 23 (1997), pp. 1113–1128.

[34] T. Rossi and J. Toivanen, *A nonstandard cyclic reduction method, its variants and stability*, SIAM J. Matrix Anal. Appl., 20 (1999), pp. 628–645.

[35] ———, *A parallel fast direct solver for block tridiagonal systems with separable matrices of arbitrary dimension*, SIAM J. Sci. Comput., 20 (1999), pp. 1778–1796.

[36] H. Rutishauser, *Solution of eigenvalue problems with the LR-transformation*, U.S. Bur. Stand. Appl. Math. Ser., 49 (1958), p. 49.

[37] S. Sengupta, M. Harris, Y. Zhang, and J. D. Owens, *Scan primitives for gpu computing*, in Proceedings of the 22Nd ACM SIGGRAPH/EUROGRAPHICS Symposium on Graphics Hardware, GH '07, Aire-la-Ville, Switzerland, 2007, Eurographics Association, pp. 97–106.

[38] R. Storn, *On the usage of differential evolution for function optimization*, in Fuzzy Information Processing Society, 1996. NAFIPS., 1996 Biennial Conference of the North American, Jun 1996, pp. 519–523.

[39] R. Storn and K. Price, *Differential evolution — a simple and efficient heuristic for global optimization over continuous spaces*, J. Global Optim., 11 (1997), pp. 341–359.

[40] R. A. Sweet, *A cyclic reduction algorithm for solving block tridiagonal systems of arbitrary dimension*, SIAM J. Numer. Anal., 14 (1977), pp. 706–719.

[41] ———, *A parallel and vector variant of the cyclic reduction algorithm*, SIAM J. Sci. Statist. Comput., 9 (1988), pp. 761–765.

[42] P. Vassilevski, *Fast algorithm for solving a linear algebraic problem with separable variables*, C.R. Acad. Bulgare Sci., 37 (1984), pp. 305–308.

[43] P Vassilevski, *Fast algorithm for solving discrete poisson equation in a rectangle*, C.R. Acad. Bulgare Sci., 38 (1985), pp. 1311–1314.

[44] S. Williams, A. Waterman, and D. Patterson, *Roofline: An insightful visual performance model for multicore architectures*, Commun. ACM, 52 (2009), pp. 65–76.

[45] Y. Zhang, J. Cohen, and J. D. Owens, *Fast tridiagonal solvers on the GPU*, in Proceedings of the 15th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming PPoPP 10, no. 1, ACM Press, 2010, pp. 127–136.

PIV

A NEW AUGMENTED LAGRANGIAN APPROACH FOR L^1-MEAN CURVATURE IMAGE DENOISING

by

Mirko Myllykoski, Roland Glowinski, Tommi Kärkkäinen, and Tuomo Rossi

2015

SIAM Journal on Imaging Sciences, 8(1):95–125

SIAM J. IMAGING SCIENCES
Vol. 8, No. 1, pp. 95–125

A New Augmented Lagrangian Approach for L^1-mean Curvature Image Denoising[*]

M. Myllykoski[†], R. Glowinski[‡], T. Kärkkäinen[†], and T. Rossi[†]

Abstract. Variational methods are commonly used to solve noise removal problems. In this paper, we present an augmented Lagrangian-based approach that uses a discrete form of the L^1-norm of the mean curvature of the graph of the image as a regularizer, discretization being achieved via a finite element method. When a particular alternating direction method of multipliers is applied to the solution of the resulting saddle-point problem, this solution reduces to an iterative sequential solution of four subproblems. These subproblems are solved using Newton's method, the conjugate gradient method, and a partial solution variant of the cyclic reduction method. The approach considered here differs from existing augmented Lagrangian approaches for the solution of the same problem; indeed, the augmented Lagrangian functional we use here contains three Lagrange multipliers "only," and the associated augmentation terms are all quadratic. In addition to the description of the solution algorithm, this paper contains the results of numerical experiments demonstrating the performance of the novel method discussed here.

Key words. alternating direction methods of multipliers, augmented Lagrangian method, image denoising, image processing, mean curvature, variational model

AMS subject classifications. 68U10, 94A08, 53A05, 35A15

DOI. 10.1137/140962164

1. Introduction.

Let Ω be a bounded domain of $\mathbb{R}^2$ (a rectangle in practice). In the simplest form, denoising is a process in which a given noisy image $f : \Omega \to \mathbb{R}$ is separated into the actual image $u : \Omega \to \mathbb{R}$ and the remaining noise $g : \Omega \to \mathbb{R}$, that is, $f = u + g$. Variational methods, e.g., partial differential equation (PDE)–based (Euler–Lagrange equation) and nonlinear and nonsmooth optimization–based (corresponding energy functional), introduce a special family of techniques for image restoration and denoising in the general field of image processing and computer vision [14, 49]. A landmark in such techniques is the work by Perona and Malik related to anisotropic diffusion [41, 42]. Since then, many formulations and corresponding algorithms have been proposed, analyzed, realized, and utilized to improve the quality or understandability of digital images.

Let V be the space of restored functions, and let us consider the following minimization

[*]Received by the editors March 24, 2014; accepted for publication (in revised form) November 17, 2014; published electronically January 15, 2015.
http://www.siam.org/journals/siims/8-1/96216.html

[†]Department of Mathematical Information Technology, University of Jyväskylä, FI-40014, Jyväskylä, Finland (mirko.myllykoski@jyu.fi, tommi.karkkainen@jyu.fi, tuomo.j.rossi@jyu.fi). The research of the first author was supported by the Academy of Finland, grant 252549.

[‡]Department of Mathematics, University of Houston, Houston, TX 77204 (roland@math.uh.edu). The research of this author was supported by the Institute of Advanced Studies at the Hong Kong University of Science and Technology, and the Department of Mathematical Information Technology at the University of Jyväskylä.

problem:

$$\begin{cases} u \in V, \\ \mathcal{J}(u) \le \mathcal{J}(v) \quad \forall v \in V. \end{cases} \tag{1.1}$$

Typically

$$\mathcal{J}(v) = \mathcal{J}_f(v) + \varepsilon \mathcal{J}_r(v); \tag{1.2}$$

i.e., $\mathcal{J}$ consists of two terms, namely, (i) $\mathcal{J}_f(v)$, whose role is to fit (in a suitable norm) the denoised function to the noisy data f (the fidelity term), and (ii) the regularizing term $\varepsilon\mathcal{J}_r(v)$, where ε (> 0) is the regularization coefficient; examples of functionals $\mathcal{J}_r$ will be given below. The value of ε can be determined from the noise variance [10] if this information is known, or using some suitable heuristics [34].

Well-known variational approaches for image denoising relying on (1.2) include (but are not restricted to) the following:

BV (or TV) regularization: To be able to restore and denoise images with discontinuous intensity, a regularization using a norm that is not embedded in $C(\bar{\Omega})$ is needed. During the last two decades, the image denoising scene has been dominated by a method using such a regularization. The Rudin–Osher–Fatemi (ROF) method [44] relies on a discrete variant (obtained by finite difference discretization) of the minimization problem (1.1) with $\mathcal{J}$ defined by

$$\mathcal{J}(v) = \varepsilon \int_\Omega |\nabla v|\, dx + \frac{1}{2}\int_\Omega |f - v|^2 dx. \tag{1.3}$$

A natural candidate for the function space V is the space $BV(\Omega)$ of the functions with bounded variation over Ω. Problem (1.1) with $\mathcal{J}$ defined by (1.3) and close variants of it have motivated a large literature where their denoising properties, approximation, and iterative solution have been extensively discussed. We refer the reader to, e.g., [33, 34, 35, 45] and references therein for further information.

Euler's elastica: Euler's elastica as a prior curve model for image restoration was introduced in [39]. The work was continued, in connection with image inpainting, in [13]. In particular, Ambrosio and Masnou [1, 2] advocated using as regularizer the level set of v via the functional

$$\mathcal{J}_r(v) = \int_\Omega \left[a + b\left(\nabla \cdot \frac{\nabla v}{|\nabla v|}\right)^2 \right] |\nabla v| dx. \tag{1.4}$$

Let us return for a moment to the ROF model. Although the functional in (1.3) is convex, the nonreflexivity of the space $BV(\Omega)$ makes its analysis nontrivial, particularly the behavior of its solution when $\varepsilon \to 0$. This issue and many others, such as the derivation of dual formulations to (1.1) with $\mathcal{J}$ defined by (1.3), its discretization, the implementation, and the convergence of iterative solution methods (of the splitting type), are thoroughly discussed in [31].

Actually, as pointed out by [38] (see also [3], [12], and [51]), the ROF model has some significant drawbacks, such as the loss of image contrast, the smearing of corners, and the staircase effect. To remedy these unfavorable properties, several cures have been proposed (see [51] for a list of related references), among them the one introduced in [51], namely, instead of (1.1) with $\mathcal{J}$ defined by (1.3) use the following model:

$$u = \arg\min_{v \in V} \varepsilon \int_\Omega \left| \nabla \cdot \frac{\nabla v}{\sqrt{1 + |\nabla v|^2}} \right| dx + \frac{1}{s} \int_\Omega |f - v|^s \, dx, \tag{1.5}$$

commonly known these days as the L^1-mean curvature denoising model; $s \geq 1$, $s = 2$ being the most common choice. The fidelity term in (1.5) is of the simplest form compared to the proposed formulations. In particular, as depicted in [40], one can, via adding two linear transformations to this model, address other important image processing tasks related to deconvolution, inpainting, and superresolution. The rationale for (1.5) is discussed in much detail in [51]. However, to the best of our knowledge, the variational problem (1.5) is not yet fully understood mathematically due to the nonconvexity and nonsmoothness of the functional in (1.5) and to the fact that a natural choice for V is not clear. Concerning this last issue, taking $V = BV(\Omega)$ is suggested in [51]. We have, however, a problem with such a choice since we think that the definition of V has to include conditions on the second derivatives. We can only hope that a variant of [31] dedicated to problem (1.5) will appear in the near future.

Considering the above situation, our goals in this paper are more modest and purely finite dimensional algorithmic. They can be summarized as follows:

1. Taking advantage of the relative simplicity of the formalism of the continuous problem, we derive in section 2 a (necessarily formal) augmented Lagrangian algorithm. Our algorithm is a simplified variant of the one considered in [52] since we use a projection on a nonconvex set to treat a nonlinear constraint instead of treating it by penalty-duality, which would imply one extra augmentation functional and the related Lagrange multiplier. Thus, our algorithms involve three augmentation terms instead of four and three Lagrange multipliers instead of four. Indeed, when several Lagrange multipliers are used, one of the main issues is their adjustment to optimize convergence. We will return to the details of this reduction in section 2.
2. Taking advantage of the augmented Lagrangian algorithm described in section 2, we define in section 3 a discrete version of this algorithm to be applied to a (kind of) mixed finite element approximation of problem (1.5). We choose finite element methods for approximating the problem instead of finite differences since, when applied on uniform triangulations (like the one in Figure 1), in particular, these finite element methods automatically generate finite difference approximations with, among other attributes, good accuracy, stability, and monotonicity properties. Moreover, the variational derivation of Galerkin/finite element approximations (like the one we use here) makes them the perfect match for the solution of problems from Calculus of Variations, such as (1.1) with $\mathcal{J}$ defined by (1.3) and (1.5) (see [24] for other examples). Another advantage of finite element approximations is their ability to handle nonuniform meshes, adaptively or not, which may be of interest for some applications. More precisely, the minimization of the discrete counterpart of the functional in (1.5)

is transformed into the iterative sequential solution of four subproblems: one being smooth but nonlinear in $\mathbb{R}^2$, one purely explicit vertexwise, and two linear with positive definite and symmetric coefficient matrices of a scalar- (assuming $s = 2$) and vector-valued type.

3. In section 5 we apply the algorithms defined in section 4 to the solution of a variety of test problems of variable complexity in order to evaluate the capability of the methodology discussed in this paper. The actual solution process is somewhat simplified by assuming that $s = 2$ (which was also recommended by Zhu and Chan [51]).

2. Augmented Lagrangian formulation and basic solution algorithm.

2.1. Some preliminary observations. Despite the fact that problem (1.5) is not fully understood mathematically, we are going to take advantage of the simplicity of its formalism to derive a formal solution algorithm of the augmented Lagrangian type. This algorithm will be useful since in section 3 we will take it as a model to define a finite dimensional analogue dedicated to the solution of a finite element approximation of problem (1.5).

Actually, augmented Lagrangian techniques have a well-established role in analyzing constrained optimization problems as well as in deriving general solution algorithms for such problems [5, 10, 22, 25, 29, 32]. With BV-regularization a reformulation with an augmented Lagrangian method can introduce one or two new variables to deal with ∇u in the nonsmooth regularization term (e.g., [15, 48, 40] and the references provided in the reviews therein) or additionally to represent u in the fidelity term [11, 50]. In the latter case, three subproblems (alternating directions) typically appear and are then solved using linear solvers, explicit formulae (projection or shrinking), and nonlinear optimization methods (for nonsmooth fidelity). Moreover, when more than one additional variable is introduced, one typically applies varying regularization parameters for the penalized constraint (e.g., [11, 50]). More examples where many variational formulations (including Euler's elastica) for image processing have been efficiently treated using augmented Lagrangian approaches can be found in, e.g., [20, 30, 46, 50].

As already mentioned in section 1, an augmented Lagrangian algorithm was used in [52] for the solution of (1.5). The augmented Lagrangian approach we apply in this paper is of the ALG-2 type [22, 29] (better known as ADMM for alternating direction methods of multipliers). The basic idea of ADMM and the convergence proof in a convex situation were presented in the 1970s by Glowinski and Marrocco [27] and Gabay and Mercier [23]. Augmented Lagrangian methods, with partly similar ingredients for the solution of other challenging problems, are described, e.g., in [8, 16, 17, 19, 18, 21, 28]; most of these problems are nonconvex, as is the one discussed here.

The solution method in [52] is also of the ALG-2 (or ADMM) type, but it uses a different (and, we think, more complicated) augmented Lagrangian functional as the functional involves four augmentation functionals and four Lagrange multipliers. What made us uncomfortable was the Lagrange multiplier treatment of the nonlinear nonsmooth condition $|\hat{\mathbf{q}}_1| - \hat{\mathbf{q}}_1 \cdot \hat{\mathbf{q}}_2 = 0$, where $\hat{\mathbf{q}}_1$ and $\hat{\mathbf{q}}_2$ belong (formally) to $(L^2(\Omega))^3$. The related condition in our approach is

$$\mathbf{q}_2 = \frac{\mathbf{q}_1}{\sqrt{1 + |\mathbf{q}_1|^2}}, \tag{2.1}$$

where $\mathbf{q}_1$ and $\mathbf{q}_2$ belong (formally) to $(L^2(\Omega))^2$. The discrete analogue of the aforementioned equality will be treated by projection, avoiding those two terms associated with it in the augmented Lagrangian and reducing to three the number of Lagrange multipliers. This is a notable simplification considering that the adjustment of these parameters is one of the main issues associated with ADMM-type methods (it is discussed in [19] in a particular case). Also, in our approach, all the constraints treated by penalty-duality are linear, which is not the case in [52] (one of the constraints there is not only nonlinear but also nonsmooth).

Remark 1. At present, there is no available theory (as far as we know) for the convergence of ADMM methods for nonconvex problems, even in a finite dimension. Currently, the most popular publication concerning augmented Lagrangian and ADMM algorithms is certainly [7], a large review article (>100 pages) uniquely concerned with finite dimensional problems. The part of the article dedicated to nonconvex problems covers four (inconclusive) pages, suggesting that convergence proofs are difficult to obtain in most nonconvex cases. However, various investigations concerning nonconvex problems and comparisons with known solutions have shown the capability of augmented Lagrangian methods at solving nonconvex problems (as shown, for example, in [18] and [29]). This certainly encouraged us to apply this methodology to the solution of problem (1.5).

2.2. An augmented Lagrangian formulation. Assuming that a minimizer exists, the minimization problem (1.5) is clearly equivalent to

$$\begin{cases} (u, \mathbf{p}_1, \mathbf{p}_2, \mathbf{p}_3, \psi) \in E, \\ j(u, \mathbf{p}_1, \mathbf{p}_2, \mathbf{p}_3, \psi) \leq j(v, \mathbf{q}_1, \mathbf{q}_2, \mathbf{q}_3, \varphi) \quad \forall\, (v, \mathbf{q}_1, \mathbf{q}_2, \mathbf{q}_3, \varphi) \in E, \end{cases} \tag{2.2}$$

where

$$j(v, \mathbf{q}_1, \mathbf{q}_2, \mathbf{q}_3, \varphi) = \varepsilon \int_\Omega |\varphi|\, dx + \frac{1}{s} \int_\Omega |f - v|^s\, dx. \tag{2.3}$$

Above,

$$E = \Bigg\{ (v, \mathbf{q}_1, \mathbf{q}_2, \mathbf{q}_3, \varphi) : v \in V, (\mathbf{q}_1, \mathbf{q}_2) \in (L^2(\Omega))^{2\times 2}, \mathbf{q}_3 \in H(\Omega; \mathrm{div}), \\ \varphi \in L^2(\Omega),\ \mathbf{q}_1 = \nabla v,\ \mathbf{q}_2 = \frac{\mathbf{q}_1}{\sqrt{1 + |\mathbf{q}_1|^2}},\ \mathbf{q}_3 = \mathbf{q}_2,\ \varphi = \nabla\cdot \mathbf{q}_3 \Bigg\}, \tag{2.4}$$

where

$$H(\Omega; \mathrm{div}) = \left\{ \mathbf{q} \in (L^2(\Omega))^2 : \nabla\cdot \mathbf{q} \in L^2(\Omega) \right\}. \tag{2.5}$$

Remark 2. In section 1, we mentioned that the choice of V in the denoising model (1.5) is a critical issue. Actually, a reasonable candidate is (for its simplicity) the Sobolev space $W^{2,1}(\Omega)$ since the two terms defining the cost functional in (1.5) make sense in that space (we recall that from the Rellich–Kondrachov compact imbedding theorem the injection of $W^{2,1}(\Omega)$ in $L^q(\Omega)$ is compact $\forall q \in [1, +\infty)$). From a practical (but formal) point of view, there is an advantage to taking $V = H^2(\Omega)(= W^{2,2}(\Omega))$ for the following reasons: (i) $H^2(\Omega)$ is a Hilbert space; (ii) $H^2(\Omega)$ being dense in $W^{2,1}(\Omega)$, one obtains the same infimium when one minimizes the cost functional in (1.5) over these two spaces; and (iii) the above two spaces lead to the same discrete minimization problem.

Let us define

$$\Upsilon = [V \times \mathbf{E}_{12} \times H(\Omega; \mathrm{div}) \times L^2(\Omega)] \times [(L^2(\Omega))^2 \times (L^2(\Omega))^2 \times L^2(\Omega)], \tag{2.6}$$

with

$$\mathbf{E}_{12} = \left\{ (\mathbf{q}_1, \mathbf{q}_2) \in \left(L^2(\Omega)\right)^{2\times 2} : \mathbf{q}_2 = \frac{\mathbf{q}_1}{\sqrt{1+|\mathbf{q}_1|^2}} \right\}. \tag{2.7}$$

With problem (2.2) we associate the following augmented Lagrangian functional $\mathcal{L}_{\mathbf{r}} : \Upsilon \to \mathbb{R}$ (with $\mathbf{r} = (r_1, r_2, r_3), r_i > 0\ \forall i = 1, 2, 3$):

$$\begin{aligned} \mathcal{L}_{\mathbf{r}}(v, \mathbf{q}_1, \mathbf{q}_2, \mathbf{q}_3, \varphi; \boldsymbol{\mu}_1, \boldsymbol{\mu}_2, \mu_3) = \varepsilon \int_\Omega |\varphi| dx + \frac{1}{s} \int_\Omega |f - v|^s dx \\ + \frac{r_1}{2} \int_\Omega |\nabla v - \mathbf{q}_1|^2 dx + \int_\Omega \boldsymbol{\mu}_1 \cdot (\nabla v - \mathbf{q}_1) dx \\ + \frac{r_2}{2} \int_\Omega |\mathbf{q}_2 - \mathbf{q}_3|^2 dx + \int_\Omega \boldsymbol{\mu}_2 \cdot (\mathbf{q}_2 - \mathbf{q}_3) dx \\ + \frac{r_3}{2} \int_\Omega |\nabla \cdot \mathbf{q}_3 - \varphi|^2 dx + \int_\Omega \mu_3 (\nabla \cdot \mathbf{q}_3 - \varphi) dx, \end{aligned} \tag{2.8}$$

where $(\mathbf{q}_1, \mathbf{q}_2) \in \mathbf{E}_{12}$.

Now, suppose that the augmented Lagrangian $\mathcal{L}_{\mathbf{r}}$ has a saddle-point

$$\omega = (u, \mathbf{p}_1, \mathbf{p}_2, \mathbf{p}_3, \psi; \boldsymbol{\lambda}_1, \boldsymbol{\lambda}_2, \lambda_3) \in \Upsilon, \tag{2.9}$$

that is

$$\begin{aligned} \mathcal{L}_{\mathbf{r}}(u, \mathbf{p}_1, \mathbf{p}_2, \mathbf{p}_3, \psi; \boldsymbol{\mu}_1, \boldsymbol{\mu}_2, \mu_3) \leq \mathcal{L}_{\mathbf{r}}(u, \mathbf{p}_1, \mathbf{p}_2, \mathbf{p}_3, \psi; \boldsymbol{\lambda}_1, \boldsymbol{\lambda}_2, \lambda_3) \\ \leq \mathcal{L}_{\mathbf{r}}(v, \mathbf{q}_1, \mathbf{q}_2, \mathbf{q}_3, \varphi; \boldsymbol{\lambda}_1, \boldsymbol{\lambda}_2, \lambda_3), \end{aligned} \tag{2.10}$$

for all $(v, \mathbf{q}_1, \mathbf{q}_2, \mathbf{q}_3, \varphi; \boldsymbol{\mu}_1, \boldsymbol{\mu}_2, \mu_3) \in \Upsilon$.

It can be easily shown that if ω is a saddle-point of $\mathcal{L}_{\mathbf{r}}$ over Υ, then u is a solution of the minimization problem (1.5) and

$$\mathbf{p}_1 = \nabla u,\ \mathbf{p}_2 = \frac{\mathbf{p}_1}{\sqrt{1+|\mathbf{p}_1|^2}},\ \mathbf{p}_3 = \mathbf{p}_2,\ \text{and } \psi = \nabla \cdot \mathbf{p}_3. \tag{2.11}$$

2.3. The basic algorithm. A natural candidate for the solution of the saddle-point problem (2.9)–(2.10) is a particular ADMM called ALG-2 by various practitioners (see, e.g., [4, 22, 24, 29]). Among the several algorithms of the ALG-2 type, the following is considered in this paper.

Algorithm 1.

```
Input: $f, \varepsilon, s, \mathbf{r}, N$
Initialize($u^0, \mathbf{p}_1^0, \mathbf{p}_2^0, \mathbf{p}_3^0, \psi^0; \boldsymbol{\lambda}_1^0, \boldsymbol{\lambda}_2^0, \lambda_3^0$)
for $n = 0, \ldots, N$ do

    (2.12)    $(\mathbf{p}_1^{n+1}, \mathbf{p}_2^{n+1}) = arg\ min_{(\mathbf{q}_1,\mathbf{q}_2)\in\mathbf{E}_{12}} \mathcal{L}_\mathbf{r}\left(u^n, \mathbf{q}_1, \mathbf{q}_2, \mathbf{p}_3^n, \psi^n; \boldsymbol{\lambda}_1^n, \boldsymbol{\lambda}_2^n, \lambda_3^n\right)$
    (2.13)    $\mathbf{p}_3^{n+1} = arg\ min_{\mathbf{q}_3\in H(\Omega;div)} \mathcal{L}_\mathbf{r}\left(u^n, \mathbf{p}_1^{n+1}, \mathbf{p}_2^{n+1}, \mathbf{q}_3, \psi^n; \boldsymbol{\lambda}_1^n, \boldsymbol{\lambda}_2^n, \lambda_3^n\right)$
    (2.14)    $\psi^{n+1} = arg\ min_{\varphi\in L^2(\Omega)} \mathcal{L}_\mathbf{r}\left(u^n, \mathbf{p}_1^{n+1}, \mathbf{p}_2^{n+1}, \mathbf{p}_3^{n+1}, \varphi; \boldsymbol{\lambda}_1^n, \boldsymbol{\lambda}_2^n, \lambda_3^n\right)$
    (2.15)    $u^{n+1} = arg\ min_{v\in V} \mathcal{L}_\mathbf{r}\left(v, \mathbf{p}_1^{n+1}, \mathbf{p}_2^{n+1}, \mathbf{p}_3^{n+1}, \psi^{n+1}; \boldsymbol{\lambda}_1^n, \boldsymbol{\lambda}_2^n, \lambda_3^n\right)$
              $\boldsymbol{\lambda}_1^{n+1} = \boldsymbol{\lambda}_1^n + r_1(\nabla u^{n+1} - \mathbf{p}_1^{n+1})$
              $\boldsymbol{\lambda}_2^{n+1} = \boldsymbol{\lambda}_2^n + r_2(\mathbf{p}_2^{n+1} - \mathbf{p}_3^{n+1})$
              $\lambda_3^{n+1} = \lambda_3^n + r_3(\nabla\cdot\mathbf{p}_3^{n+1} - \psi^{n+1})$

    if stopping criterion is satisfied then
        return $(u^{n+1}, \mathbf{p}_1^{n+1}, \mathbf{p}_2^{n+1}, \mathbf{p}_3^{n+1}, \psi^{n+1})$
    end if
end for
return ERROR
```

A more explicit formulation of subproblem (2.12) reads as follows:

$$(\mathbf{p}_1^{n+1}, \mathbf{p}_2^{n+1}) = \text{arg min}_{(\mathbf{q}_1,\mathbf{q}_2)\in\mathbf{E}_{12}}\left[\frac{1}{2}\int_\Omega \left(r_1|\mathbf{q}_1|^2 + r_2|\mathbf{q}_2|^2\right)dx - \int_\Omega (r_1\nabla u^n + \boldsymbol{\lambda}_1^n)\cdot\mathbf{q}_1 dx - \int_\Omega (r_2\mathbf{p}_3^n - \boldsymbol{\lambda}_2^n)\cdot\mathbf{q}_2 dx\right]. \tag{2.16}$$

Similarly, the minimization problem (2.13) is equivalent to the following well-posed linear variational problem in $H(\Omega;\text{div})$:

$$\mathbf{p}_3^{n+1} \in H(\Omega;\text{div}),$$
$$r_2\int_\Omega \mathbf{p}_3^{n+1}\cdot\mathbf{q}\,dx + r_3\int_\Omega \nabla\cdot\mathbf{p}_3^{n+1}\nabla\cdot\mathbf{q}\,dx = \int_\Omega \left(r_2\mathbf{p}_2^{n+1} + \boldsymbol{\lambda}_2^n\right)\cdot\mathbf{q}\,dx + \int_\Omega \left(r_3\psi^n - \lambda_3^n\right)\nabla\cdot\mathbf{q}\,dx,\ \forall\mathbf{q}\in H(\Omega;\text{div}). \tag{2.17}$$

Next, a more explicit formulation of the minimization problem (2.14) is given by

$$\psi^{n+1} = \text{arg min}_{\varphi\in L^2(\Omega)}\left[\varepsilon\int_\Omega|\varphi|\,dx + \frac{r_3}{2}\int_\Omega|\varphi|^2\,dx - \int_\Omega\left(r_3\nabla\cdot\mathbf{p}_3^{n+1} + \lambda_3^n\right)\varphi\,dx\right]. \tag{2.18}$$

Finally, the minimization problem (2.15) is nothing but equivalent to the following well-posed, nonlinear, elliptic, variational problem (linear if $s = 2$):

$$(2.19)\qquad \begin{aligned} &u^{n+1} \in V, \\ &r_1 \int_\Omega \nabla u^{n+1} \cdot \nabla v \, dx + \int_\Omega \left|u^{n+1} - f\right|^{s-2} (u^{n+1} - f) \, v \, dx \\ &\qquad = \int_\Omega \left(r_1 \mathbf{p}_1^{n+1} - \boldsymbol{\lambda}_1^n\right) \cdot \nabla v \, dx \quad \forall v \in V. \end{aligned}$$

Remark 3. Assuming the minimizing sequences converge to a limit, we do not know in which space the related weak convergence takes place and if the functional under consideration has the weak lower semicontinuity property necessary to guaranty the convergence to a minimizer. Indeed, since our main concern is mostly to find a simpler alternative to the method discussed in [52], we skip this theoretical aspect of the problem.

Remark 4. As mentioned in the introduction, the augmented Lagrangian approach discussed in this section is largely formal, unlike its finite element realization discussed in section 3. This formality yields to various variational crimes, one of them being to take φ in $L^2(\Omega)$, and consequently $\mathbf{q}_3$ in $H(\Omega; \mathrm{div})$, while the natural functional space for φ is obviously $L^1(\Omega)$. Actually, a similar variational crime is committed when associating (as done by various practitioners today) with the functional $\mathcal{J}$ defined by (1.3) the augmented Lagrangian

$$(2.20)\qquad \begin{aligned} \mathcal{L}_r(v, \mathbf{q}; \boldsymbol{\mu}) = \varepsilon \int_\Omega |\mathbf{q}| \, dx + \frac{1}{2} \int_\Omega |f - v|^2 \, dx \\ + \frac{r}{2} \int_\Omega |\nabla v - \mathbf{q}|^2 \, dx + \int_\Omega \boldsymbol{\mu} \cdot (\nabla v - \mathbf{q}) \, dx, \end{aligned}$$

which is well suited to operations in $H^1(\Omega)$, but definitely not in $BV(\Omega)$, which is the natural space in which to minimize the above functional $\mathcal{J}$. Of course, the finite dimensional analogues of (2.20) make sense, authorizing, for example, the use of ADMM to solve the corresponding minimization problem.

3. Finite element realization.

3.1. Generalities. The rationale for using finite elements instead of finite differences was given in the introduction. An inspection of relations (2.16)–(2.19) shows that none of them explicitly involve derivatives of an order higher than one, implying that finite element spaces consisting of piecewise polynomial functions are well suited for defining a discrete analogue of Algorithm 1. Moreover, the expected lack of smoothness of the solutions (or quasi solutions) of problem (1.5) strongly suggests employing low degree polynomials (of a degree typically less than or equal to one). Actually, the approximation we will use is of the mixed type, like those used, for example, in [9, 16, 21, 31]; it allows solving a nonsmooth fourth-order elliptic problem using approximation spaces commonly used for the solution of second-order elliptic problems. For a thorough discussion of mixed finite element methods and some applications, see [6].

Concerning the solution of problem (1.5), we assume that Ω is a rectangle and denote by $\partial\Omega$ the boundary of Ω. Since Ω is polygonal, we can triangulate it using a standard finite element triangulation $\mathcal{T}_h$ (verifying therefore the assumptions listed in, e.g., [24]). A typical finite element triangulation (uniform here) is shown in Figure 1.

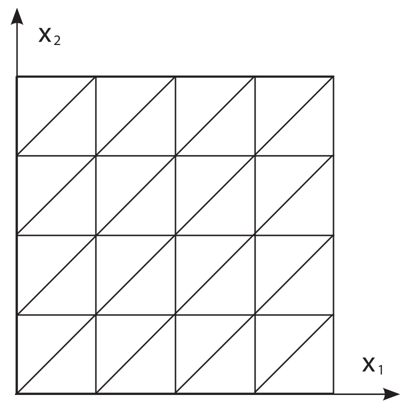


Figure 1. *A uniform triangulation of* $\Omega = (0,1)^2$.

We denote by Σ_h (respectively, Σ_{0h}) the finite set of the vertices of $\mathcal{T}_h$ (respectively, the finite set of the vertices of $\mathcal{T}_{0h}$ that do not belong to $\partial\Omega$). From now on we will assume that

$$\Sigma_{0h} = \{P_j\}_{j=1}^{N_{0h}} \text{ and } \Sigma_h = \Sigma_{0h} \cup \{P_j\}_{j=N_{0h}+1}^{N_h}, \tag{3.1}$$

where N_{0h} (respectively, N_h) is the number of elements of Σ_{0h} (respectively, Σ_h). Finally, we denote by Ω_j the polygon that is the union of those triangles of $\mathcal{T}_h$ that have P_j as a common vertex, and by $|\Omega_j|$ the measure of Ω_j.

3.2. Fundamental finite element spaces and the discrete divergence operator. Following Remark 2, we assume from now on that $V = H^2(\Omega)$. Using the appropriate discrete Green's formula, there is no difficulty in approximating the saddle-point problem (2.9)–(2.10) using classical C^0-conforming finite element spaces. To approximate the spaces $H^1(\Omega)$ and $H^2(\Omega)$, we will use

$$V_h = \left\{ v \in C^0(\bar{\Omega}) : v|_T \in P_1 \ \forall\, T \in \mathcal{T}_h \right\}. \tag{3.2}$$

Above, P_1 is the space of the polynomials of two variables of degree less than or equal to one.

Now, for $j = 1, \dots, N_h$, let us uniquely define the shape function w_j associated with the vertex P_j by

$$\begin{cases} w_j \in V_h, \\ w_j(P_j) = 1, \\ w_j(P_k) = 0 \ \forall k, \ 1 \le k \le N_h, \ k \neq j. \end{cases} \tag{3.3}$$

The set $\mathcal{B}_h = \{w_j\}_{j=1}^{N_h}$ is a vector basis of V_h, and we have

$$v = \sum_{j=1}^{N_h} v(P_j) w_j \quad \forall v \in V_h. \tag{3.4}$$

Other finite element spaces will prove useful in what follows. The first, denoted by V_{0h}, is the subspace of V_h consisting of the functions vanishing on $\partial\Omega$, that is,

$$V_{0h} = \{v \in V_h : v(P_j) = 0 \, \forall j = N_{0h} + 1, \dots, N_h\}\,; \tag{3.5}$$

we clearly have

$$v = \sum_{j=1}^{N_{0h}} v(P_j) w_j \quad \forall v \in V_{0h}. \tag{3.6}$$

The other space, denoted by $\mathbf{Q}_h$, is defined by

$$\mathbf{Q}_h = \left\{\mathbf{q} \in (L^\infty)^2 : \mathbf{q}|_T \in (P_0)^2 \; \forall \, T \in \mathcal{T}_h\right\}, \tag{3.7}$$

where P_0 is the space of those polynomials that are constant. We clearly have

$$\mathbf{q} = \sum_{T \in \mathcal{T}_h} (\mathbf{q}|_T)_{\chi_T} \quad \forall \mathbf{q} \in \mathbf{Q}_h, \tag{3.8}$$

where χ_T is the characteristic function of T and

$$\nabla V_h \subset \mathbf{Q}_h. \tag{3.9}$$

The linear space $\mathbf{Q}_h$ is a suitable candidate for the approximation of the space $H(\Omega; \mathrm{div})$, the main issue being to properly approximate the divergence of an arbitrary element of $\mathbf{Q}_h$. Suppose that $\mathbf{q} \in H(\Omega; \mathrm{div})$ and $v \in H_0^1(\Omega)$; we have (from the divergence theorem)

$$\int_\Omega \nabla\cdot \mathbf{q} \, v \, dx = -\int_\Omega \mathbf{q} \cdot \nabla v \, dx \quad \forall (v, \mathbf{q}) \in H_0^1(\Omega) \times H(\Omega; \mathrm{div}). \tag{3.10}$$

Suppose now that $\mathbf{q} \in \mathbf{Q}_h$; relation (3.10) suggests defining the discrete divergence operator div_h by: $\forall \mathbf{q} \in \mathbf{Q}_h$ we have

$$\begin{aligned} &\mathrm{div}_h \mathbf{q} \in V_{0h}, \\ &\int_\Omega (\mathrm{div}_h \mathbf{q}) \, v \, dx = -\int_\Omega \mathbf{q} \cdot \nabla v \, dx \quad \forall v \in V_{0h}, \end{aligned} \tag{3.11}$$

or equivalently $\forall \mathbf{q} \in \mathbf{Q}_h$, we have

$$\begin{aligned} &\mathrm{div}_h \mathbf{q} \in V_{0h}, \\ &\int_{\Omega_j} (\mathrm{div}_h \mathbf{q}) w_j \, dx = -\int_{\Omega_j} \mathbf{q} \cdot \nabla w_j \, dx \quad \forall j = 1, \dots, N_{0h}. \end{aligned} \tag{3.12}$$

Since the functions $\mathbf{q}$ and ∇w_j are constant over the triangles of $\mathcal{T}_h$, the integrals on the right-hand sides of the equations in (3.12) can be computed exactly (and easily). On the other hand, to simplify the computation of the integrals on the left-hand sides, we advocate using the trapezoidal rule to compute (approximately this time) these integrals; we then obtain

$$(\mathrm{div}_h\mathbf{q})(P_j) = -\frac{3}{|\Omega_j|}\int_{\Omega_j}\mathbf{q}\cdot\nabla w_j\,dx \quad \forall j = 1,\ldots,N_{0h}. \tag{3.13}$$

Remark 5. Albeit satisfactory conceptually, the use of the discrete Green's formulas (as done above to approximate the divergence operator) may lead to spurious oscillations (see [9] for dramatic evidence of this unwanted phenomenon), particularly when combined with low-order approximations, as in the case here. In order to eliminate (or at least strongly dampen) these unwanted oscillations, we advocate the following regularization (some say also stabilization) procedure: replace (3.12) or (3.13) by: $\forall\mathbf{q}\in\mathbf{Q}_h$ we have

$$\begin{aligned} &\mathrm{div}_h\mathbf{q}\in V_{0h},\\ &C\sum_{T\in\mathcal{T}_h}\int_T |T|\nabla(\mathrm{div}_h\mathbf{q})\cdot\nabla v\,dx + \int_\Omega(\mathrm{div}_h\mathbf{q})\,v\,dx\\ &\qquad = -\int_\Omega \mathbf{q}\cdot\nabla v\,dx \quad \forall v\in V_{0h}, \end{aligned} \tag{3.14}$$

with $C(>0)$; boundary layer thickness considerations suggest $C\approx 1$. The above kind of Tychonov regularization procedure has been successful when applied to the solution of the Dirichlet problem for the Monge–Ampère equation in two dimensions, using mixed finite element approximations based on low-order C^0-conforming finite element spaces; see [9] for further details. Actually, it has not been tested yet for the solution of problem (1.5).

3.3. Discrete Lagrangian and discretized subproblems. Since our goal in this paper is to compute and approximate the solution of problem (1.5), using a discrete variant of Algorithm 1, a first step in that direction is to define an approximation of the augmented Lagrangian (2.8). The candidate functional $\mathcal{L}_{\mathbf{r}h} : (V_h\times\mathbf{Q}_h^3\times V_{0h})\times(\mathbf{Q}_h^2\times V_{0h})\to\mathbb{R}$ proposed in this paper is the following:

$$\begin{aligned}
\mathcal{L}_{\mathbf{r}h}(v,\mathbf{q}_1,\mathbf{q}_2,\mathbf{q}_3,\varphi;\boldsymbol{\mu}_1,\boldsymbol{\mu}_2,\mu_3) = {} & \frac{\varepsilon}{3}\sum_{j=1}^{N_{0h}}|\Omega_j||\varphi(P_j)| \\
& + \frac{1}{3s}\sum_{j=1}^{N_h}|\Omega_j||f(P_j)-v(P_j)|^s \\
& + \frac{r_1}{2}\int_\Omega|\nabla v-\mathbf{q}_1|^2dx + \int_\Omega\boldsymbol{\mu}_1\cdot(\nabla v-\mathbf{q}_1)dx \\
& + \frac{r_2}{2}\int_\Omega|\mathbf{q}_2-\mathbf{q}_3|^2dx + \int_\Omega\boldsymbol{\mu}_2\cdot(\mathbf{q}_2-\mathbf{q}_3)dx \\
& + \frac{r_3}{6}\sum_{j=1}^{N_{0h}}|\Omega_j||(\mathrm{div}_h\mathbf{q}_3)(P_j)-\varphi(P_j)|^2 \\
& + \frac{1}{3}\sum_{j=1}^{N_{0h}}|\Omega_j|\mu_3(P_j)\left[(\mathrm{div}_h\mathbf{q}_3)(P_j)-\varphi(P_j)\right],
\end{aligned} \tag{3.15}$$

where $(\mathbf{q}_1,\mathbf{q}_2)\in\mathbf{E}_{12h}$ with

$$\mathbf{E}_{12h} = \left\{(\mathbf{q}_1,\mathbf{q}_2)\in\mathbf{Q}_h^2 : \mathbf{q}_2 = \frac{\mathbf{q}_1}{\sqrt{1+|\mathbf{q}_1|^2}}\right\}. \tag{3.16}$$

Based on $\mathcal{L}_{\mathbf{r}h}$, subproblem (2.12) can be approximated by

$$\begin{aligned}
(\mathbf{p}_1^{n+1},\mathbf{p}_2^{n+1}) = \arg\min\nolimits_{(\mathbf{q}_1,\mathbf{q}_2)\in\mathbf{E}_{12h}} & \left[\frac{1}{2}\int_\Omega\left(r_1|\mathbf{q}_1|^2+r_2|\mathbf{q}_2|^2\right)dx\right. \\
& \left. -\int_\Omega(r_1\nabla u^n+\boldsymbol{\lambda}_1^n)\cdot\mathbf{q}_1dx - \int_\Omega(r_2\mathbf{p}_3^n-\boldsymbol{\lambda}_2^n)\cdot\mathbf{q}_2dx\right].
\end{aligned} \tag{3.17}$$

Since functional (3.17) does not contain derivatives of $\mathbf{q}_1$ and $\mathbf{q}_2$, its minimization can be performed pointwise (in practice on the triangles of the finite element triangulation $\mathcal{T}_h$). This leads to the solution, a.e. in Ω, of a four-dimensional problem of the following type:

$$(\mathbf{x}_1,\mathbf{x}_2) = \arg\min\nolimits_{(\mathbf{y}_1,\mathbf{y}_2)\in\mathbf{e}_{12}}\left[\frac{1}{2}\left(r_1|\mathbf{y}_1|^2+r_2|\mathbf{y}_2|^2\right)-\mathbf{b}_1\cdot\mathbf{y}_1-\mathbf{b}_2\cdot\mathbf{y}_2\right], \tag{3.18}$$

with $\mathbf{b}_1,\mathbf{b}_2\in\mathbb{R}^2$ and

$$\mathbf{e}_{12} = \left\{(\mathbf{y}_1,\mathbf{y}_2)\in\mathbb{R}^2\times\mathbb{R}^2 : \mathbf{y}_2 = \frac{\mathbf{y}_1}{\sqrt{1+|\mathbf{y}_1|^2}}\right\}. \tag{3.19}$$

The term $\mathbf{y}_2$ from (3.18) can be easily eliminated using the nonlinear relation in (3.19). This leads to the following unconstrained (and nonconvex) two-dimensional problem:

$$\mathbf{x}_1 = \arg\min\nolimits_{\mathbf{y}\in\mathbb{R}^2}\left[\frac{|\mathbf{y}|^2}{2}\left(r_1+\frac{r_2}{1+|\mathbf{y}|^2}\right)-\left(\mathbf{b}_1+\frac{\mathbf{b}_2}{\sqrt{1+|\mathbf{y}|^2}}\right)\cdot\mathbf{y}\right]. \tag{3.20}$$

Since the objective function in (3.20) is differentiable, an obvious choice for the solution of the problem is Newton's method.

Problem (3.20) can be reduced even further by observing that if $\mathbf{x}$ is a solution of (3.20), then it follows from the Schwarz inequality in $\mathbb{R}^2$ that $\mathbf{x}$ and the vector $\mathbf{b}_1 + \frac{\mathbf{b}_2}{\sqrt{1+|\mathbf{x}|^2}}$ are positively co-linear, that is, there is $\alpha \geq 0$ such that

$$\mathbf{x}_1 = \alpha\left(\mathbf{b}_1 + \frac{\mathbf{b}_2}{\sqrt{1+|\mathbf{x}_1|^2}}\right). \tag{3.21}$$

It follows from (3.20) and (3.21) that

$$\begin{aligned} \mathbf{x}_1 &= \alpha\left(\mathbf{b}_1 + \frac{\mathbf{b}_2}{\sqrt{1+\rho^2}}\right), \\ (\rho,\alpha) &= \arg\ \min_{(\sigma,\tau)\in A}\left[\frac{\tau^2}{2}\left|\mathbf{b}_1 + \frac{\mathbf{b}_2}{\sqrt{1+\sigma^2}}\right|^2\left(r_1 + \frac{r_2}{1+\sigma^2}\right)\right. \\ &\qquad \left. -\ \tau\left|\mathbf{b}_1 + \frac{\mathbf{b}_2}{\sqrt{1+\sigma^2}}\right|^2\right], \end{aligned} \tag{3.22}$$

where

$$A = \left\{(\sigma,\tau) \in \mathbb{R}^+ \times \mathbb{R}^+ : \tau\left|\mathbf{b}_1 + \frac{\mathbf{b}_2}{\sqrt{1+\sigma^2}}\right| = \sigma\right\}. \tag{3.23}$$

Now clearly, due to (3.23), we have

$$\alpha = \frac{\rho}{\left|\mathbf{b}_1 + \frac{\mathbf{b}_2}{\sqrt{1+\rho^2}}\right|}, \tag{3.24}$$

and thus

$$\rho = \arg\ \min_{\sigma\in\mathbb{R}^+}\left[\frac{\sigma^2}{2}\left(r_1 + \frac{r_2}{1+\sigma^2}\right) - \sigma\left|\mathbf{b}_1 + \frac{\mathbf{b}_2}{\sqrt{1+\sigma^2}}\right|\right]. \tag{3.25}$$

Similarly, subproblem (2.17) can be approximated by

$$\begin{aligned} &\mathbf{p}_3^{n+1} \in \mathbf{Q}_h, \\ &r_2\int_\Omega \mathbf{p}_3^{n+1}\cdot\mathbf{q}\,dx + \frac{r_3}{3}\sum_{j=1}^{N_{0h}}|\Omega_j|(\mathrm{div}_h\mathbf{p}_3^{n+1})(P_j)(\mathrm{div}_h\mathbf{q})(P_j) \\ &= \int_\Omega (r_2\mathbf{p}_2^{n+1} + \boldsymbol{\lambda}_2^n)\cdot\mathbf{q}\,dx + \frac{1}{3}\sum_{j=1}^{N_{0h}}|\Omega_j|(r_3\psi^n - \lambda_3^n)(P_j)(\mathrm{div}_h\mathbf{q})(P_j) \\ &\forall\mathbf{q} \in \mathbf{Q}_h. \end{aligned} \tag{3.26}$$

The bilinear functional on the left-hand side of (3.26) being positive definite and symmetric, an obvious choice for the solution of the above problem is the conjugate gradient algorithm, initialized with the vector-valued function $\mathbf{p}_3^n$.

Subproblem (2.18) can be approximated by

$$\begin{aligned}\psi = \arg\min_{\varphi \in V_{0h}} \Bigg[\varepsilon \sum_{j=1}^{N_{0h}} |\Omega_j||\varphi(P_j)| + \frac{r_3}{2} \sum_{j=1}^{N_{0h}} |\Omega_j||\varphi(P_j)|^2 \\ - \sum_{j=1}^{N_{0h}} |\Omega_j|(r_3 \mathrm{div}_h \mathbf{p}_3^{n+1} + \lambda_3^n)(P_j)\varphi(P_j)\Bigg].\end{aligned} \tag{3.27}$$

Let $X_j^n = (r_3 \mathrm{div}_h \mathbf{p}_3^{n+1} + \lambda_3^n)(P_j)$. The closed form solution of subproblem (3.27) is given by

$$\psi^{n+1}(P_j) = \frac{1}{r_3} \operatorname{sgn}(X_j^n) \max(0, \left|X_j^n\right| - \varepsilon) \quad \forall j = 1, \ldots, N_{0h}. \tag{3.28}$$

Finally, subproblem (2.19) can be approximated by

$$\begin{aligned} & u^{n+1} \in V_h, \\ & r_1 \int_\Omega \nabla u^{n+1} \cdot \nabla v \, dx + \frac{1}{3} \sum_{j=1}^{N_h} |\Omega_j||(u^{n+1} - f)(P_j)|^{s-2}(u^{n+1} - f)(P_j)v(P_j) \\ & \qquad = \int_\Omega (r_1 \mathbf{p}_1^{n+1} - \boldsymbol{\lambda}_1^n) \cdot \nabla v \, dx \quad \forall v \in V_h.\end{aligned} \tag{3.29}$$

Problem (3.29) could be solved, for $1 \leq s < 2$, for example, using a semismooth Newton method or a nonlinear overrelaxation method like that discussed in [26]. Alternately, if $s = 2$, problem (3.29) reduces to

$$\begin{aligned} & u^{n+1} \in V_h, \\ & r_1 \int_\Omega \nabla u^{n+1} \cdot \nabla v \, dx + \frac{1}{3} \sum_{j=1}^{N_h} |\Omega_j|(u^{n+1} - f)(P_j)v(P_j) \\ & \qquad = \int_\Omega (r_1 \mathbf{p}_1^{n+1} - \boldsymbol{\lambda}_1^n) \cdot \nabla v \, dx \quad \forall v \in V_h.\end{aligned} \tag{3.30}$$

It that case, a wide variety of methods could be applied. This article advocates a method called radix-4 partial solution variant of the cyclic reduction (PSCR) [36, 37, 43, 47] in the cases where the discretization mesh is orthogonal. In other cases, subproblem (3.30) could be solved, for example, using multigrid-type methods.

4. Implementation.

Subproblems excluding the first one can be addressed in a straightforward fashion. Subproblem (3.26) is solved using the conjugate gradient method without any preconditioning since we use a uniform mesh in practice. The subproblem (3.30) is solved using the radix-4 PSCR method. Finally, the solution of subproblem (3.27) is a simple trianglewise operation.

Before a digital image $\hat{v} : \{1, \dots, \hat{W}\} \times \{1, \dots, \hat{H}\} \to \mathbb{R}$ can be presented as a member of the finite element space V_h, we must first choose how we are going to deal with the dimensions of Ω and the spatial discretization step h. When one of these is chosen, the other is also fixed. We decided to normalize the dimensions of $\Omega = (0, W) \times (0, H)$ by setting $\max(W, H) = 1$. As a result, the spatial discretization step h is fixed to $1/(\max(\hat{W}, \hat{H}) - 1)$, implying $h \ll 1$ in practice. The fact that the spatial discretization step depends on the pixel size of the image could potentially have an effect on the final denoised image. See Remark 7 for further discussion.

4.1. Solution of the first subproblem. Apparently the most involved part of the discrete analogue of Algorithm 1 is the solution of the first subproblem (3.17). We could solve either the two-dimensional form (3.20) or the one-dimensional form (3.25). Depending on $\mathbf{b}_1, \mathbf{b}_2$ and r_1, r_2 both forms can have multiple local minimas due to the nonconvex nature of the mean curvature. Our actual realization first applies Newton's method to the two-dimensional form (3.20) starting from obvious initial guess $\mathbf{p}_1^n(x)$. Assuming that the method converges and the achieved solution is actually a local minimum, we then use the explicit relation (3.21) to test the obtained solution candidate. Only then is the solution candidate accepted.

If Newton's method fails, we proceed with the one-dimensional form (3.25) and apply the well-known bisection method. Some fine-tuning is needed because the best local mimima should be obtained to guarantee overall convergence. Hence, our actual heuristic algorithm for the minimization of the one-dimensional form (3.25) reads as follows:

Algorithm 2.

```
$\eta = \lceil \log_4(\sqrt{2}h^{-1}) \rceil$
$L = \{[4^{k-1}, 4^k] : k = 1, \dots, \eta\} \cup \{\{0\}, [0, 1], [4^\eta, +\infty[\}$
$K = \left\{ bisection(\Psi, l, \frac{1}{10}) \in \mathbb{R} : l \in L \right\}$
$\bar{k} = arg\ min_{k \in K}\ \Psi(k)$
$\rho = bisection(\Psi, [\bar{k} - \frac{1}{10}, \bar{k} + \frac{1}{10}], \sigma)$
$\mathbf{p}_{bisection}(x) = \frac{\rho}{\left| \mathbf{b}_1 + \frac{\mathbf{b}_2}{\sqrt{1+\rho^2}} \right|} \left( \mathbf{b}_1 + \frac{\mathbf{b}_2}{\sqrt{1 + \rho^2}} \right)$
```

Above, Ψ is the objective function from the one-dimensional form (3.25), and the function $bisection(\Psi, l, \varsigma)$ applies the bisection search to the function Ψ on the interval l with accuracy tolerance ς. If interval l is unbounded, then interval l is adjusted by moving the right-hand side boundary in such a way that the value of the function Ψ is larger on the right-hand side boundary than it is on the left-hand side boundary. In addition, h is the spatial discretization step and σ (> 0) is the requested accuracy. Finally, we set

$$\mathbf{p}_1^{n+1}(x) = \arg\min_{\mathbf{q} \in \{\mathbf{p}_{\text{newton}}, \mathbf{p}_{\text{bisection}}, \mathbf{p}_1^n(x)\}} \Phi(\mathbf{q}), \tag{4.1}$$

where Φ is the objective function from (3.20) and $\mathbf{p}_{\text{newton}}$ is the solution (or the result of the last iteration) obtained by Newton's method.

We obtained this heuristic algorithm by observing the cases where Newton's method failed to solve the two-dimensional form (3.20) and by investigating the behavior of the one-dimensional form (3.25) on those cases. Since $\mathbf{p}_1^n$ begins to approximate the gradient of u^n as the discrete analogue of Algorithm 1 progresses and the solution of the one-dimensional form (3.25) is the length of $\mathbf{p}_1^n$, it makes sense to concentrate on the interval $[0, \sqrt{2}h^{-1}]$ (assuming $\mathrm{Im}(u^n) \subset [0,1]$). The overall shape of the graph motivated us to divide the interval into subintervals as described above. In rare cases, the global mimima was actually located outside the range $[0, \sqrt{2}h^{-1}]$, which also had to be taken into account.

In practice, the combination of the two aforementioned algorithms works well in almost all cases. Newton's method converges quickly for most of the triangles of $\mathcal{T}_h$; for the few triangles for which Newton's method fails, one uses Algorithm 2. The cases where both algorithms fail are rare and are limited to individual triangles.

4.2. Initialization and stopping criterion. Concerning the use of the discrete variant of Algorithm 1, several issues will be addressed, obvious ones being initialization, stopping criterion, and the choice of the Lagrange multipliers $\mathbf{r}$ and ε. The choice of parameters $\mathbf{r}$ and ε is addressed in section 5, and thus only the initialization and stopping criterion will be discussed here.

A number of different kinds of initialization methods were considered; the most prominent were

$$\begin{aligned} u^0 = f,\ \mathbf{q}_1^0 = \nabla u^0,\ \mathbf{q}_2^0 = \frac{\mathbf{q}_1^0}{\sqrt{1+|\mathbf{q}_1^0|^2}},\ \mathbf{q}_3^0 = \mathbf{q}_2^0,\ \psi^0 = \nabla\cdot \mathbf{q}_3^0, \\ \boldsymbol{\lambda}_1^0 = \boldsymbol{\lambda}_2^0 = 0,\ \lambda_3^0 = 0, \end{aligned} \tag{4.2}$$

and

$$\begin{aligned} u^0 = 0,\ \mathbf{q}_1^0 = \mathbf{q}_2^0 = \mathbf{q}_3^0 = 0,\ \psi^0 = 0, \\ \boldsymbol{\lambda}_1^0 = \boldsymbol{\lambda}_2^0 = 0,\ \lambda_3^0 = 0. \end{aligned} \tag{4.3}$$

In the case of initialization (4.2), only the term corresponding to the mean curvature of the image surface is nonzero in the functional $\mathcal{L}_{\mathbf{r}h}$; and, in the case of initialization (4.3), only the term $|u-f|^s$ is nonzero. These differences play a major role in the overall behavior of the algorithm; the effect of both initializations will be discussed in further detail in section 5.

The second issue to be addressed is the stopping criterion. Several candidates were considered, such as

$$\frac{|u^n - u^{n+1}|_\infty}{|u^n|_\infty} < \delta,\ \frac{|u^n - u^{n+1}|_2}{|u^n|_2} < \delta,\ \text{and}\ \frac{|\mathcal{J}(u^n) - \mathcal{J}(u^{n+1})|}{|\mathcal{J}(u^n)|} < \delta, \tag{4.4}$$

where $\delta > 0$. The following criterion was found to be the most straightforward:

$$\frac{|\mathcal{L}_{\mathbf{r}h}(\omega^n) - \mathcal{L}_{\mathbf{r}h}(\omega^{n+1})|}{|\mathcal{L}_{\mathbf{r}h}(\omega^n)|} < \delta, \tag{4.5}$$

where $\omega^n = (u^n, \mathbf{p}_1^n, \mathbf{p}_2^n, \mathbf{p}_3^n, \psi^n; \boldsymbol{\lambda}_1^n, \boldsymbol{\lambda}_2^n, \lambda_3^n)$.

The aforementioned criterion works well as long as the Lagrange multipliers r_1, r_2, and r_3 are selected to be large enough so that they accurately enforce the equality constraints in (2.4).

5. Numerical results.

In order to demonstrate the functionality of the discrete variant of Algorithm 1, we applied it against a large variety of test problems. These problems include synthetic and photographic images. For all these test problems, the values of the noise function g are uniformly distributed on the closed interval $[-p, p]$, $p > 0$. We took $\delta = 10^{-4}$ for the stopping criterion defined by (4.5). All images are grayscale, and the original images are scaled to the range $[0, 1]$.

5.1. Choice of initialization method and parameters.

The behavior of the algorithm varied drastically depending on its initialization. Initialization (4.2) had a tendency to cause extremely slow convergence and an imperceptible low decrease of the value of the objective function in (1.5). By adjusting the parameters ε and $\mathbf{r}$ accordingly, reasonably good results were obtained in some cases. Usually this required a large parameter ε and small values for the Lagrange multipliers r_1, r_2, and r_3. However, each Lagrange multiplier combination worked only for a specific problem, and the undertaking of finding a Lagrange multiplier combination applicable to all problems proved futile. When the method was successfully initialized this way, it retained a considerable amount of detail while leaving some residual noise.

On the other hand, the initialization (4.3) caused a completely different behavior as the convergence was much faster, particularly during the first few tens of iterations. Finding a suitable Lagrange multiplier combination was difficult. The equality constraints in (2.4) and a crude dimensional analysis suggest that if $|\nabla u|^2 \ll 1$, the augmentation functionals behave such that $\mathbf{p}_1 \sim h^{-1}, \mathbf{p}_2 \sim h^{-1}, \mathbf{p}_3 \sim h^{-1}$, and $\psi \sim h^{-2}$. Thus, taking into account the homogeneity considerations, we should choose the following: $\varepsilon \sim h^2, r_1 \sim h^2, r_2 \sim h^2$, and $r_3 \sim h^4$. On the other hand, the same analysis suggests that for $|\nabla u|^2 \gg 1$, the augmentation functionals behave such that $\mathbf{p}_1 \sim h^{-1}, \mathbf{p}_2 \sim 1, \mathbf{p}_3 \sim 1$, and $\psi \sim h^{-1}$, and thus we should choose the following: $\varepsilon \sim h, r_1 \sim h^2, r_2 \sim 1$, and $r_3 \sim h^2$. In addition, from the formulation of subproblem (3.26), it can be seen that $h^2 r_2 \approx r_3$ could be a suitable choice because it would balance the left-hand side terms. Otherwise the conjugate gradient method would converge extremely slowly.

However, by applying the discrete analogue of Algorithm 1 against a large number of test problems and observing the residuals associated with the equality constraints in (2.4), we concluded that we should choose $\varepsilon \sim h$, $r_1 \sim h, r_2 \sim 1$, and $r_3 \sim h^2$. This was due to the fact that the equality constraint associated with the Lagrange multiplier r_1 did not converge when $r_1 \sim h^2$. Further testing leads us to the following Lagrange multiplier combination:

$$
\begin{aligned}
\varepsilon &= r_0 h, \\
r_1 &= 10 r_0 h, \\
r_2 &= 5 r_0, \\
r_3 &= 5 r_0 h^2,
\end{aligned}
\tag{5.1}
$$

where $r_0 (> 0)$ is a parameter that can be tuned depending on the amount of noise.

In (5.1), the Lagrange multipliers r_1, r_2, and r_3 are large enough to accurately enforce the equality constraints in (2.4) while keeping the convergence speed reasonable. When one combines the initialization (4.3) with the above-mentioned Lagrange multipliers, the method removes a considerable amount of noise while filtering out some detail. From these observations it was decided to use initialization (4.3) and the Lagrange multiplier combination (5.1) for all examples and comparisons.

Remark 6. A "simple" way to fix the problem with the selection of ε is to take

$$\varepsilon = Ch^{1+\frac{1}{1+|\nabla u|^2}}, \tag{5.2}$$

with C on the order of 1 (other nonlinear functions of $|\nabla u|$ are possible). We intend to investigate this approach in another paper. A simple way to use this variable in space parameter ε is as follows:

1. Solve the image restoration problem using a simpler method (based on BV-regularization, for example). Call u_0 the solution to this problem.
2. Set $\varepsilon_0 = Ch^{1+\frac{1}{1+|\nabla u_0|^2}}$.
3. For $i = 1, \ldots, M$, solve

$$u_i = \arg\min_{v\in V} \int_\Omega \varepsilon_{i-1} \left| \nabla\cdot \frac{\nabla v}{\sqrt{1+|\nabla v|^2}} \right| dx + \frac{1}{s}\int_\Omega |f-v|^s\, dx \tag{5.3}$$

and set $\varepsilon_i = Ch^{1+\frac{1}{1+|\nabla u_i|^2}}$. We believe that $M = 2$ should be enough.

Remark 7. In our approach, $\Omega = (0, W) \times (0, H)$ is normalized such that $\max(W, H) = 1$. This means that the spatial discretization step h depends on the pixel dimensions of the image. However, this issue was treated very differently in [51] and [52]. In these papers, the spatial discretization step h was chosen first and, thus, in contrast to our approach, the dimensions of Ω depend on the pixel dimensions of the image.

As pointed out in [51] and [52], the spatial discretization step h plays an important role in the behavior of the method. This is due to the fact that h affects the magnitude of the gradient. In order to justify the choices made in this paper we investigated the behavior of the discrete regularization term

$$k(v)(P_j) = \left| \left(\mathrm{div}_h \frac{\nabla v}{\sqrt{1+|\nabla v|^2}} \right)(P_j) \right|, \; v \in V_h, P_j \in \Sigma_{0h}. \tag{5.4}$$

Our goal was to find out how $k(v)$ behaves pointwise as a function of h. This refines the view on how h should be chosen. We generated a large number of test images containing only random noise and analyzed the obtained data statistically. In addition, we modified our implementation so that the parameter h we consider in the remainder of the current remark is consistent with the one used in [51], [52], implying that h may be chosen freely, and tested various values of this parameter (including some larger than 1).

Figures 2, 3, and 4 show the obtained results. In Figure 2, the pixel values are uniformly distributed over various intervals; i.e., the figure shows how the regularization term reacts to the changes in the "intensity" of the noise. In Figure 3, the nonzero pixel values are uniformly distributed in the interval $[-0.2, 0.2]$, but only a certain percentage of the pixels contains nonzero values; i.e., the figure shows how the regularization term reacts to the changes in the "quantity" of the noise. Figure 4 shows the same results in a scaled form.

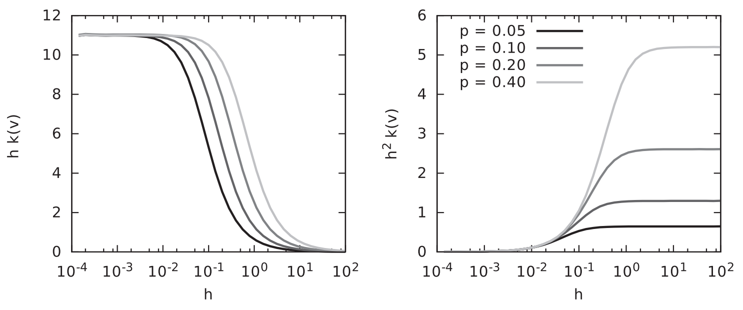


Figure 2. *The average pointwise behavior of the discrete regularization term $k(v)$ as a function of h. The pixel values are uniformly distributed in the interval $[-p, p]$.*

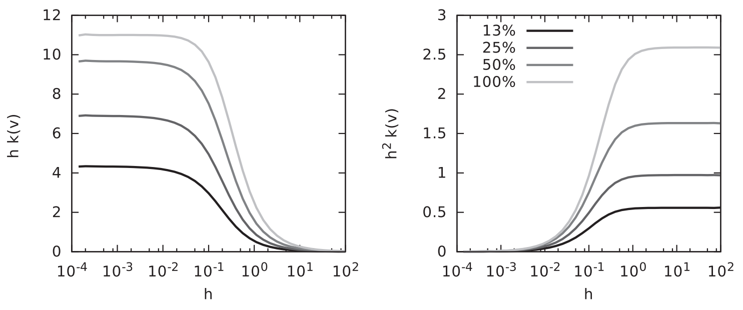


Figure 3. *The average pointwise behavior of the discrete regularization term $k(v)$ as a function of h. Only a certain percentage of the pixels contains nonzero values.*

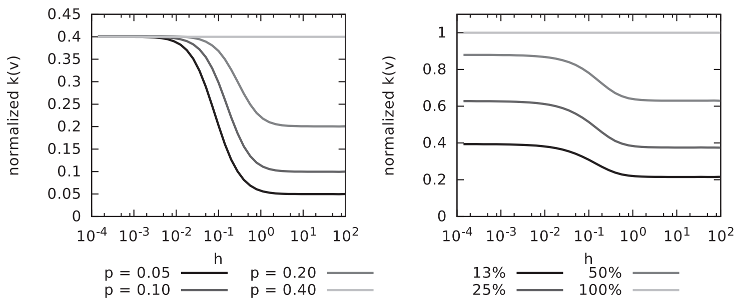


Figure 4. *Scaled versions of Figures 2 (on the left) and 3 (on the right).*

It is clear that $k(v)$ exhibits three different behaviors when the value of h is varied:

1. If h is small enough ($\sim 10^{-3}$), then $k(v) \sim h^{-1}$, which is consistent with the aforementioned dimensional analysis. Another defining property is that the regularization term $k(v)$ is unable to differentiate between low intensity and high intensity noise.

This has three main consequences: First, the regularization term is almost always very large when dealing with photographic images; thus the model wants to produce overly smooth solutions. As a result, the parameter ε must be small to preserve small details. Second, it is likely that the model may misidentify some noise as a true jump in the data and preserve it. This observation was also noted in [51] and [52]. Third, the regularization term probably becomes even more difficult to deal with because the transition from a smooth image to a noisy image is extremely steep. This could also explain why the initialization (4.2) did not work: If $u^0 = f$, then the value of the regularization term would be the same almost everywhere at u^0, and a transition to even slightly lower "energy" would require significant smoothing of the image. In other words, it is nearly impossible to find a descending path from u^0 to the global minimizer because the regularization term is "flat" near u^0 $(= f)$. It should be noted that the regularization term can still differentiate between low and high quantity of noise.

2. If h is large enough ($\sim 10^1$), then $k(v) \sim h^{-2}$, which is again consistent with the aforementioned dimensional analysis. The results show that the value of the regularization term is directly proportional to the intensity of the noise (see, in particular, Figure 4). This is not particularly surprising because $k(v) \sim \triangle v$ when $|\nabla v|^2 \ll 1$. Numerical experiments indicate that our method works really well when $h = 1$, but the output images are quite blurred, as expected, since the regularization term favors shallow gradients over the steeper ones. In addition, the initialization (4.2) actually works even better than the initialization (4.3) in the sense that the method converges much faster. Both initializations also lead to the same solution in most cases. Again, this is not a surprise because the nonconvex part of the regularization term does not have a major impact when $|\nabla v|^2 \ll 1$.
3. If $h \sim 10^{-1}$, then the regularization term does not have a clear asymptotic behavior. This parameter range is clearly the most interesting because of the way the regularization term reacts to the changes in the intensity of the noise. The value of h determines how steep the transition from a smooth image to a noisy image is, and, thus, it has a significant effect on the final output image. Unfortunately, it is far from clear how h should be chosen. In principle, the parameter h determines how the regularization term reacts to noise, and the parameter ε determines how strongly this reaction is taken into account. However, in practice, there seems to be some overlap between these two parameters.

Our choice of h falls into the first category if it is assumed that the number of pixels is large. Although this choice has many disadvantages, the effects of which can be seen in some of the numerical results presented in this paper, we believe that our choice is justified, at least in the context of this paper, for the following reasons: (i) If the input image contains substantial amount of noise, then only a limited amount of information can be recovered even under the best conditions. By taking this into consideration, a solution that is a little too smooth is not a big disadvantage. In addition, if the original image is smooth, then small h is a reasonable choice. (ii) If $h \ll 1$, then the two terms in the objective function can be easily balanced by selecting $\varepsilon \sim h$. This means that tuning the parameters is going to be a much easier task. Considering that the goals of this paper are purely algorithmic, we do not want to focus too much on such tuning. (iii) Since the objective function is likely to be more challenging to deal with when h is chosen to be small, problems with small h can be seen as benchmark problems. In that sense, small h is is well suited for the goals of this paper.

5.2. Examples and comparisons. Figure 5 shows the results obtained while applying the implementation against one of the synthetic test images (Test9). This synthetic test image contains only simple patterns and shapes. The purpose of this test image is to show that the algorithm works effectively in the sense that edges, corners, and image contrast are preserved. Here the noise parameter p was 0.2, and the parameter r_0 was 0.015. The value of the objective function in (1.5) at noisy image f was 0.027159 and $\mathcal{J}(u^0) = 0.364139$. The convergence to the given tolerance was achieved after 291 iterations with $\mathcal{J}(u^{291}) = 0.007054$. The l^2-residual between the vector representations of the original and noisy images was 58.95. A similar residual between the original and output images was 9.049. The algorithm was able to eliminate practically all noise, and the output image is almost indistinguishable from the original image. However, when the difference between the output and noisy images is examined more closely, it is clear that the algorithm had some minor difficulties with the diagonal tips of the star.

Figure 6 shows similar results for a second synthetic test image (Test6). This synthetic test image contains many different challenges (sine waves, high and low contrast text, gradients, edges, corners, and narrow bounces) to the algorithm. Here $p = 0.2$, $r_0 = 0.004$, $\mathcal{J}(f) = 0.004503$, and $\mathcal{J}(u^0) = 0.207021$. The convergence to the given tolerance was achieved after 215 iterations with $\mathcal{J}(u^{215}) = 0.004130$. The residuals between the noisy and original and the output and original images were 58.16 and 14.35, respectively. The algorithm filtered out a considerable amount of noise, and it is very clear that the algorithm does not have problems with sine waves or gradients. However, the algorithm had difficulties in preserving certain details, such as low-contrast text and the narrowest bounces in the surface.

Figure 7 shows results for a photographic image (Barbara). This test image is particularly challenging because it contains image details and noise on similar scales. Here, $p = 0.2$, $r_0 = 0.005$, $\mathcal{J}(f) = 0.008823$, and $\mathcal{J}(u^0) = 0.134979$. After 268 iterations, the value of the objective function in (1.5) was 0.008394. The residuals between the noisy and original and the output and original images were 58.86 and 35.50, respectively. Again, the algorithm filtered out a considerable amount of noise, but some details were lost in the process as it is clear that the algorithm was unable to resolve the stripes and grids in the table cloth and pants. This can be seen clearly in the intersection plot.

The values of the objective function along the iterations are plotted in Figures 8, 9, and 10. The figures also include the following normalized residuals:

$$\begin{aligned}
R_1^n &= \frac{1}{2}\frac{|\nabla u^n - \mathbf{p}_1^n|}{\max(|\nabla u^n|, |\mathbf{p}_1^n|)},\\
R_2^n &= \frac{1}{2}\frac{|\mathbf{p}_2^n - \mathbf{p}_3^n|}{\max(|\mathbf{p}_2^n|, |\mathbf{p}_3^n|)},\\
R_3^n &= \frac{1}{2}\frac{|\mathrm{div}_h \mathbf{p}_3^n - \psi^n|}{\max(|\mathrm{div}_h \mathbf{p}_3^n|, |\psi^n|)}.
\end{aligned} \tag{5.5}$$

In all three cases, the value of the objective function drops sharply during the first few tens of

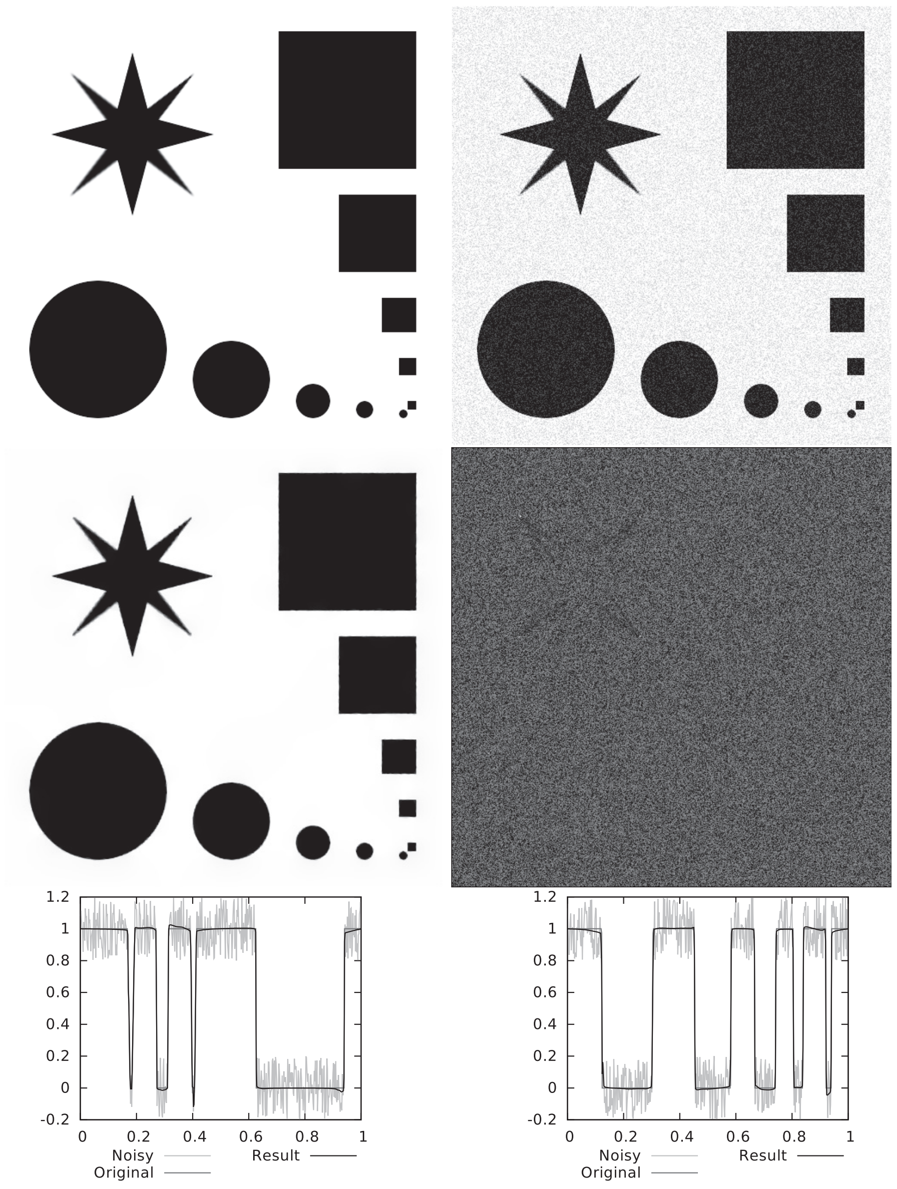


Figure 5. *A synthetic test image (Test9) containing simple shapes. From top left: Original image, noisy image ($p = 0.2$), output image, difference between the output and noisy images, and two horizontal intersections.*

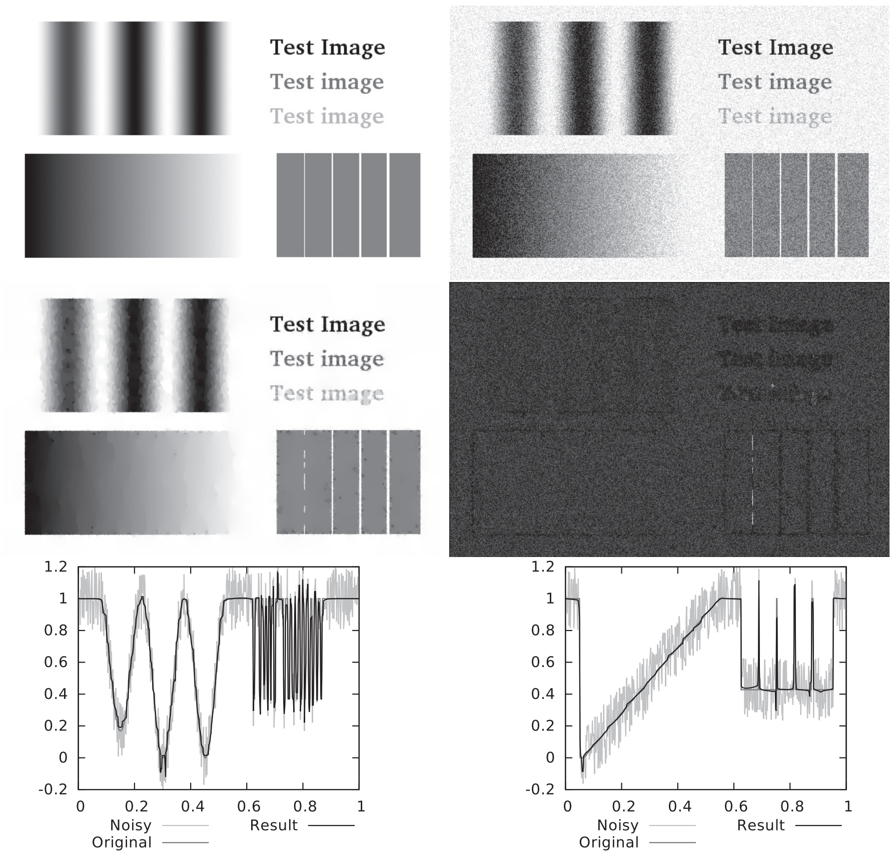


Figure 6. *A synthetic test image (Test6) containing sine waves, text, gradients, and narrow bounces. From top left: Original image, noisy image ($p = 0.2$), output image, difference between the output and noisy images, and two horizontal intersections.*

iterations. However, in the case of the last two, it takes a few tens of iterations more before the value of the objective function drops below $\mathcal{J}(f)$. Also, the decrease is not monotonic as the value of the objective function jumps momentarily after a few iterations. This jump takes place at the same time as the value of the normalized residual R_1^n jumps and was observed with almost all the test images in varying extent. Similar behavior was also presented in [52].

Figure 11 shows a comparison between different test problems with varying amounts of noise. The relevant parameters, objective function values, iteration counts, and residuals are shown in Table 1. It is clear that the algorithm performed commendably when $p = 0.05$ or $p = 0.1$. In the case of $p = 0.2$, more details were lost in the process, and when $p = 0.4$, almost all small details were lost.

Finally, Figure 12 visualizes the influence of r_0 on the resulting denoised image. The

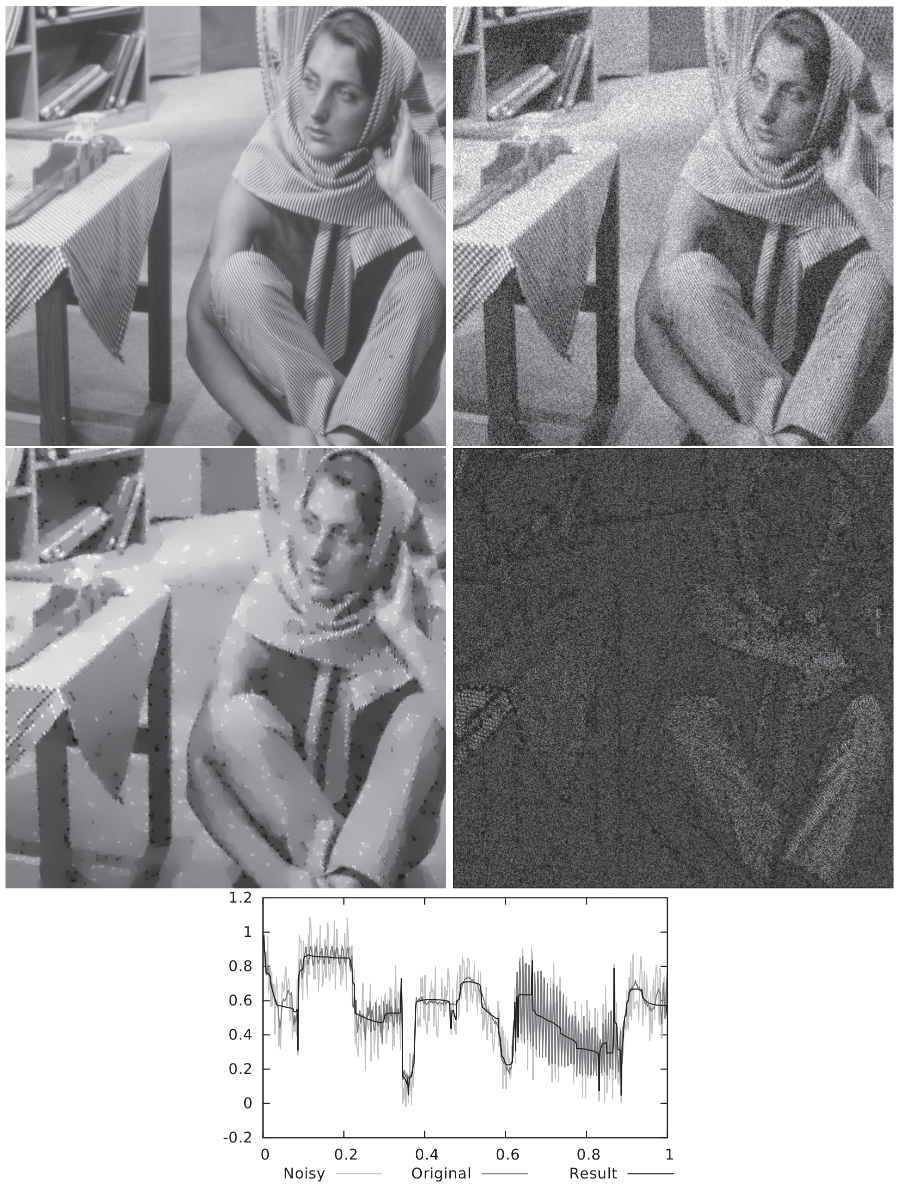


Figure 7. *A photographic test image (Barbara). From top left: Original image, noisy image ($p = 0.2$), output image, difference between the output and noisy images, and horizontal intersection from the center of the image.*

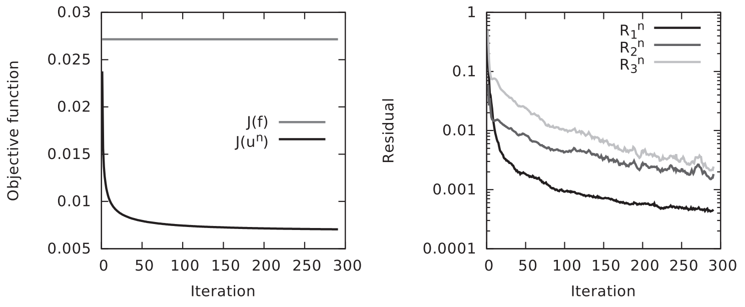


Figure 8. *Values of the objective function in* (1.5) *and normalized residuals* (5.5) *along the iterations for a synthetic test image (Test9).*

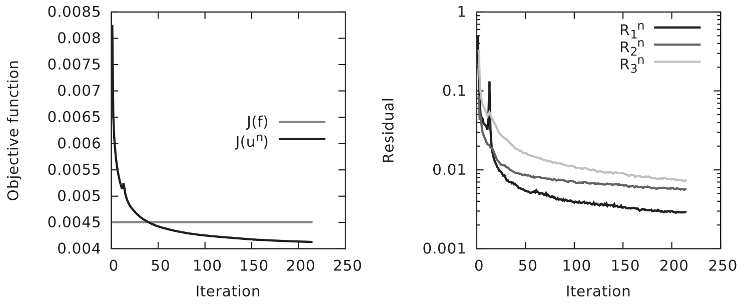


Figure 9. *Values of the objective function in* (1.5) *and normalized residuals* (5.5) *along the iterations for a synthetic test image (Test6).*

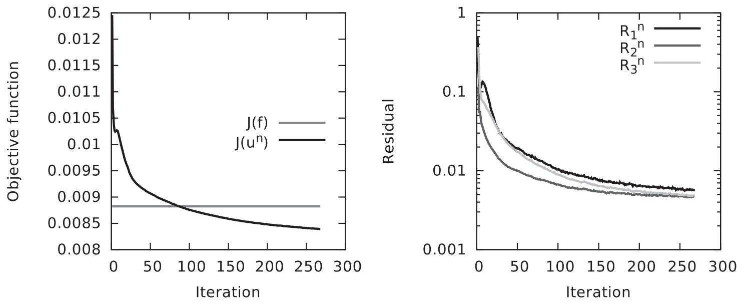


Figure 10. *Values of the objective function in* (1.5) *and normalized residuals* (5.5) *along the iterations for the photographic test image (Barbara).*

relevant objective function values, iteration counts, and residuals are shown in Table 2. In all cases, $p = 0.2$. It is clear that the optimal value of r_0 is somewhere near 0.005. These results illustrate a clear trend that can be observed in all considered test images: the larger the value of r_0 is, the better the algorithm behaves. Actually, when $r_0 = 0.001$, the value of

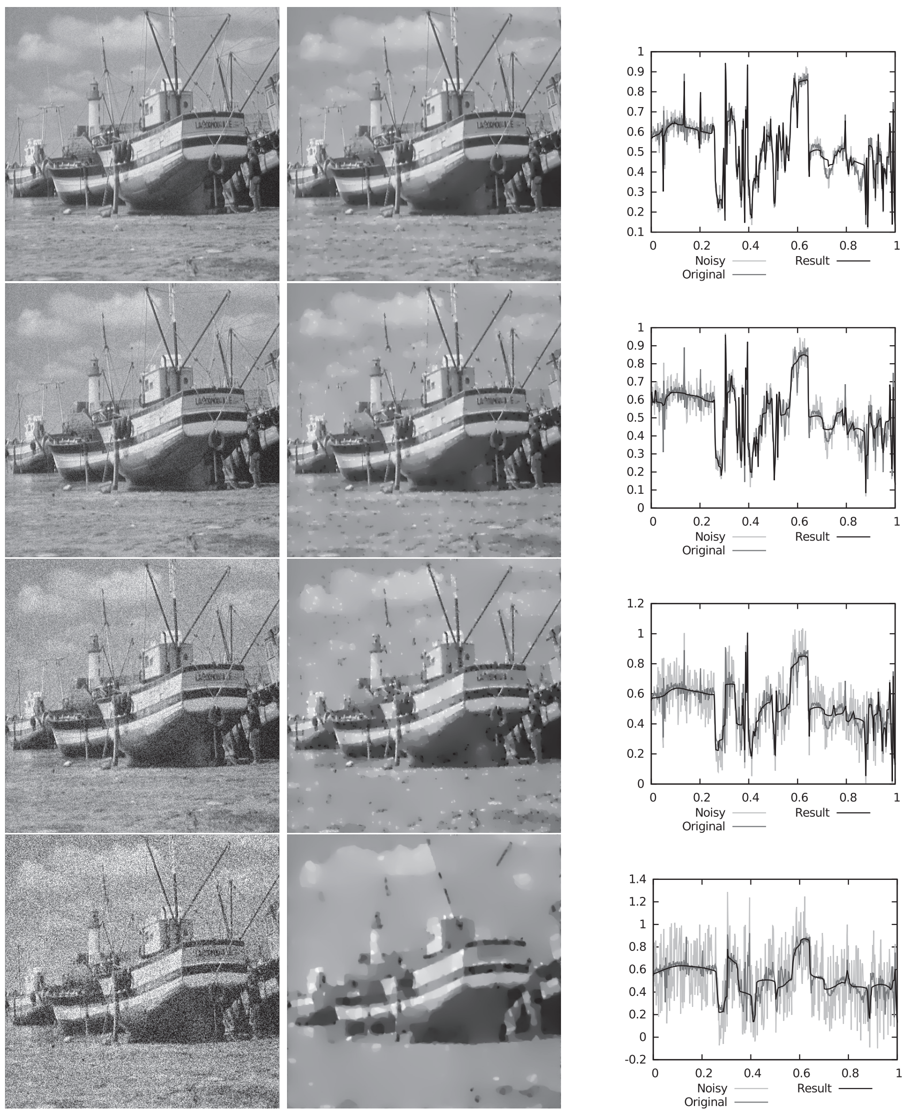


Figure 11. *A comparison between different test problems with varying amounts of noise:* $p = 0.05$ *(*$r_0 = 0.001$*),* $p = 0.1$ *(*$r_0 = 0.002$*),* $p = 0.2$ *(*$r_0 = 0.005$*), and* $p = 0.4$ *(*$r_0 = 0.02$*). From left to right: The noisy image, the output image, and a horizontal intersection from the center of the image.*

the objective function at the achieved solution u^n is higher than it is at the noisy image f, although the algorithm appears to be working properly otherwise. On the other hand, a large value of r_0 means that more detail is lost in the process. Thus, selecting an optimal value for r_0 is a difficult balancing act. Fortunately, it appears that the optimal value of the parameter r_0 mainly depends on the amount of noise and does not depend as strongly on the image itself. The previous sentence holds true at least when success is measured by the l^2-norm. If success is estimated by visual inspection, then it appears that r_0 must be selected image dependently, because even a small change in the value of the parameter r_0 can have a visible impact on the final result. This is consistent with comments in [52]. However, the values provided in Table 2 work fairly well for most images.

All presented numerical results show that the L^1-mean curvature allows both smooth transitions and large jumps without staircasing. In two dimensions the output results do not appear perfect, but the one-dimensional intersections demonstrate desired overall behavior; the qualitative challenges and difficulties are mostly related to the mixtures of image details and noise on similar scales.

Table 1

A comparison between different test problems with varying amounts of noise, with n denoting the number of iterations necessary to achieve convergence and u^n the achieved solution. The Residuals column shows the residuals between the vector presentations of the original and noisy images and the original and output images.

p	r_0	$\mathcal{J}(f)$	$\mathcal{J}(u^0)$	$\mathcal{J}(u^n)$	Iterations	Residuals
0.05	0.001	0.001626	0.145999	0.000831	150	14.72, 13.10
0.1	0.002	0.003435	0.147198	0.002228	173	29.48, 18.53
0.2	0.005	0.008909	0.152132	0.007485	255	58.99, 26.49
0.4	0.020	0.036206	0.172260	0.029445	181	117.8, 37.31

Table 2

A comparison between different values of r_0. The Residuals column shows the residuals between the vector presentations of the original and noisy images and the original and output images. The function u^n is the achieved solution.

r_0	$\mathcal{J}(f)$	$\mathcal{J}(u^0)$	$\mathcal{J}(u^n)$	Iterations	Residuals
0.001	0.001795	0.142086	0.004210	193	58.93, 39.59
0.005	0.008979	0.142142	0.007126	220	58.92, 21.13
0.01	0.017945	0.142136	0.007911	180	58.85, 21.91
0.05	0.089864	0.141968	0.010577	242	58.87, 33.91

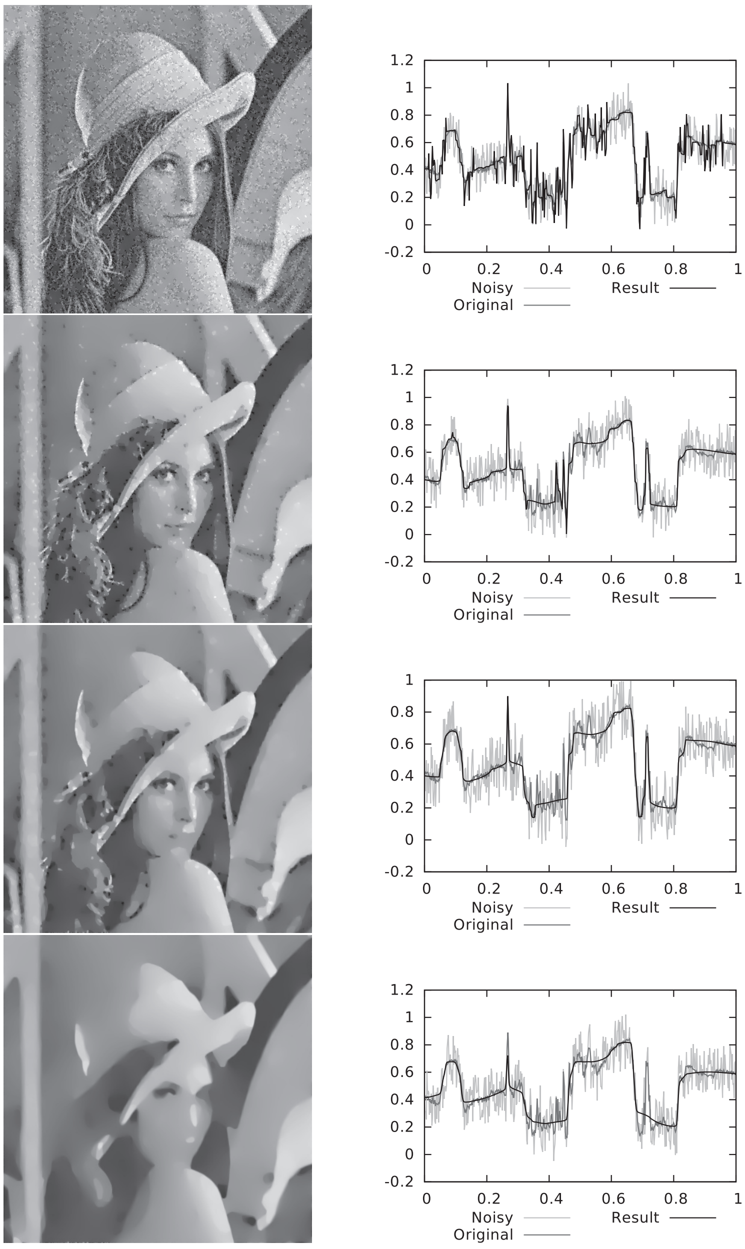


Figure 12. *A comparison between different values of* r_0*:* $0.001, 0.005, 0.01,$ *and* 0.05*. The left side shows the output images, and the right side shows the horizontal intersections from the center of the image.*

6. Conclusions.

This paper presents an image denoising algorithm based on an augmented Lagrangian approach that uses the L^1-mean curvature of the image surface as a regularizer. The main difference between this paper and existing literature (e.g., [52]) is that our methodology relies on a novel augmented Lagrangian functional where the equality constraints treated by augmentation-duality are all linear, resulting in different (and simpler) subproblems. The functionality of the proposed algorithm was demonstrated by applying it against a large set of different types of test problems, some of which were presented in further detail. Based on the numerical experiments, it can be concluded that the algorithm can remove considerable amounts of noise within a reasonable number of iterations. The cpu time used by our implementation is dominated by the solution of the first subproblem; thus we feel that the effort of improving our method should be directed toward this subproblem.

Acknowledgment. The authors thank anonymous reviewers for their valuable feedback. The presentation of the paper was significantly improved thanks to their comments and suggestions.

REFERENCES

[1] L. Ambrosio and S. Masnou, *A direct variational approach to a problem arising in image reconstruction*, Interfaces Free Bound., 5 (2003), pp. 63–81.
[2] L. Ambrosio and S. Masnou, *On a variational problem arising in image reconstruction*, in Free Boundary Problems, Internat. Ser. Numer. Math. 147, Birkhäuser, Basel, 2004, pp. 17–26.
[3] G. Bellettini, V. Caselles, and M. Novaga, *The total variation flow in $\mathbb{R}^N$*, J. Differential Equations, 184 (2002), pp. 475–525.
[4] J. D. Benamou and Y. Brenier, *A computational fluid mechanics solution to the Monge-Kantorovich mass transfer problem*, Numer. Math., 84 (2000), pp. 375–393.
[5] D. P. Bertsekas, *Constrained Optimization and Lagrange Multiplier Methods*, Academic Press, New York, 1982.
[6] D. Boffi, M. Fortin, and F. Brezzi, *Mixed Finite Element Methods and Applications*, Springer Series in Computational Mathematics, Springer, Berlin, Heidelberg, 2013.
[7] S. Boyd, N. Parikh, E. Chu, B. Peleato, and J. Eckstein, *Distributed optimization and statistical learning via the alternating direction method of multipliers*, Found. Trends Mach. Learn., 3 (2011), pp. 1–122.
[8] A. Caboussat, R. Glowinski, and V. Pons, *An augmented Lagrangian approach to the numerical solution of a non-smooth eigenvalue problem*, J. Numer. Math., 17 (2009), pp. 3–26.
[9] A. Caboussat, T. Glowinski, and D.C. Sorensen, *A least-squares method for the numerical solution of the Dirichlet problem for the Monge-Ampére equation in two dimensions*, ESAIM Control Optim. Calc. Var., 19 (2013), pp. 780–810.
[10] A. Chambolle and P. L. Lions, *Image recovery via total variational minimization and related problems*, Numer. Math., 76 (1997), pp. 167–188.
[11] R. Chan, M. Tao, and X. Yuan, *Constrained total variation deblurring models and fast algorithms based on alternating direction method of multipliers*, SIAM J. Imaging Sci., 6 (2013), pp. 680–697.
[12] T. Chan, S. Esedoglu, F. Park, and A. Yip, *Total variation image restoration: Overview and recent developments*, in Handbook of Mathematical Models in Computer Vision, Ni. Paragios, Y. Chen, and O. Faugeras, eds., Springer, New York, 2006, pp. 17–31.
[13] T. F. Chan, S. H. Kang, and J. Shen, *Euler's elastica and curvature-based inpainting*, SIAM J. Appl. Math., 63 (2002), pp. 564–592.
[14] T. F. Chan and J. J. Shen, *Image Processing and Analysis: Variational, PDE, Wavelet, and Stochastic Methods*, SIAM, Philadelphia, 2005.

[15] Y. Chen, W. Hager, F. Huang, D. Phan, X. Ye, and W. Yin, *Fast algorithms for image reconstruction with application to partially parallel MR imaging*, SIAM J. Imaging Sci., 5 (2012), pp. 90–118.

[16] E. J. Dean and R. Glowinski, *Numerical methods for fully nonlinear elliptic equation of the Monge-Ampére type*, Comput. Methods Appl. Mech. Engrg., 195 (2006), pp. 1344–1386.

[17] E. J. Dean, R. Glowinski, and G. Guidoboni, *On the numerical simulation of Bingham visco-plastic flow: Old and new results*, J. Non-Newtonian Fluid Mech., 142 (2007), pp. 36–62.

[18] E. J. Dean and R. Glowinski, *An augmented Lagrangian approach to the numerical solution of the Dirichlet problem for the elliptic Monge-Ampére equation in two dimensions*, Electron. Trans. Numer. Anal., 22 (2006), pp. 71–96.

[19] F. Delbos, J. Ch. Gilbert, R. Glowinski, and D. Sinoquet, *Constrained optimization of seismic reflection tomography: A Gauss-Newton augmented Lagrangian approach*, Geophys. J. Int., 164 (2006), pp. 670–684.

[20] Y. Duan, Y. Wang, X.-C. Tai, and J. Hahn, *A fast augmented Lagrangian method for Euler's elastica model*, in Scale Space and Variational Methods in Computer Vision, Lecture Notes in Comput. Sci. 6667, Springer-Verlag, Berlin, Heidelberg, 2012, pp. 144–156.

[21] X. Feng, R. Glowinski, and M. Neilan, *Recent developments in numerical methods for fully nonlinear second order partial differential equations*, SIAM Rev., 55 (2013), pp. 205–267.

[22] M. Fortin and R. Glowinski, *Augmented Lagrangian Methods: Applications to the Numerical Solution of Boundary Value Problems*, North–Holland, Amsterdam, 1983.

[23] D. Gabay and B. Mercier, *A dual algorithm for the solution of nonlinear variational problems via finite element approximation*, Comput. Math. Appl., 2 (1976), pp. 17–40.

[24] R. Glowinski, *Numerical Methods for Nonlinear Variational Problems*, Springer, New York, 1984.

[25] R. Glowinski, *Finite Element Methods for Imcompressible Viscous Flow*,. in Handbook of Numerical Analysis, vol. 9, North-Holland, Amsterdam, 2003, pp. 3–1176.

[26] R. Glowinski, T. Kärkkäinen, T. Valkonen, and A. Ivannikov, *Nonsmooth SOR for L^1-fitting: Convergence study and discussion of related issues*, J. Sci. Comput., 37 (2008), pp. 103–138.

[27] R. Glowinski and A. Marrocco, *Sur l'approximation par éléments finis d'ordre un, et la résolution par pénalisation-dualité d'une classe de problemes de Dirichlet non linéaires*, RAIRO Anal. Numér., 9 (1975), pp. 41–76.

[28] R. Glowinski and A. Quaini, *On an inequality of C. Sundberg: A computational investigation via nonlinear programming*, J. Optim. Theory Appl., 158 (2013), pp. 739–772.

[29] R. Glowinski and P. Le Tallec, *Augmented Lagrangian and Operator-Splitting Methods in Nonlinear Mechanics*, SIAM, Philadelphia, 1989.

[30] J. Hahn, C. Wu, and X.-C. Tai, *Augmented Lagrangian method for generalized TV-Stokes model*, J. Sci. Comput., 50 (2012), pp. 235–264.

[31] M. Hintermüller, C. N. Rautenberg, and J. Hahn, *Functional-analytic and numerical issues in splitting methods for total variation-based image reconstruction*, Inverse Problems, 30 (2014), 055014.

[32] K. Ito and K. Kunisch, *Lagrange Multiplier Approach to Variational Problems and Applications*, SIAM, Philadelphia, 2008.

[33] T. Kärkkäinen and K. Majava, *Nonmonotone and monotone active-set methods for image restoration, part* 1*: Convergence analysis*, J. Optim. Theory Appl., 106 (2000), pp. 61–80.

[34] T. Kärkkäinen and K. Majava, *Nonmonotone and monotone active-set methods for image restoration, part* 2*: Numerical results*, J. Optim. Theory Appl., 106 (2000), pp. 81–105.

[35] T. Kärkkäinen, K. Majava, and M. M. Mäkelä, *Comparison of formulations and solution methods for image restoration problems*, Inverse Problems, 17 (2001), pp. 1977–1995.

[36] Y. A. Kuznetsov, *Numerical methods in subspaces*, Vychislitel'-nye Processy i Sistemy II, 37 (1985), pp. 265–350.

[37] Yu. A. Kuznetsov and T. Rossi, *Fast direct method for solving algebraic systems with separable symmetric band matrices*, East-West J. Numer. Math., 4 (1996), pp. 53–68.

[38] Y. Meyer, *Oscillating Patterns in Image Processing and Nonlinear Evolution Equations: The Fifteenth Dean Jacqueline B. Lewis Memorial Lectures*, AMS, Providence, RI, 2001.

[39] D. Mumford, *Elastica and computer vision*, in Algebraic Geometry and Its Applications, Springer-Verlag, New York, 1994, pp. 491–506.

[40] M. Ng, P. Weiss, and X. Yuan, *Solving constrained total-variation image restoration and reconstruction problems via alternating direction methods*, SIAM J. Sci. Comput., 32 (2010), pp. 2710–2736.

[41] P. Perona and J. Malik, *Scale space and edge detection using anisotropic diffusion*, in Proceedings of IEEE Workshop on Computer Vision, IEEE, Washington, DC, 1987, pp. 16–27.

[42] P. Perona and J. Malik, *Scale-space and edge detection using anisotropic diffusion*, IEEE Trans. Pattern Anal. Mach. Intell., 12 (1990), pp. 629–639.

[43] T. Rossi and J. Toivanen, *A parallel fast direct solver for block tridiagonal systems with separable matrices of arbitrary dimension*, SIAM J. Sci. Comput., 20 (1999), pp. 1778–1796.

[44] L. Rudin, S. Osher, and E. Fatemi, *Nonlinear total variation based noise removal algorithms*, Phys. D, 60 (1992), pp. 259–268.

[45] Y. Shi, L.-L. Wang, and X.-C. Tai, *Geometry of total variation regularized l^p-model*, J. Comput. Appl. Math., 236 (2012), pp. 2223–2234.

[46] X.-C. Tai, J. Hahn, and G. J. Chung, *A fast algorithm for Euler's elastica model using augmented Lagrangian method*, SIAM J. Imaging Sci., 4 (2011), pp. 313–344.

[47] P. Vassilevski, *Fast algorithm for solving a linear algebraic problem with separable variables*, Comptes Rendus de Academie Bulgare des Sciences, 37 (1984), pp. 305–308.

[48] Y. Wang, J. Yang, W. Yin, and Y. Zhang, *A new alternating minimization algorithm for total variation image reconstruction*, SIAM J. Imaging Sci., 1 (2008), pp. 248–272.

[49] J. Weickert, *Anisotropic Diffusion in Image Processing*, B.G. Teubner, Stuttgart, Germany, 1998.

[50] C. Wu and X.-C. Tai, *Augmented Lagrangian method, dual methods, and split Bregman iteration for ROF, vectorial TV, and high order models*, SIAM J. Imaging Sci., 3 (2010), pp. 300–339.

[51] W. Zhu and T. Chan, *Image denoising using mean curvature of image surface*, SIAM J. Imaging Sci., 5 (2012), pp. 1–32.

[52] W. Zhu, X.-C. Tai, and T. Chan, *Augmented Lagrangian method for a mean curvature based image denoising model*, Inverse Probl. Imaging, 7 (2013), pp. 1409–1432.

PV

A GPU-ACCELERATED AUGMENTED LAGRANGIAN BASED L^1-MEAN CURVATURE IMAGE DENOISING ALGORITHM IMPLEMENTATION

by

Mirko Myllykoski, Roland Glowinski, Tommi Kärkkäinen, and Tuomo Rossi
2015

In M. Gavrilova, V. Skala, editors, WSCG 2015: 23rd International Conference in Central Europe on Computer Graphics, Visualization and Computer Vision'2015: Full Papers Proceedings, pages 119–128

A GPU-Accelerated Augmented Lagrangian Based L^1-mean Curvature Image Denoising Algorithm Implementation

Mirko Myllykoski[1]
University of Jyväskylä
mirko.myllykoski@jyu.fi

Roland Glowinski[2]
University of Houston
roland@math.uh.edu

Tommi Kärkkäinen[1]
University of Jyväskylä
tommi.karkkainen@jyu.fi

Tuomo Rossi[1]
University of Jyväskylä
tuomo.j.rossi@jyu.fi

Abstract

This paper presents a graphics processing unit (GPU) implementation of a recently published augmented Lagrangian based L^1-mean curvature image denoising algorithm. The algorithm uses a particular alternating direction method of multipliers to reduce the related saddle-point problem to an iterative sequence of four simpler minimization problems. Two of these subproblems do not contain the derivatives of the unknown variables and can therefore be solved point-wise without inter-process communication. In particular, this facilitates the efficient solution of the subproblem that deals with the non-convex term in the original objective function by modern GPUs. The two remaining subproblems are solved using the conjugate gradient method and a partial solution variant of the cyclic reduction method, both of which can be implemented relatively efficiently on GPUs. The numerical results indicate up to 33-fold speedups when compared against a single-threaded CPU implementation. The pointwise treated subproblem that takes care of the non-convex term in the original objective function was solved up to 76 times faster.

Keywords

augmented Lagrangian method, GPU computing, image denoising, image processing, mean curvature, OpenCL

1 INTRODUCTION

Image denoising, or more generally noise reduction, is a process in which a given noisy signal, such as a digital image, is cleared from excess noise. This process has numerous applications since all recording devices have some traits that make them susceptible to interference. For example, thermal noise effecting digital image sensors is a typical source of interference in digital photography. The noise must be removed, or at least significantly reduced, before essential information can be successfully extracted from an image.

Image denoising methods are divided into multiple subcategories. For example, wavelet methods are based around the idea of decomposing the image into the wavelet basis and shrinking (or otherwise modifying) the wavelet coefficients in order to denoise the image. A somewhat similar approach is to process the image in the frequency domain using the fast Fourier transformation (FFT) method. Statistical methods, on the other hand, utilize local statistical information, such as median and mean, from the neighboring pixels that fall within an appropriately selected window.

The most relevant sub-category to the topic of this paper is referred to as variational-based methods. These methods treat the noisy image as a discretely differentiable function and denoise the image using derivate information. In more formal terms, let Ω be a rectangular domain of $\mathbb{R}^2$ and the function $f : \Omega \to \mathbb{R}$ represent the given noisy image. We want to find a function $u : \Omega \to \mathbb{R}$ that is a representative of the desired denoised image. A variety of variational-based techniques have been developed and a significant proportion of them (see, e.g., [Mum94, Rud92]) are based on solving an unconstrained minimization problem of the form

$$\begin{cases} u \in V, \\ \mathscr{J}(u) \leq \mathscr{J}(v), \forall v \in V, \end{cases} \tag{1}$$

where

$$\mathscr{J}(v) = \varepsilon \mathscr{J}_r(v) + \mathscr{J}_f(v), \tag{2}$$

V is a suitable function space, and $\varepsilon > 0$. Here $\mathscr{J}_r$ is so-called regularization term and $\mathscr{J}_f$ is so-called fidelity

[1] University of Jyväskylä, Department of Mathematical Information Technology, P.O. Box 35, FI-40014 University of Jyväskylä, Finland.
[2] University of Houston, Department of Mathematics, Houston, TX 77204, USA.

term whose role is to fit the obtained solution u to the noisy data f. The work in this field of research is aimed primarily at finding a suitable regularization term that is able to detect noise but preserves as much relevant information as possible.

During the last two decades, the variational-based image denoising scene has been dominated to a large extent by Rudin–Osher–Fatemi (ROF) method [Rud92] which uses the following objective function:

$$\mathscr{J}(v) = \varepsilon \int_\Omega |\nabla v|\, dx + \frac{1}{2}\int_\Omega |f - v|^2\, dx. \tag{3}$$

Here, $\int_\Omega |\nabla v|\, dx$ is so-called total variation norm (TV norm) and the function space V made out of function whose total variation is bounded (a.k.a BV space). This very popular method has, however, some well-known drawbacks, such as the loss of image contrast, the smearing of corners, and the staircase effect.

Some attempts to remedy these drawbacks have led to higher-order variational-models which seek to take advantage of the higher-order derivatives. One approach suggested by Zhu and Chan [Zhu12] is to treat an image $v : \Omega \to \mathbb{R}$ as a surface in $\Omega \times \mathbb{R}$ and utilize the surface mean curvature information in the regularization term. More specifically, the surface in question is defined by the equation $F_v(x,y,z) = v(x,y) - z = 0$ and the mean curvature of the function F_v is given by

$$\kappa(F_v) = -\nabla \cdot \frac{\nabla F_v}{|\nabla F_v|} = -\nabla \cdot \frac{\nabla v}{\sqrt{1 + |\nabla v|^2}}. \tag{4}$$

All in all, the objective function used in the model suggested by Zhu and Chan is of the form:

$$\mathscr{J}(v) = \varepsilon \int_\Omega |\kappa(F_v)|\, dx + \frac{1}{2}\int_\Omega |f - v|^2\, dx. \tag{5}$$

This model is commonly known these days as the L^1-mean curvature denoising model. As noted in [Zhu12], this model has the ability to remove noise without the undesirable drawbacks associated with the ROF model. However, the non-convex and non-smooth nature of the objective function (5) makes the problem very difficult to solve.

1.1 Related work

Although the model suggested by Zhu and Chan is very difficult to solve as noted above, effective solution algorithms for this particular formulation have been proposed, for example, in [Zhu13] and [Myl15]. The solution algorithm presented in [Zhu13] uses an augmented Lagrangian based approach and solves the arising saddle-point problem using a particular alternating direction method of multipliers. This leads to an iterative sequence of five simpler minimization problems. These subproblems can be solved using explicit formulas and the FFT method. The solution algorithm presented in [Myl15] uses the same alternating direction approach as [Zhu13] but relies on a different type of augmented Lagrangian functional. Two of the four arising subproblems can be solved pointwise using the Newton's method, a bisection search algorithm, and explicit formulas. The two remaining subproblems can be solved using the conjugate gradient method and a partial solution variant of the cyclic reduction (PSCR) method [Kuz85, Kuz96, Vas84, Val85].

It should be noted that the model suggested by Zhu and Chan is closely related to the model depicted in [Lys04]. The model uses the following regularization term:

$$\mathscr{J}_r(v) = \int_\Omega \left| \nabla \cdot \frac{\nabla v}{|\nabla v|} \right| dx. \tag{6}$$

In [Lys04], the authors explained that their goal was to minimize the "TV norm" of the unit normal vectors of the level curves of the image. An alternative interpretation is that the regularization term (6) measures the total mean curvature at every level curve of the image. In contrast, the regularization term in the model suggested by Zhu and Chan measures the total mean curvature at the surface defined by the image (graph).

From a practical point of view the most relevant connection comes from the fact that the resulting model is often regularized in such a way that $|\nabla v|$ in (6) is replaced by $|\nabla v|_\beta = \sqrt{|\nabla v|^2 + \beta}$, $\beta > 0$. Thus, the solution algorithms developed for this denoising model and its variants (see, e.g., [Bri10, Sun14, Yan14]) could be in principle generalized for the model suggested by Zhu and Chan by taking $\beta = 1$.

The solution algorithm presented in [Bri10] solves the related Euler-Lagrange (EL) equation using a stabilized fixed point method and a geometric multigrid (MG) algorithm. The authors in [Sun14] aimed to improve upon that by introducing an additional operator splitting step. They then moved on to solving the EL equations associated with the related constrained minimization problem using a linearized fixed point method and a nonlinear MG method. In [Yan14], the problem is tacked with a relaxed fixed point method and a homotopy algorithm. The papers [Bri10, Sun14, Yan14] included comparisons where the value of the parameter β was varied (including the case $\beta = 1$). In other aspects these three recent papers were mostly interested in the case where the regularization term (6) is replaced by

$$\mathscr{J}_r(v) = \int_\Omega \left(\nabla \cdot \frac{\nabla v}{|\nabla v|_\beta} \right)^2 dx. \tag{7}$$

and $\beta \ll 1$.

1.2 Motivation and structure

Now that the overall context and main related works have been dealt with, we can move on to the main topic of this paper. We present a graphics processing unit (GPU) implementation of the solution algorithm depicted in [Myl15] and compare the GPU implementation against a single-threaded CPU implementation.

Our main motivation is that the most demanding step in the solution algorithm is a pointwise treated subproblem that handles the non-convex term in the original objective function. This makes the solution algorithm very suitable for GPU computation since the solution of this subproblem does not require inter-process communication. Thus, it is very likely that GPU-acceleration would bring significant performance benefits.

The rest of this paper is organized as follows: Section 2 describes the augmented Lagrangian based image denoising algorithm closely following the presentation in [Myl15]. Section 3 gives a brief introduction to GPU computing and describes the GPU implementation. Section 4 presents the numerical results, comparisons, and discussion. The final conclusions are given in Section 5.

2 SOLUTION ALGORITHM

2.1 Augmented Lagrangian formulation

Augmented Lagrangian techniques are a well-established framework for analyzing (constrained) optimization problems and deriving solution algorithms for such problems. When applied to convex minimization problems, the basic idea is to decompose the problem with the help of auxiliary variables. This so-called operator splitting operation effectively splits the problem into subproblems which can be treated separately using methods that are best suited for each subproblem. This greatly improves the effectiveness of the resulting solution algorithm.

The addition of these new auxiliary variables leads to a new constrained minimization problem that has the same minimizer as the original minimization problem. This constrained minimization is then associated with a suitable augmented Lagrangian functional whose saddle-point correspond to the minimizer of the constrained minimization problem. The saddle-points can be solved by, for example, using an alternating direction type approach. See, for example, [For83] for further information.

Although the objective function in the model suggested by Zhu and Chan is not convex, we will now describe a formal augmented Lagrangian formulation for the minimization problem similarly to [Myl15]. To begin with, let us define

$$\begin{aligned}\Upsilon = [V \times \mathbf{E}_{12} \times H(\Omega;\mathrm{div}) \times L^2(\Omega)] \\ \times [(L^2(\Omega))^2 \times (L^2(\Omega))^2 \times L^2(\Omega)],\end{aligned} \tag{8}$$

where

$$H(\Omega;\mathrm{div}) = \left\{\mathbf{q} \in (L^2(\Omega))^2 : \nabla\cdot\mathbf{q} \in L^2(\Omega)\right\} \tag{9}$$

and

$$\begin{aligned}\mathbf{E}_{12} = \Bigg\{ &(\mathbf{q}_1,\mathbf{q}_2) \in \left(L^2(\Omega)\right)^{2\times 2} \\ &: \mathbf{q}_2 = \frac{\mathbf{q}_1}{\sqrt{1+|\mathbf{q}_1|^2}} \Bigg\}.\end{aligned} \tag{10}$$

Following the remarks made in [Myl15], we take $V = H^2(\Omega)$. The minimization problem (1) with $\mathscr{J}$ defined by (5) is associated with the following augmented Lagrangian functional $\mathscr{L} : \Upsilon \to \mathbb{R}$:

$$\begin{aligned}&\mathscr{L}(v,\mathbf{q}_1,\mathbf{q}_2,\mathbf{q}_3,\varphi;\boldsymbol{\mu}_1,\boldsymbol{\mu}_2,\mu_3) \\ &= \varepsilon\int_\Omega |\varphi|\,dx + \frac{1}{2}\int_\Omega |f-v|^2\,dx \\ &+ \frac{r_1}{2}\int_\Omega |\nabla v - \mathbf{q}_1|^2\,dx + \int_\Omega \boldsymbol{\mu}_1\cdot(\nabla v - \mathbf{q}_1)\,dx \\ &+ \frac{r_2}{2}\int_\Omega |\mathbf{q}_2 - \mathbf{q}_3|^2\,dx + \int_\Omega \boldsymbol{\mu}_2\cdot(\mathbf{q}_2 - \mathbf{q}_3)\,dx \\ &+ \frac{r_3}{2}\int_\Omega |\nabla\cdot\mathbf{q}_3 - \varphi|^2\,dx \\ &+ \int_\Omega \mu_3(\nabla\cdot\mathbf{q}_3 - \varphi)\,dx,\end{aligned} \tag{11}$$

where $(\mathbf{q}_1,\mathbf{q}_2) \in \mathbf{E}_{12}$ and $r_i > 0$, $i = 1,2,3$. Above, $\mathbf{q}_1$, $\mathbf{q}_2$, $\mathbf{q}_3$, and φ are the previously mentioned auxiliary variables, and $\boldsymbol{\mu}_1$,$\boldsymbol{\mu}_2$, and μ_3 are called Lagrange multipliers. Note that the non-convex term $\mathbf{q}_2 = \frac{\mathbf{q}_1}{\sqrt{1+|\mathbf{q}_1|^2}}$ is treated by projection in (10) and thus does not appear in the augmented Lagrangian functional $\mathscr{L}$.

Now, if we can find a saddle-point

$$\omega = (u,\mathbf{p}_1,\mathbf{p}_2,\mathbf{p}_3,\psi;\boldsymbol{\lambda}_1,\boldsymbol{\lambda}_2,\lambda_3) \in \Upsilon \tag{12}$$

for the augmented Lagrangian $\mathscr{L}$, that is

$$\begin{aligned}&\mathscr{L}(u,\mathbf{p}_1,\mathbf{p}_2,\mathbf{p}_3,\psi;\boldsymbol{\mu}_1,\boldsymbol{\mu}_2,\mu_3) \\ &\le \mathscr{L}(u,\mathbf{p}_1,\mathbf{p}_2,\mathbf{p}_3,\psi;\boldsymbol{\lambda}_1,\boldsymbol{\lambda}_2,\lambda_3) \\ &\le \mathscr{L}(v,\mathbf{q}_1,\mathbf{q}_2,\mathbf{q}_3,\varphi;\boldsymbol{\lambda}_1,\boldsymbol{\lambda}_2,\lambda_3),\end{aligned} \tag{13}$$

for all $(v,\mathbf{q}_1,\mathbf{q}_2,\mathbf{q}_3,\varphi;\boldsymbol{\mu}_1,\boldsymbol{\mu}_2,\mu_3) \in \Upsilon$, then

$$\begin{aligned}\mathbf{p}_1 = \nabla u,\ \mathbf{p}_2 = \frac{\mathbf{p}_1}{\sqrt{1+|\mathbf{p}_1|^2}}, \\ \mathbf{p}_3 = \mathbf{p}_2,\ \psi = \nabla\cdot\mathbf{p}_3,\end{aligned} \tag{14}$$

and, more importantly, u is a local minimizer of the minimization problem (1) with $\mathscr{J}$ defined by (5).

2.2 Subproblems

In [Myl15], the saddle-point problem (12) – (13) is solved using a particular alternating direction method of multipliers called ALG-2 [For83, Glo89]. The idea is to minimize the augmented Lagrangian functional (11) one variable at a time until the method converges. The Lagrange multipliers are update accordingly after each iteration.

Since we have five variables and two of the auxiliary variables ($\mathbf{q}_1$ and $\mathbf{q}_2$) are coupled together, this leads to an iterative sequential solution of four subproblems. More precisely, the task of finding a saddle-point for the augmented Lagrangian functional (11) is transformed into one smooth but nonlinear and non-convex minimization problem in $\mathbb{R}^2$, one purely explicit pointwise treated minimization problem, and two linear minimization problems with positive definite and symmetric coefficient matrices.

Each outer iteration is defined as follows: Let $(u^n, \mathbf{p}_1^n, \mathbf{p}_2^n, \mathbf{p}_3^n, \psi^n; \boldsymbol{\lambda}_1^n, \boldsymbol{\lambda}_2^n, \lambda_3^n) \in \Upsilon$ be the output of the previous iteration. *The first subproblem* minimizes the augmented Lagrangian functional (11) with respect to the pair $(\mathbf{q}_1, \mathbf{q}_2) \in \mathbf{E}_{12}$. Using the nonlinear relation in (10), the function $\mathbf{p}_1^{n+1}$ can be solved pointwise from the following non-convex minimization problem:

$$\mathbf{x} = \arg\min_{\mathbf{y}\in\mathbb{R}^2} \left[\frac{|\mathbf{y}|^2}{2} \left(r_1 + \frac{r_2}{1+|\mathbf{y}|^2} \right) - \left(\mathbf{b}_1 + \frac{\mathbf{b}_2}{\sqrt{1+|\mathbf{y}|^2}} \right) \cdot \mathbf{y} \right], \tag{15}$$

where $\mathbf{b}_1, \mathbf{b}_2 \in \mathbb{R}^2$ depend on the other variables. Or, alternatively, by noticing that the following nonlinear relation must hold

$$\mathbf{x} = \alpha \left(\mathbf{b}_1 + \frac{\mathbf{b}_2}{\sqrt{1+|\mathbf{x}|^2}} \right), \tag{16}$$

where $\alpha \geq 0$, we can write the two-dimensional minimization problem (15) as

$$\mathbf{x} = \frac{\rho}{\left| \mathbf{b}_1 + \frac{\mathbf{b}_2}{\sqrt{1+\rho^2}} \right|} \left(\mathbf{b}_1 + \frac{\mathbf{b}_2}{\sqrt{1+\rho^2}} \right), \tag{17}$$

where

$$\rho = \arg\min_{\sigma\in[0,\infty)} \left[\frac{\sigma^2}{2} \left(r_1 + \frac{r_2}{1+\sigma^2} \right) - \sigma \left| \mathbf{b}_1 + \frac{\mathbf{b}_2}{\sqrt{1+\sigma^2}} \right| \right]. \tag{18}$$

In *the second subproblem* we minimize the augmented Lagrangian functional (11) with respect to the variable $\mathbf{q}_3$. More specifically, we solve the following linear vector-valued minimization problems with positive definite and symmetric coefficient matrix:

$$\begin{aligned} &\mathbf{p}_3^{n+1} \in H(\Omega; \mathrm{div}), \\ r_2 \int_\Omega \mathbf{p}_3^{n+1} \cdot \mathbf{q}\, dx + r_3 \int_\Omega \nabla\cdot \mathbf{p}_3^{n+1} \nabla\cdot \mathbf{q}\, dx \\ &= \int_\Omega \left(r_2 \mathbf{p}_2^{n+1} + \boldsymbol{\lambda}_2^n \right) \cdot \mathbf{q}\, dx \\ &+ \int_\Omega (r_3 \psi^n - \lambda_3^n) \nabla\cdot \mathbf{q}\, dx, \\ &\forall \mathbf{q} \in H(\Omega; \mathrm{div}). \end{aligned} \tag{19}$$

The third subproblem minimizes the augmented Lagrangian functional (11) with respect to the variable φ and is of the form:

$$\psi^{n+1} = \arg\min_{\varphi\in L^2(\Omega)} \left[\varepsilon \int_\Omega |\varphi|\, dx + \frac{r_3}{2} \int_\Omega |\varphi|^2\, dx - \int_\Omega \left(r_3 \nabla\cdot \mathbf{p}_3^{n+1} + \lambda_3^n \right) \varphi\, dx \right]. \tag{20}$$

The minimization problem (20) has a closed-form solution

$$\psi^{n+1}(x) = \frac{1}{r_3} \operatorname{sgn}(\xi(x)) \max(0, |\xi(x)| - \varepsilon), \tag{21}$$

with $\xi(x) = (r_3 \nabla\cdot \mathbf{p}_3^{n+1} + \lambda_3^n)(x)$.

The fourth subproblem minimizes the augmented Lagrangian functional (11) with respect to the variable v. The subproblem can be written as the following linear scalar-valued minimization problems with positive definite, symmetric, and separable coefficient matrix:

$$\begin{aligned} &u^{n+1} \in V, \\ r_1 \int_\Omega \nabla u^{n+1} \cdot \nabla v\, dx + \int_\Omega (u^{n+1} - f)\, v\, dx \\ &= \int_\Omega \left(r_1 \mathbf{p}_1^{n+1} - \boldsymbol{\lambda}_1^n \right) \cdot \nabla v\, dx, \forall v \in V. \end{aligned} \tag{22}$$

Finally, *the Lagrange multipliers* are updated as follows:

$$\begin{aligned} \boldsymbol{\lambda}_1^{n+1} &= \boldsymbol{\lambda}_1^n + r_1 (\nabla u^{n+1} - \mathbf{p}_1^{n+1}), \\ \boldsymbol{\lambda}_2^{n+1} &= \boldsymbol{\lambda}_2^n + r_2 (\mathbf{p}_2^{n+1} - \mathbf{p}_3^{n+1}), \\ \lambda_3^{n+1} &= \lambda_3^n + r_3 (\nabla\cdot \mathbf{p}_3^{n+1} - \psi^{n+1}). \end{aligned} \tag{23}$$

2.3 Finite element realization

The domain Ω is triangulated using a uniform finite element triangulation $\mathscr{T}_h$. The function space V is approximated by a piecewise linear finite element space

$$V_h = \left\{ v \in C^0(\bar{\Omega}) : v|_T \in P_1, \forall\, T \in \mathscr{T}_h \right\}, \tag{24}$$

where P_1 is the space of the polynomials of two variables of degree ≤ 1. The spaces $(L^2(\Omega))^2$ and

$H(\Omega;\mathrm{div})$ are approximated by the following piecewise constant finite element space:

$$\mathbf{Q}_h = \left\{ \mathbf{q} \in (L^\infty)^2 : \mathbf{q}|_T \in (P_0)^2, \forall\, T \in \mathscr{T}_h \right\}, \tag{25}$$

where P_0 is the space of the constant functions. Clearly, we have $\nabla V_h \subset \mathbf{Q}_h$.

Let $\{X_j\}_{j=1}^{N_h}$ be the set of vertices of $\mathscr{T}_h$ and $\mathbf{q} \in \mathbf{Q}_h$. The divergence operator is approximated by using an appropriate discrete Green's formula and the trapezoidal rule as follows:

$$(\mathrm{div}_h \mathbf{q})(X_j) = -\frac{3}{|\Omega_j|} \int_{\Omega_j} \mathbf{q} \cdot \nabla w_j \, dx, \tag{26}$$

where X_j is a vertex that does not belong to $\partial\Omega$, Ω_j is the polygon that is the union of those triangles of $\mathscr{T}_h$ that have X_j as a common vertex, $|\Omega_j|$ is the measure of Ω_j, and the shape function $w_j \in V_h$ is uniquely defined as

$$\begin{cases} w_j(X_j) = 1, \\ w_j(X_k) = 0,\ k \neq j. \end{cases} \tag{27}$$

3 GPU IMPLEMENTATION

3.1 GPU computing and OpenCL

The GPU implementation presented in this paper is written using the OpenCL framework. This section introduces the reader to general OpenCL concepts and terminology. Some additional information related to Nvidia's current hardware is provided since that information is essential for the understanding of the implementation and obtained numerical results.

A contemporary high-end GPU contains thousands of processing elements (cores) which are grouped into multiple computing units. The processing elements inside the same computing unit share a fast (on-chip) memory space called local memory which can be used for sharing data among the processing elements. The local memory is divided into 32-bit (or 64-bit) memory banks organized in such a way that successive 32-bit (or 64-bit) words map to successive memory banks. Multiple processing elements also share the same scheduler, which means that the processing elements are executing the program code in a synchronous manner. In addition to the local memory, all processing elements can access a much larger but slower (off-chip) memory space called global memory. The global memory can serve memory requests at the optimal rate when processing elements are synchronously accessing data that is located inside a same memory block.

GPU-side program code execution is based on the concept of a special kind of subroutine called (OpenCL) kernel. All work-items (threads) start from the beginning of the kernel but each work-item is given a unique index number which allows the execution paths of different work-items to branch off. The work-items are grouped into work groups which are also given unique index numbers and a work group can share a portion of the local memory. Nvidia uses the term warp when referring to a set of work-items that are executed together in a synchronized manner. Diverging execution paths, also known as warp divergences, lead to a suboptimal performance as all the necessary paths have to be evaluated by the whole warp. In contemporary Nvidia GPUs the warp size is 32 work-items.

3.2 General notes

The GPU implementation is principally identical with the CPU implementation described in [Myl15] but the low level details vary considerably. The less simplified two-dimensional form of the critical non-convex subproblem (15) is initially solved using the Newton's method, whose solution candidate is then tested against the explicit relation (16). If the solution candidate does not fulfill the explicit relation, the implementation proceeds to the one-dimensional form (18) which is solved using the bisection search algorithm as described in [Myl15].

The linear vector-valued subproblem (19) is solved using the conjugate gradient algorithm without preconditioning. While more generalized GPU implementations have been presented in the past (see, for example, [Ame10, Bol03, Hel12]), the conjugate gradient solver used in the GPU implementation described in this paper was tailored for this specific subproblem and the matrix-vector multiplication operation was hard coded into the kernels. The explicit subproblem (20) is solved using the closed form solution (21) and the linear scalar-valued subproblem (22) is solved using the PSCR method.

All computational operations are carried out in the GPU side. The floating point division operation was accelerated using a Newton-Raphson division algorithm [Fly70] and an initial approximation that leads to full double precision accuracy with only four iterations [Par92].

3.3 Element numbering

The elements of the finite element space V_h are numbered in a row-wide fashion. This means that the coefficient matrix in the linear scalar-valued subproblem (22) is block tridiagonal and presentable in a separable form using so-called Kronecker matrix tensor product. This is required by the PSCR method.

The numbering of the elements of the finite element space $\mathbf{Q}_h$ can be chosen more freely. Figure 1 shows two possible numbering schemes. If the numbering scheme shown on the left (referred to hereinafter as

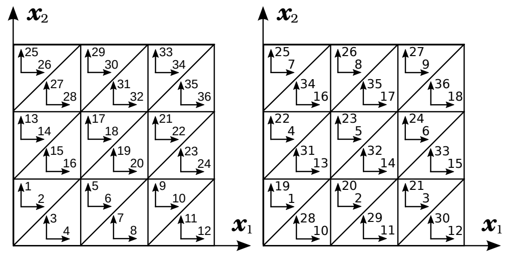


Figure 1: Two possible numbering schemes for the elements of the finite element space $\mathbf{Q}_h$ (3×3 grid): the dense numbering scheme (on the left) and the sparse numbering scheme (on the right).

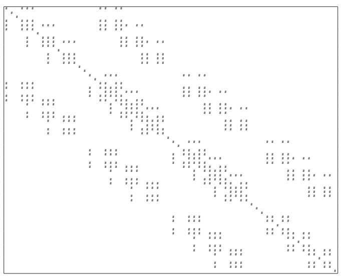


Figure 2: The non-zero elements of the coefficient matrix in the linear vector-valued subproblem (19) when the dense numbering scheme is used (4×4 grid).

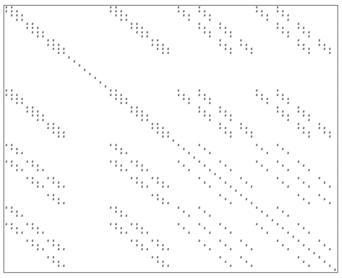


Figure 3: The non-zero elements of the coefficient matrix in the linear vector-valued subproblem (19) when the sparse numbering scheme is used (4×4 grid).

the dense numbering scheme) is chosen, then the coefficient matrix in the linear vector-valued subproblem (19) is of the form shown in Figure 2. On the one hand, if the numbering scheme shown on the right (referred to hereinafter as the sparse numbering scheme) is chosen, then the coefficient matrix is of the form shown in Figure 3. Each numbering scheme has its own advantages and disadvantages.

The dense numbering scheme leads to a more optimal global memory access pattern during the solution of the linear vector-valued subproblem (19) as the non-zero elements of the coefficient matrix are packed tightly to three bands and elements of each band can be shared among work-items using the local memory. This is particularly important because many contemporary high end GPUs have an extremely high peak floating-point performance but a relatively low peak global memory bandwidth. Thus, the use of the global memory should be kept at minimum. The most significant downside of this numbering scheme is that a straightforward implementation would lead to warp divergences throughout the implementation. Most of these warp divergences could be avoided by re-arranging the computational tasks appropriately with the help of the local memory. However, this re-arranging would complicate the implementation considerably and introduce memory bank conflicts in many places.

The sparse numbering scheme leads to a simpler implementation but the pattern of non-zero elements in the coefficient matrix is much more fragmented. This means that less data can be shared between the work-items using the local memory and, thus, the global memory usage increases significantly. Despite this, the sparse numbering scheme was chosen for the GPU implementation described in this paper because it was not clear whether this choice would lead to an actual global memory bottleneck that would limit the performance of the whole GPU implementation. In addition, if the dense numbering scheme is chosen, then the increased complexity in the other parts of the implementation might negate the potential benefits. The reference CPU implementation uses the dense numbering scheme because it allows more effective utilization of the CPU caches.

3.4 PSCR implementation

The PSCR method [Kuz85, Kuz96, Vas84, Val85] is a block cyclic reduction type direct solver which can be applied to certain separable block tridiagonal linear systems. To put it briefly, the PSCR method solves the linear scalar-valued subproblem (22) by recursively eliminating block-rows from the corresponding linear system and then solves the generated sub-systems in the reverse order during so-called back substitution stage. Each reduction and back substitution step produces a large set of tridiagonal linear system.

The GPU implementation of the PSCR method used in this paper is based on the radix-4 variant described in [Ros99] and it is in many respects similar to the simplified radix-4 block cyclic reduction GPU implementation presented in [Myl13]. However, the GPU implementation used in this paper is much more generalized as the problem size can be arbitrary.

The arising tridiagonal subproblems are solved using the cyclic reduction (CR) [Hoc65], the parallel cyclic reduction (PCR) [Hoc81] and the Thomas [Con80] methods. If a tridiagonal system does not fit into the allocated local memory buffer,then the system size is first reduced using the CR method and the global memory data permutations depicted in [Myl13]. The tridiagonal systems that do fit into the allocated local memory buffer are solved using a CR-PCR-Thomas hybrid method. The CR stage of the hybrid solver uses the local memory data permutations depicted in [Myl13]. The PCR stage further splits the reduced tridiagonal systems into smaller subsystems which are eventually solved using the Thomas method in a manner similar to [Dav11, Kim11]. Somewhat similar tridiagonal solver techniques have been used, for example, in [Göd11, Lam12, Zha10].

4 NUMERICAL RESULTS

4.1 Test setting

GPU tests were carried out on a few years old consumer level Nvidia GeForce GTX580 GPU and a high-end computing orientated Nvidia Tesla K40c GPU. The CPU implementation is the same as was used in [Myl15]. It is written using C++ and Fortran. It utilizes a single-threaded variant of the radix-4 PSCR method presented in [Ros99]. CPU tests were carried out using an Intel Xeon E5-2630 v2 (2.60GHz) CPU. All the tests were performed using double precision floating point arithmetic and ECC memory (excluding the GTX580 GPU which does not support ECC).

Four test images (shown in Figure 4) were used in the numerical tests: Test9, Lena, Boat, and Mercado. In addition, four different sized versions of the test image Mercado were included: 256 $\times$ 256, 512 $\times$ 512, 1024 $\times$ 1024, and 2048 $\times$ 2048. The dimensions of the other test images are 512 $\times$ 512. The original test images were scaled to the range $[0,1]$. Two sets of noisy input images were generated: uniformly distributed zero-mean noise with the standard deviation $\sigma = 0.025$ and uniformly distributed zero-mean noise with the standard deviation $\sigma = 0.1$.

Based on the remarks made in [Myl15], the following initialization was used:

$$\begin{aligned} u^0 = 0,\ \mathbf{q}_1^0 = \mathbf{q}_2^0 = \mathbf{q}_3^0 = 0,\ \psi^0 = 0, \\ \boldsymbol{\lambda}_1^0 = \boldsymbol{\lambda}_2^0 = 0,\ \lambda_3^0 = 0. \end{aligned} \tag{28}$$

In the same way, the stopping criterion read as

$$\frac{|\mathscr{L}_h(\upsilon^n) - \mathscr{L}_h(\upsilon^{n+1})|}{|\mathscr{L}_h(\upsilon^n)|} < 10^{-4}, \tag{29}$$

where $\mathscr{L}_h$ is the discrete counterpart of the augmented Lagrangian functional (11) and $\upsilon^n = (u^n, \mathbf{p}_1^n, \mathbf{p}_2^n, \mathbf{p}_3^n, \psi^n; \boldsymbol{\lambda}_1^n, \boldsymbol{\lambda}_2^n, \lambda_3^n)$. The parameter ε and the Lagrangian multipliers were coupled as follows: $\varepsilon = r_0 h$, $r_1 = 10 r_0 h$, $r_2 = 5 r_0$, and $r_3 = 5 r_0 h^2$, where $r_0 > 0$. We took $h = 0.005$ for the spatial discretization step.

4.2 Comparisons

Figure 4 shows the original test images, the generated input images ($\sigma = 0.1$), and the obtained output images. Table 1 shows the used parameter values, iteration counts and execution times for the Intel Xeon CPU. The input images and parameter values were the same for the three platforms. In addition, as the GPUs used in the numerical experiments are fully IEEE 754 compliant, the iteration counts, objective function values, and output images were also identical.

The "Whole" column shows the average total per iteration execution times; the "Sub. #1", "Sub. #2", "Sub. #3", and "Sub. #4" columns show the average per iteration execution times for each subproblem; and the "Misc." column shows the combined average per iteration execution times for augmentation term update kernels and an objective function value computation kernel. The CPU results show that a significant portion of the total execution time goes to solving the critical non-convex subproblem (15).

Tables 2 and 3 show the average per iteration execution times and the obtained speedups for the GTX580 and K40c GPUs, respectively. The GTX580 was on average 15.6 times faster than the Xeon CPU. The highest speedups were obtained in the case of the synthetic test image Test9 in which case the GTX580 GPU was up to 21.5 times faster. The K40c GPU was on average 26.0 times faster than the Intel Xeon CPU and the Test9 test image was processed up to 33.7 times faster. Both GPUs achieved the highest speedups in the case of the critical non-convex subproblem (15). The K40c GPU was at its best 76.0 times faster than the Intel Xeon CPU at solving the subproblem.

4.3 Discussion

A significant portion of the total execution time still goes to solving the critical non-convex subproblem (15) but the gap between it and the linear vector-valued subproblems (19) has narrowed considerably. However, even if we managed to overcome the potential global memory bottleneck associated with the linear vector-valued subproblems (19), the critical non-convex subproblem (15) would still dominate the total execution time in such a degree that it probably would not be of a significant improvement. Finally, the speedups obtained with the Mercado test images show that GPU's computational resources can be utilized best when the image size is relatively large.

Although the highest speedups were obtained in the case of the critical non-convex subproblem (15), the K40c GPU did not perform quite as well as expected. One culprit might be the Newton-bisection hybrid method which was used to solve the subproblem. For example, in the case of the Lena ($\sigma = 0.1$) input image, the Newton's method had an average success rate of

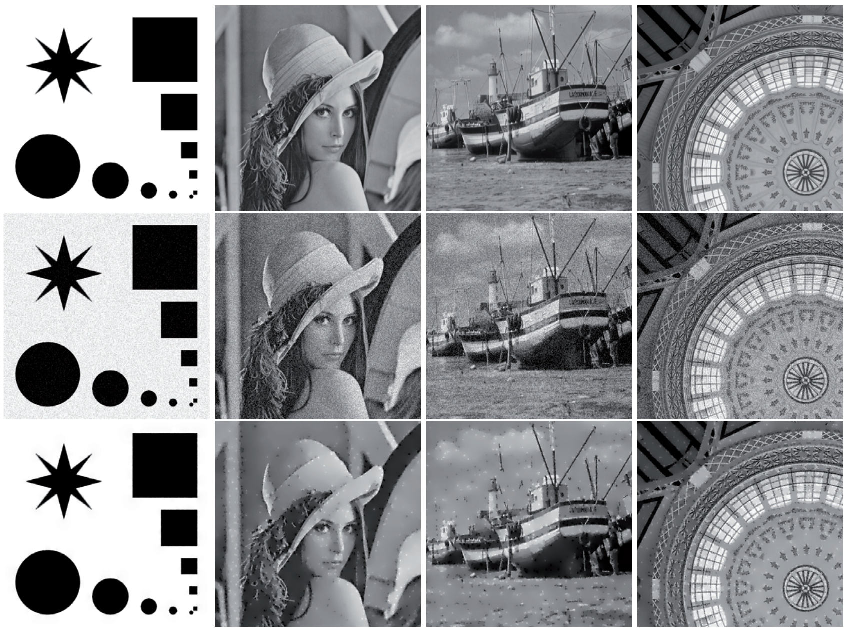


Figure 4: From top to bottom: the original images, the noisy input images ($\sigma = 0.1$), and the obtained output images. From left to right: Test9, Lena, Boat, and Mercado512[3].

Image	r_0	Iter.	Whole	Sub. #1	Sub. #2	Sub. #3	Sub. #4	Misc.
Test9, $\sigma = 0.025$	0.005	103	0.6170	0.4489	0.0985	0.0031	0.0412	0.0206
Lena, $\sigma = 0.025$	0.002	77	0.7463	0.5676	0.1097	0.0031	0.0405	0.0206
Boat, $\sigma = 0.025$	0.001	75	0.7967	0.6154	0.1115	0.0031	0.0412	0.0206
Mercado256, $\sigma = 0.025$	0.002	125	0.1719	0.1335	0.0233	0.0008	0.0081	0.0050
Mercado512, $\sigma = 0.025$	0.002	143	0.6813	0.5147	0.0977	0.0031	0.0406	0.0206
Mercado1024, $\sigma = 0.025$	0.002	172	2.6356	1.9665	0.3869	0.0125	0.1670	0.0847
Mercado2048, $\sigma = 0.025$	0.002	169	10.691	7.7660	1.6705	0.0499	0.7866	0.3460
Test9, $\sigma = 0.1$	0.015	163	0.5985	0.4348	0.0942	0.0031	0.0412	0.0206
Lena, $\sigma = 0.1$	0.005	158	0.6881	0.5189	0.0997	0.0031	0.0412	0.0206
Boat, $\sigma = 0.1$	0.005	163	0.6889	0.5181	0.1013	0.0031	0.0412	0.0206
Mercado256, $\sigma = 0.1$	0.005	213	0.1692	0.1309	0.0232	0.0008	0.0082	0.0050
Mercado512, $\sigma = 0.1$	0.005	205	0.6778	0.5104	0.0979	0.0031	0.0412	0.0206
Mercado1024, $\sigma = 0.1$	0.005	208	2.6719	1.9906	0.3958	0.0125	0.1694	0.0857
Mercado2048, $\sigma = 0.1$	0.005	205	10.661	7.7299	1.6658	0.0500	0.7982	0.3458

Table 1: Parameter values, iteration counts, and average per iteration execution times (in seconds) for the Intel Xeon CPU.

99.40%. This is perfectly fine for the CPU since the cost of processing the remaining triangles using the bisection search algorithm is neglectable. However, on the basis of the same data, there is on average 16.59% probability that an individual warp contains a work-item that has to process a triangle using the bisection search algorithm. This has a significant impact on the performance since the cost of processing a single triangle in this way is the same as processing similarly all the 32 triangles as the longest execution path of work-items within the warp determines the cost of completing the computational task assigned to this warp.

The above does not, however, explain why the considerably more powerful K40c GPU did not outperform the GTX580 GPU in such a large extent as would have been expected. The results could be partly explained by the fact that, based on our measurements, the K40c GPU is only 2-3 times faster than the GTX580 GPU at performing special operations such as computing reciprocals and square roots. The Newton-Raphson division algorithm improved performance less than 10%. In addition, we noticed that the K40c GPU was unusually sensitive to how the work group size was chosen. The critical non-convex subproblem (15) required us to set

[3] The Mercado test image is based on the works of Diego Delso and licensed under Wikimedia Commons license CC-BY-SA 3.0 (http://creativecommons.org/licenses/by-sa/3.0/legalcode).

Image	Whole		Sub. #1		Sub. #2		Sub. #3		Sub. #4		Misc.	
Test9, $\sigma = 0.025$	0.0287	**21.5**	0.0147	30.5	0.0085	11.6	0.0002	17.7	0.0041	10.0	0.0012	17.3
Lena, $\sigma = 0.025$	0.0507	**14.7**	0.0357	15.9	0.0095	11.5	0.0002	17.7	0.0041	9.9	0.0012	17.3
Boat, $\sigma = 0.025$	0.0654	**12.2**	0.0501	12.3	0.0097	11.5	0.0002	17.7	0.0041	10.0	0.0012	17.4
Mercado256, $\sigma = 0.025$	0.0136	**12.6**	0.0095	14.0	0.0027	8.6	0.0001	14.1	0.0010	8.5	0.0003	17.1
Mercado512, $\sigma = 0.025$	0.0472	**14.4**	0.0335	15.4	0.0082	11.8	0.0002	17.9	0.0041	10.0	0.0012	17.3
Mercado1024, $\sigma = 0.025$	0.1681	**15.7**	0.1124	17.5	0.0296	13.1	0.0006	19.6	0.0208	8.0	0.0046	18.5
Mercado2048, $\sigma = 0.025$	0.6457	**16.6**	0.4095	19.0	0.1178	14.2	0.0025	20.0	0.0975	8.1	0.0182	19.0
Test9, $\sigma = 0.1$	0.0278	**21.5**	0.0142	30.6	0.0080	11.7	0.0002	17.8	0.0042	9.9	0.0012	17.2
Lena, $\sigma = 0.1$	0.0454	**15.1**	0.0312	16.6	0.0087	11.4	0.0002	17.8	0.0041	10.0	0.0012	17.3
Boat, $\sigma = 0.1$	0.0452	**15.2**	0.0310	16.7	0.0087	11.6	0.0002	17.8	0.0041	10.0	0.0012	17.3
Mercado256, $\sigma = 0.1$	0.0133	**12.7**	0.0093	14.1	0.0027	8.5	0.0001	14.2	0.0009	8.7	0.0003	17.1
Mercado512, $\sigma = 0.1$	0.0464	**14.6**	0.0325	15.7	0.0084	11.7	0.0002	17.9	0.0041	10.1	0.0012	17.3
Mercado1024, $\sigma = 0.1$	0.1726	**15.5**	0.1167	17.1	0.0303	13.1	0.0006	19.6	0.0203	8.3	0.0046	18.7
Mercado2048, $\sigma = 0.1$	0.6440	**16.6**	0.4075	19.0	0.1178	14.1	0.0025	20.0	0.0977	8.2	0.0182	19.0
Average speedup		**15.6**		18.2		11.8		17.8		9.3		17.7

Table 2: Average per iteration execution times (in seconds) and obtained speedups for the Nvidia GeForce GTX580 GPU.

Image	Whole		Sub. #1		Sub. #2		Sub. #3		Sub. #4		Misc.	
Test9, $\sigma = 0.025$	0.0183	**33.7**	0.0059	76.0	0.0078	12.7	0.0001	29.7	0.0036	11.3	0.0009	23.3
Lena, $\sigma = 0.025$	0.0297	**25.1**	0.0163	34.8	0.0087	12.5	0.0001	29.5	0.0036	11.2	0.0009	23.4
Boat, $\sigma = 0.025$	0.0361	**22.1**	0.0225	27.3	0.0089	12.6	0.0001	29.5	0.0036	11.3	0.0009	23.4
Mercado256, $\sigma = 0.025$	0.0091	**18.8**	0.0045	29.4	0.0031	7.6	0.0000	16.6	0.0012	7.0	0.0003	17.1
Mercado512, $\sigma = 0.025$	0.0277	**24.6**	0.0153	33.6	0.0077	12.6	0.0001	29.9	0.0037	11.1	0.0009	23.2
Mercado1024, $\sigma = 0.025$	0.0970	**27.2**	0.0509	38.6	0.0262	14.8	0.0003	37.3	0.0163	10.2	0.0033	26.0
Mercado2048, $\sigma = 0.025$	0.3736	**28.6**	0.1841	42.2	0.1033	16.2	0.0012	40.6	0.0721	10.9	0.0127	27.2
Test9, $\sigma = 0.1$	0.0178	**33.6**	0.0057	75.7	0.0074	12.7	0.0001	29.5	0.0037	11.2	0.0009	23.3
Lena, $\sigma = 0.1$	0.0269	**25.5**	0.0143	36.3	0.0080	12.5	0.0001	29.6	0.0036	11.4	0.0009	23.4
Boat, $\sigma = 0.1$	0.0272	**25.4**	0.0144	35.9	0.0080	12.6	0.0001	29.7	0.0037	11.2	0.0009	23.2
Mercado256, $\sigma = 0.1$	0.0088	**19.1**	0.0045	29.3	0.0031	7.6	0.0000	16.0	0.0010	8.4	0.0003	18.2
Mercado512, $\sigma = 0.1$	0.0274	**24.8**	0.0149	34.2	0.0077	12.6	0.0001	29.8	0.0037	11.2	0.0009	23.2
Mercado1024, $\sigma = 0.1$	0.0996	**26.8**	0.0530	37.5	0.0267	14.8	0.0003	37.5	0.0162	10.4	0.0032	26.4
Mercado2048, $\sigma = 0.1$	0.3721	**28.7**	0.1826	42.3	0.1033	16.1	0.0012	40.7	0.0721	11.1	0.0127	27.2
Average speedup		**26.0**		40.9		12.7		30.4		10.6		23.5

Table 3: Average per iteration execution times (in seconds) and obtained speedups for the Nvidia Tesla K40c GPU.

the work group size as low as 64 work-items. In turn, the GTX580 GPU performed just fine when the work group size was set as high as 512 work-items. This suggest that Nvidia's OpenCL compiler might have problems with resource management. In general, the compiler seems to generate less optimal code for the K40c GPU in many situations. It also appears that the K40c GPU does not perform well in situations where the solution of a subproblem requires multiple kernel launches.

5 CONCLUSIONS

This paper presented a GPU implementation of an augmented Lagrangian based L^1-mean curvature image denoising algorithm and numerical results obtained while comparing the GPU implementation against a single-threaded CPU implementation. Up to 33-fold speedups were obtained, the average speedup being 26-fold. The pointwise handled non-convex subproblem predictably benefited most from the GPU-acceleration. The numerical results indicate that GPUs provide demonstrable benefits in the context of the higher-order variational-based image denoising algorithms and alternating direction type augmented Lagrangian methods.

6 ACKNOWLEDGMENTS

The authors thank anonymous reviewers for their valuable feedback. The presentation of our paper was significantly improved thanks to their comments and suggestions. The research of the first author was supported by the Academy of Finland (grant #252549), the Jyväskylä Doctoral Program in Computing and Mathematical Sciences (COMAS), and the Foundation of Nokia Corporation.

7 REFERENCES

[Ame10] Ament, M., Knittel, G., Weiskopf, D., and Strasser, W. A parallel preconditioned conjugate gradient solver for the Poisson problem on a multi-GPU platform. In: 18th Euromicro International Conference on Parallel, Distributed and Network-Based Processing (PDP), pp. 583–592, IEEE, 2010.

[Bri10] Brito-Loeza, C., Chen, K. Multigrid algorithm for high order denoising. SIAM J. Imaging Sci., 3(3), pp. 363–389, 2010.

[Bol03] Bolz, J., Farmer, I., Grinspun, E., and Schröoder, P. Sparse matrix solvers on the GPU: conjugate gradients and multigrid. ACM Trans. Graph., 22(3), pp. 917–924, 2003.

[Con80] Conte, S. D., and de Boor, C. Elementary numerical analysis: an algorithmic approach. Mcgraw-Hill College, 1980.

[Dav11] Davidson, A., Zhang, Y., and Owens, J. D. An auto-tuned method for solving large tridiagonal systems on the GPU. In: Proceedings of the 25th IEEE International Parallel and Distributed Processing Symposium, pp. 956–965, IEEE, 2011.

[Fly70] Flynn, M. On division by functional iteration. IEEE T. Comput., C19(8), pp. 702–706, 1970.

[For83] Fortin, M., and Glowinski, R. Augmented Lagrangian methods: applications to the numerical solution of boundary value problems. North-Holland, Amsterdam, 1983.

[Glo89] Glowinski, R., and Le Tallec, P.. Augmented Lagrangian and operator-splitting methods in nonlinear mechanics. SIAM, Philadelphia, 1989.

[Göd11] Göddeke, D., and Strzodka, R. Cyclic reduction tridiagonal solvers on GPUs applied to mixed precision multigrid. IEEE T. Parall. Distr., Special Issue: High Performance Computing with Accelerators, 22(1), pp. 22–32, 2011.

[Hel12] Helfenstein, R., and Koko, J. Parallel preconditioned conjugate gradient algorithm on GPU. J. Comput. Appl. Math., 236(15), pp. 3584–3590, 2012.

[Hoc65] Hockney, R. W. A fast direct solution of Poisson's equation using Fourier analysis. J. Assoc. Comput. Mach., 12(1), pp. 95–113, 1965.

[Hoc81] Hockney, R. W., and Jesshope, C. R. Parallel computers: architecture, programming and algorithms. Bristol, UK, 1981.

[Kim11] Kim, H.-S., Wu, S., Chang. L., and Hwu W. W. A scalable tridiagonal solver for GPUs. In: 42nd International Conference on Parallel Processing, pp. 444–453, IEEE Computer Society, Los Alamitos, CA, USA, 2011.

[Kuz85] Kuznetsov, Y. A. Numerical methods in subspaces. Vychislitel'-nye Processy i Sistemy II. 37, pp. 265–350, 1985.

[Kuz96] Kuznetsov, Yu. A., and Rossi, T. Fast direct method for solving algebraic systems with separable symmetric band matrices. East-West J. Numer. Math., 4, pp. 53–68, 1996.

[Lam12] Lamas-Rodriguez, J., Arguello, F., Heras, D.B., and Boo, M. Memory hierarchy optimization for large tridiagonal system solvers on GPU. In: IEEE 10th International Symposium on Parallel and Distributed Processing with Applications (ISPA), pp. 87–94, IEEE Press, Piscataway, NJ, USA, 2012.

[Lys04] Lysaker, M., Osher, S., Tai, X.-C. Noise removal using smoothed normals and surface fitting. IEEE T. Image Process., 13(10), pp. 1345 – 1357, 2004.

[Mum94] Mumford, D. Elastica and computer vision. Algebraic geometry and its applications, pp. 491–506, Springer-Verlag, New York, 1994.

[Myl13] Myllykoski, M., Rossi, T., and Toivanen, J. Fast Poisson solvers for graphics processing units. In: Applied Parallel and Scientific Computing, Manninen, P., and Öster, P. (eds.), Lecture Notes in Computer Science, 7782, pp. 265–279, 2013.

[Myl15] Myllykoski, M., Glowinski, R., Kärkkäinen, T., and Rossi, T. A new augmented Lagrangian approach for L^1-mean curvature image denoising. SIAM J. Imaging Sci., 8(1), pp. 95–125, 2015.

[Par92] Parker, A., and Hamblen, J.O. Optimal value for the Newton-Raphson division algorithm. Inform. Process. Lett., 42(3), pp. 141–144, 1992.

[Ros99] Rossi, T., and Toivanen T. A parallel fast direct solver for block tridiagonal systems with separable matrices of arbitrary dimension. SIAM J. Sci. Comput. 20(5), pp. 1778–1796, 1999.

[Rud92] Rudin, L., Osher, S., and Fatemi, E. Nonlinear total variation based noise removal algorithms. Phys. D, 60(1–4), pp. 259–268, 1992.

[Sun14] Sun, L., and Chen, K. A new iterative algorithm for mean curvature-based variational image denoising. BIT, 54(2), pp. 523–553, 2014.

[Vas84] Vassilevski, P. Fast algorithm for solving a linear algebraic problem with separable variables. C.R. Acad. Bulgare Sci., 37, pp. 305–308, 1984.

[Val85] Vassilevski, P. Fast algorithm for solving discrete Poisson equation in a rectangle. C.R. Acad. Bulgare Sci., 38, pp. 1311–1314, 1985.

[Yan14] Yangab, F., Chenc, K., Yub, B., Fang, D. A relaxed fixed point method for a mean curvature-based denoising model. Optim. Method Softw, 29(2), pp. 274 – 285, 2014.

[Zha10] Zhang, Y. and Cohen, J., and Owens, J. D. Fast tridiagonal solvers on the GPU. In: Proceedings of the 15th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming PPoPP 10, pp. 127–136, ACM Press, New York, NY, USA, 2010.

[Zhu12] Zhu, W., and Chan, T. Image denoising using mean curvature of image surface. SIAM J. Imaging Sci., 5(1), pp. 1–32, 2012.

[Zhu13] Zhu, W., Tai, X.-C., and Chan, T. Augmented Lagrangian method for a mean curvature based image denoising model. Inverse Probl., 7(4), pp. 1409–1432, 2013.