Jyväskylän yliopiston julkaisuarkisto
Jyväskylä University Digital Archive

UNIVERSITY OF JYVÄSKYLÄ

This is an electronic reprint of the original article.
This reprint *may differ* from the original in pagination and typographic detail.

Author(s): Myllykoski, Mirko; Rossi, Tuomo

Title: A parallel radix-4 block cyclic reduction algorithm

Year: 2014

Version:

Please cite the original version:

Myllykoski, M., & Rossi, T. (2014). A parallel radix-4 block cyclic reduction algorithm. Numerical Linear Algebra with Applications, 21(4), 540-556. https://doi.org/10.1002/nla.1909

All material supplied via JYX is protected by copyright and other intellectual property rights, and duplication or sale of all or part of any of the repository collections is not permitted, except that material may be duplicated by you for your research use or educational purposes in electronic or print form. You must obtain permission for any other use. Electronic or print copies may not be offered, whether for sale or otherwise to anyone who is not an authorised user.

A parallel radix-4 block cyclic reduction algorithm

M. Myllykoski[†¶] T. Rossi[†]

November 18, 2014

Abstract

A conventional block cyclic reduction algorithm operates by halving the size of the linear system at each reduction step, i.e., the algorithm is a radix-2 method. An algorithm analogous to the block cyclic reduction known as the radix-q PSCR method allows the use of higher radix numbers and is thus more suitable for parallel architectures as it requires fever reduction steps. This paper presents an alternative and more intuitive way of deriving a radix-4 block cyclic reduction method for systems with a coefficient matrix of the form tridiag$\{-I, D, -I\}$. This is done by modifying an existing radix-2 block cyclic reduction method. The resulting algorithm is then parallelized by using the partial fraction technique. The parallel variant is demonstrated to be less computationally expensive when compared to the radix-2 block cyclic reduction method in the sense that the total number of emerging sub-problems is reduced. The method is also shown to be numerically stable and equivalent to the radix-4 PSCR method. The numerical results archived correspond to the theoretical expectations.

keywords: block cyclic reduction; direct solver, fast Poisson solver; parallel computing; partial fraction technique; PSCR

This is the accepted version of the following article: Myllykoski, M. and Rossi, T. (2014), A parallel radix-4 block cyclic reduction algorithm. Numer. Linear Algebra Appl., 21: 540–556. doi: 10.1002/nla.1909, which has been published in final form at http://onlinelibrary.wiley.com/doi/10.1002/nla.1909/abstract.

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

[¶]Email: mirko.myllykoski@jyu.fi

1 Introduction

Linear system solvers of the block cyclic reduction (BCR) type have a long history starting from year 1965 [1]. The first formulation was found to be numerically unstable, but it was soon realized that the method can be stabilized by slightly modifying the formulation [2]. Another early attempt was to combine the method with the so-called Fourier analysis method [1]. This yielded the FACR(l) method [3, 4]. Later on, a more generalized BCR algorithm was also formulated [5] and, in order to produce a parallel variant, the partial fraction technique was applied to matrix rational polynomials occurring in the formulas [6]. The study of BCR type methods has generated a large amount literature dealing with different kind of variations, see e.g., [7, 8, 9, 10, 11, 12, 13, 14]. The convergence and stability of the method have also received a lot of attention [10, 15, 16, 17].

While the early formulations could only be applied to block tridiagonal systems and, preferably, to systems with a block Toeplitz structure, the method can also be formulated in such a way that it can be applied to a much wider class of problems. For example, a Schur complement type formulation can be applied to a class of Toeplitz-like block Hessenberg matrices arising in queuing problems [18, 19, 20]. Further developed BCR algorithms have been applied to a wide range of different kind of problems such as solving banded Toeplitz systems [21], factorizing matrix polynomial and power series [22, 23, 24], solving quadratic and nonlinear matrix equations [19, 20, 25], and solving algebraic Riccati equations [26, 27]. A recent and more detailed look to the cyclic reduction method and its applications can be found in [17].

In the 80's, another kind of direct method, which is nowadays called the radix-2 PSCR (Partial Solution variant of the Cyclic Reduction method), was formulated [28] and further developed in [29]. The PSCR method consists of two stages which are very similar to the reduction and back substitution stages in the BCR method. The main difference is that the PSCR method uses a so-called partial solution technique [30, 31] in order to reduce the system size. A more general radix-q algorithm was formulated later in [32]. Following the analogy between the PSCR and BCR methods, the radix number q defines the ratio according to which the size of the system is reduced at each reduction step. Thus, in the case of the conventional BCR algorithms the radix number is two. The PSCR method can be classified as a matrix decomposition algorithm and a good survey on those kind of methods can be found in [33].

In 1999, a more cyclic reduction type radix-2 PSCR formulation was introduced in [34]. In addition, the possibility of applying standard BCR techniques, such as simplifying the computations by using the rational polynomial factorization technique and parallelizing the recursion steps by applying the partial fraction technique [6], was considered. A stability estimate that was presented showed that the method is, with certain assumptions, linearly stable with respect to the size of the problem. From here onwards in this paper, the partial fraction variant of the method is referred to as the radix-2 PFCR (Partial Fraction variant of the Cyclic Reduction) method.

The radix-2 PFCR method has an interesting connection to the radix-2 PSCR method, as shown in [34], in the sense that, under certain special conditions, the radix-2 PSCR method produces exactly the same sub-problems as the radix-2 PFCR method. Thus, the radix-2 PFCR method can be thought

of as a special case of the radix-2 PSCR method. Correspondingly, as its name implies, the radix-4 PSCR method can be thought of as a radix-4 BCR method. However, the derivation of the radix-4 PSCR is somewhat more complicated than the cyclic reduction type formulation presented in [34].

The purpose of this paper is to present an alternative and more intuitive way of deriving a radix-4 BCR method for systems with a coefficient matrix of the form tridiag$\{-I, D, -I\}$. The radix-2 BCR method presented in [34] serves as a starting point and the partial fraction technique is applied to the modified formulas. The resulting parallel variant is referred from here on as the radix-4 PFCR method. The close connection between the radix-2 PFCR and PSCR methods suggests that the radix-4 PFCR method might have a similar connection to the radix-4 PSCR method. Thus, based on the computational complexity estimates presented in [35], it is expected that the total computational cost will decrease when the radix number is increased and the radix-4 PFCR method will require less sequential computation as the number of reduction steps is reduced by the factor of two. This would bring additional benefits to parallel architectures.

The rest of this paper is organized as follows: Section 2 describes a model problem to which methods may be applied; Section 3 provides an introduction to the radix-2 BCR algorithm and to the partial fraction technique; Section 4 contains the derivation of the radix-4 BCR algorithm and partial fraction expansions for rational polynomials occurring in the formulas; Section 5 provides an error analysis for the radix-4 BCR method; Section 6 shows the connection between the radix-4 PFCR and PSCR methods; Section 7 deals with numerical experiments; and Section 8 concludes the paper.

2 Problem formulation

This paper deals with the following block tridiagonal linear system

$$\begin{bmatrix} D & -I & & \\ -I & D & \ddots & \\ & \ddots & \ddots & -I \\ & & -I & D \end{bmatrix} \begin{bmatrix} u_1 \\ u_2 \\ \vdots \\ u_{n_1} \end{bmatrix} = \begin{bmatrix} f_1 \\ f_2 \\ \vdots \\ f_{n_1} \end{bmatrix}, \tag{1}$$

where $D \in \mathbb{R}^{n_2 \times n_2}$, $u_i, f_i \in \mathbb{R}^{n_2}$ and $n_1 = 2^k - 1$ for some positive integer k. Such systems arise, e.g., from finite difference discretization of a two-dimensional Poisson problem in a rectangle with homogeneous Dirichlet boundary conditions. More specifically, $D = \text{tridiag}\{-1, 4, -1\}$.

The final forms of the BCR methods presented in this paper generate a large set of sub-problems, each of which contains a linear system with a coefficient matrix of the form $D - cI$, where $c \in \mathbb{R}$. An important special case are problems where the diagonal block D is a block tridiagonal matrix of the form

$$D = \begin{bmatrix} B & -I & & \\ -I & B & \ddots & \\ & \ddots & \ddots & -I \\ & & -I & B \end{bmatrix} \in \mathbb{R}^{m_1 m_2 \times m_1 m_2}, \tag{2}$$

where $n_2 = m_1 m_2$, $B \in \mathbb{R}^{m_2 \times m_2}$ and $m_1 = 2^{\hat{k}} - 1$ for some positive integer $\hat{k}$. In this case, the methods can be applied recursively in order to solve the generated block tridiagonal sub-problems. For example, a three-dimensional Poisson problem with Dirichlet boundary conditions posed in a rectangular cuboid leads to a coefficient matrix where $B = \text{tridiag}\{-1, 6, -1\}$.

3 Radix-2 block cyclic reduction

The BCR methods are a well-known group of recursive algorithms for solving special kind of linear systems such as (1). On each recursion step, the algorithm eliminates all odd-numbered block rows from the linear system and creating thus a new linear system which is approximately half the size of the original linear system. This so-called reduction step is repeated recursively until the remaining linear systems consists only of one block equation. After this block equation is solved, the algorithm proceeds to a back substitution stage, during which the algorithm goes through all previously created subsystems in reverse order and solves all the odd-numbered block rows by using known even-numbered block rows from the previous back substitution step. In some cases, the subsystems converge in a certain sense and the BCR method can be applied as an iterative method. This section offers formulas for the radix-2 BCR algorithm closely following the presentation in [34].

3.1 Reduction Formulas

On the first reduction level, the reduction of the system (1) is performed by first multiplying the odd-numbered block rows with the matrix D^{-1} and then eliminating the rows from the system. This reduction pattern most likely appeared first in [10]. The remaining steps are defined as follows: Let $T^{(0)} = I$, $D^{(0)} = D$ and $f^{(0)} = f$. Now the subsystems are defined, for each reduction step $r = 1, 2, \ldots, k-1$, as

$$\begin{bmatrix} D^{(r)} & -T^{(r)} & & \\ -T^{(r)} & D^{(r)} & \ddots & \\ & \ddots & \ddots & -T^{(r)} \\ & & -T^{(r)} & D^{(r)} \end{bmatrix} \begin{bmatrix} u_1^{(r)} \\ u_2^{(r)} \\ \vdots \\ u_{2^{k-r}-1}^{(r)} \end{bmatrix} = \begin{bmatrix} f_1^{(r)} \\ f_2^{(r)} \\ \vdots \\ f_{2^{k-r}-1}^{(r)} \end{bmatrix}, \tag{3}$$

where

$$\begin{aligned} T^{(r)} &= \left(T^{(r-1)}\right)^2 \left(D^{(r-1)}\right)^{-1}, \\ D^{(r)} &= D^{(r-1)} - 2\left(T^{(r-1)}\right)^2 \left(D^{(r-1)}\right)^{-1}, \\ f_i^{(r)} &= f_{2i}^{(r-1)} + T^{(r-1)} \left(D^{(r-1)}\right)^{-1} \left(f_{2i-1}^{(r-1)} + f_{2i+1}^{(r-1)}\right). \end{aligned} \tag{4}$$

Note that the matrices $D^{(0)}, T^{(0)}, D^{(1)}, T^{(1)}, \ldots, D^{(k-1)}$ and $T^{(k-1)}$ commute.

The solution is produced during the back substitution stage of the algorithm

by the formula

$$u_i^{(r)} = \begin{cases} \left(D^{(r)}\right)^{-1} \left(f_i^{(r)} + T^{(r)} \left(u_{(i-1)/2}^{(r+1)} + u_{(i-1)/2+1}^{(r+1)}\right)\right), & \text{when } i \text{ is odd and} \\ u_{i/2}^{(r+1)}, & \text{when } i \text{ is even,} \end{cases} \tag{5}$$

where $r = k-1, k-2, \ldots, 0$, $i = 1, 2, \ldots, 2^{k-r}-1$ and $u_0^{(r+1)} = u_{2^{k-r-1}}^{(r+1)} = 0$. Finally, $u = u^{(0)}$. Note that the above formulas are well-defined only when the matrices $\left(D^{(r)}\right)^{-1}$, $r = 0, 1, \ldots, k-1$, exist. This is the case, for example, when the matrix D^{-1} exist and the coefficient matrix in the system (1) is strongly diagonally dominant by rows [10].

The above formulation has some disadvantages, the most significant of which is that the matrices $D^{(r)}$ and $T^{(r)}$ can fill quickly even if the matrix D is sparse. This makes the matrix calculation expensive and requires additional memory to store the matrices. However, as noted in [34], this problem can be easily solved by expressing the matrices $D^{(r)}$ and $T^{(r)}$ as

$$D^{(r)} = \beta^{(r)}(D)\alpha^{(r)}(D) \quad \text{and} \quad T^{(r)} = \beta^{(r)}(D), \tag{6}$$

where the matrix polynomials $\alpha^{(r)}(D)$ and $\left(\beta^{(r)}(D)\right)^{-1}$ are defined recursively by starting with $\alpha^{(0)}(D) = D$, $\beta^{(0)}(D) = I$ and then defining

$$\begin{aligned} \alpha^{(r)}(D) &= \left(\alpha^{(r-1)}(D)\right)^2 - 2I, \\ \beta^{(r)}(D) &= \beta^{(r-1)}(D) \left(\alpha^{(r-1)}(D)\right)^{-1}. \end{aligned} \tag{7}$$

The roots of the resulting polynomials are known in a closed form. Hence, they can be factorized as:

$$\begin{aligned} \alpha^{(r)}(D) &= \prod_{j=1}^{2^r} \left(D - \theta(j,r)I\right), \quad \theta(j,r) = 2\cos\left(\frac{2j-1}{2^{r+1}}\pi\right), \\ \beta^{(r)}(D) &= \prod_{j=1}^{2^r-1} \left(D - \phi(j,r)I\right)^{-1}, \quad \phi(j,r) = 2\cos\left(\frac{j}{2^r}\pi\right). \end{aligned} \tag{8}$$

The use of these factorized forms of the polynomials reduces the amount of the required memory and they also considerably simplify the calculations in formulas (4) and (5). Solving a sequence of linear systems like (8) is often considerably easier than solving a single dense linear system. The polynomial factorizations presented above are not however the most optimal ones for parallel computing, as the sub-problems must be solved sequentially, one after another. More possibilities for parallel execution can be obtained by the partial fraction technique. In addition, a direct substitution of the factorizations (8) into the formulas (4) and (5) could lead to numerical instability [15]. For a stable way of performing the computations and for a more general formulation where the off-diagonal blocks in the system (1) do not have to be identity matrices, see [34].

3.2 Partial Fraction Technique

The matrix polynomials $\alpha^{(r)}(D)$ and $\left(\beta^{(r)}(D)\right)^{-1}$ are actually matrix counterparts of $C_{2^r}(x)$ and $S_{2^r-1}(x)$ respectively, defined as

$$\begin{aligned} C_{2^r}(x) &= \begin{cases} 2\cos\left(2^r \cos^{-1}(x/2)\right), & 0 \le x \le 2, \\ 2\cosh\left(2^r \cosh^{-1}(x/2)\right), & x > 2, \end{cases} \\ S_{2^r-1}(x) &= \begin{cases} \sin\left(2^r \cos^{-1}(x/2)\right) / \sin\left(\cos^{-1}(x/2)\right), & 0 \le x \le 2, \\ \sinh\left(2^r \cosh^{-1}(x/2)\right) / \sinh\left(\cosh^{-1}(x/2)\right), & x > 2. \end{cases} \end{aligned} \tag{9}$$

The above polynomials are modified Chebyshev polynomials and they have the following property:

$$C'_{2^r}(x) = 2^r S_{2^r-1}(x). \tag{10}$$

These observations make it easier to apply the partial fraction technique introduced in [6]. The basic idea is given by the following lemma:

Lemma 1 *Let $p(x)$ and $q(x)$ be two polynomials with the following properties:*

(i) p and q are relatively prime,

(ii) $\deg\ p < \deg\ q = n$, and

(iii) the roots, $\alpha_1, \alpha_2, \ldots, \alpha_n$, of q are distinct.

Then, the following statement applies:

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

Applying Lemma 1 leads to additive expressions for the matrices arising in the algorithm [34]. In particular

$$T^{(r)}\left(D^{(r)}\right)^{-1} = 2^{-r} \sum_{j=1}^{2^r} (-1)^{j-1} \sin\left(\frac{2j-1}{2^{r+1}}\pi\right) (D - \theta(j,r)I)^{-1} \tag{11}$$

and

$$\left(D^{(r)}\right)^{-1} = 2^{-r} \sum_{j=1}^{2^r} (D - \theta(j,r)I)^{-1}, \tag{12}$$

where $\theta(i,r)$ is defined in (8). This means that instead of solving a sequence of sub-problems sequentially, the solution is formed by solving a set of sub-problems (in parallel) and then computing the final result as a collective sum.

On each reduction step, the algorithm operates $2^{k-r} - 1$ block rows, each of which requires solution of 2^{r-1} sub-problems. During the back substitution stage the algorithm operates 2^{k-r-1} block rows per step, each of which requires solution of 2^r sub-problems. Adding all of these together gives a total of

$$N^2_{\text{count}}(k) = 2^k (k-1) + 1, \tag{13}$$

sub-problems during the execution of the algorithm.

4 Derivation of the radix-4 block cyclic reduction method

A major down side of the radix-2 PFCR method is that the recursion steps are dependent on the preceding steps. The impact of this limitation could be reduced by increasing the radix number. This section covers a new way of deriving a radix-4 BCR method. This is done by combining two consecutive radix-2 reduction steps (4) into a radix-4 reduction step and then applying the partial fraction technique.

4.1 Reduction Formulas

Let $n_1 = 4^k - 1$ for some positive integer k. By combining radix-2 reduction steps (4) the following reduction formula is obtained

$$\begin{aligned} f_i^{(2r)} &= f_{2i}^{(2r-1)} + T^{(2r-1)} \left(D^{(2r-1)}\right)^{-1} \left(f_{2i-1}^{(2r-1)} + f_{2i+1}^{(2r-1)}\right) \\ &= f_{4i}^{(2r-2)} + T^{(2r-2)} \left(D^{(2r-2)}\right)^{-1} \left(f_{4i-1}^{(2r-2)} + f_{4i+1}^{(2r-2)}\right) + \\ &\quad T^{(2r-1)} \left(D^{(2r-1)}\right)^{-1} \left(f_{4i-2}^{(2r-2)} + f_{4i+2}^{(2r-2)}\right) + \\ &\quad T^{(2r-1)} \left(D^{(2r-1)}\right)^{-1} T^{(2r-2)} \left(D^{(2r-2)}\right)^{-1} \cdot \\ &\quad \left(f_{4i-3}^{(2r-2)} + f_{4i-1}^{(2r-2)} + f_{4i+1}^{(2r-2)} + f_{4i+3}^{(2r-2)}\right), \end{aligned} \tag{14}$$

where $r = 1, 2, \ldots, k-1$ and $i = 1, 2, \ldots, 4^{k-r} - 1$.

The back substitution step is slightly more complicated. For reasons that will become apparent later, the best approach is to solve the block rows in groups of four. Let $r = k-1, k-2, \ldots, 0$ and $d = 0, 1, \ldots, 4^{k-r-1} - 1$. For the first row of the group, the combining of two radix-2 back substitution steps (5) leads to

$$\begin{aligned} u_{4d+1}^{(2r)} &= \left(D^{(2r)}\right)^{-1} \left(f_{4d+1}^{(2r)} + T^{(2r)} \left(u_{2d}^{(2r+1)} + u_{2d+1}^{(2r+1)}\right)\right) \\ &= \left(D^{(2r)}\right)^{-1} \left[f_{4d+1}^{(2r)} + T^{(2r)} \left(u_d^{(2r+2)} + \right.\right. \\ &\quad \left.\left. \left(D^{(2r+1)}\right)^{-1} \left(f_{2d+1}^{(2r+1)} + T^{(2r+1)} \left(u_d^{(2r+2)} + u_{d+1}^{(2r+2)}\right)\right)\right)\right]. \end{aligned} \tag{15}$$

The unknown term $f_{2d+1}^{(2r+1)}$ from the intermediate step $2r+1$ can be resolved by substituting it with a radix-2 reduction formula (4). The end result is

$$\begin{aligned} u_{4d+1}^{(2r)} &= \left(D^{(2r)}\right)^{-1} f_{4d+1}^{(2r)} + \left(D^{(2r)}\right)^{-1} T^{(2r)} u_d^{(2r+2)} + \\ &\quad \left(D^{(2r)}\right)^{-1} T^{(2r)} \left(D^{(2r+1)}\right)^{-1} f_{4d+2}^{(2r)} + \\ &\quad \left(D^{(2r)}\right)^{-1} T^{(2r)} \left(D^{(2r+1)}\right)^{-1} T^{(2r)} \left(D^{(2r)}\right)^{-1} \left(f_{4d+1}^{(2r)} + f_{4d+3}^{(2r)}\right) + \\ &\quad \left(D^{(2r)}\right)^{-1} T^{(2r)} \left(D^{(2r+1)}\right)^{-1} T^{(2r+1)} \left(u_d^{(2r+2)} + u_{d+1}^{(2r+2)}\right). \end{aligned} \tag{16}$$

In a similar manner, for the second block row

$$\begin{aligned} u_{4d+2}^{(2r)} &= u_{2d+1}^{(2r+1)} \\ &= \left(D^{(2r+1)}\right)^{-1}\left(f_{2d+1}^{(2r+1)} + T^{(2r+1)}\left(u_d^{(2r+2)} + u_{d+1}^{(2r+2)}\right)\right) \\ &= \left(D^{(2r+1)}\right)^{-1} f_{4d+2}^{(2r)} + \left(D^{(2r+1)}\right)^{-1} T^{(2r)}\left(D^{(2r)}\right)^{-1}\left(f_{4d+1}^{(2r)} + f_{4d+3}^{(2r)}\right) + \\ &\quad \left(D^{(2r+1)}\right)^{-1} T^{(2r+1)}\left(u_d^{(2r+2)} + u_{d+1}^{(2r+2)}\right), \end{aligned} \tag{17}$$

and for the third block row

$$\begin{aligned} u_{4d+3}^{(2r)} = &\left(D^{(2r)}\right)^{-1} f_{4d+3}^{(2r)} + \left(D^{(2r)}\right)^{-1} T^{(2r)} u_{d+1}^{(2r+2)} + \\ &\left(D^{(2r)}\right)^{-1} T^{(2r)}\left(D^{(2r+1)}\right)^{-1} f_{4d+2}^{(2r)} + \\ &\left(D^{(2r)}\right)^{-1} T^{(2r)}\left(D^{(2r+1)}\right)^{-1} T^{(2r)}\left(D^{(2r)}\right)^{-1}\left(f_{4d+1}^{(2r)} + f_{4d+3}^{(2r)}\right) + \\ &\left(D^{(2r)}\right)^{-1} T^{(2r)}\left(D^{(2r+1)}\right)^{-1} T^{(2r+1)}\left(u_d^{(2r+2)} + u_{d+1}^{(2r+2)}\right). \end{aligned} \tag{18}$$

And finally, if $d \neq 4^{k-r-1} - 1$, then for the fourth block row $u_{4d+4}^{(2r)} = u_{d+1}^{(r+2)}$.

At this point, it is possible to see why it is feasible to perform the back substitution step in this particular way. The formulas (16) and (18) have two common terms, and thus some intermediate results can be shared between these two block rows. Later on, when the partial fraction technique is applied to the matrix rational polynomials appearing in these three back substitution formulas, the number of common terms increases. As a consequence the cost of performing the radix-4 back substitution step is actually almost identical to the cost of performing the radix-4 reduction step (14).

The above formulation can be modified so that it is capable of solving problems of the size $n_1 = 2^k - 1$. The reduction stage formula (26) does not require any major modification. The indexes r and i are modified in the following manner: $r = 1, 2, \ldots, \lceil k/2 \rceil - 1$ and $i = 1, 2, \ldots, 2^{k-2r}$. However, the back substitution stage requires more modification. Firstly, one radix-2 back substitution step (5) is performed at the radix-2 level $r = k - 1$ in order to solve the block row $u_{2^{k-1}}$. Then the radix-4 back substitution stage can begin, but the indexes r and d are changed in the following manner: $r = \lfloor k/2 \rfloor - 1, \lfloor k/2 \rfloor - 2, \ldots, 0$ and $d = 0, 1, \ldots, 2^{k-2r-2}$.

It is possible to directly substitute the factorizations (8) into the formulas. However, as was noted in the case of the radix-2 BCR method, this substitution could lead to numerical instability. Also, the resulting radix-4 formulation would not have the desired benefits over the corresponding radix-2 formulation. Hence, the idea of applying the factorizations (8) directly is not discussed in this paper in further detail.

4.2 Partial Fractions

In this section, the partial fraction technique is applied to the matrices occurring in the radix-4 formulas (14), (16), (17) and (18). The matrix $B_1^{(r)} =$

$T^{(r)}\left(D^{(r)}\right)^{-1}T^{(r-1)}\left(D^{(r-1)}\right)^{-1}$ can be expressed, using (6) and (7), as $\left(\alpha^{(r)}(D)\alpha^{(r-1)}(D)\right)^{-1}$. This means that in Lemma 1, $p(x)=1$ and

$$q(x)=C_{2^r}(x)C_{2^{r-1}}(x). \tag{19}$$

Using Lemma 1 and relation (10) is now slightly more complicated because the polynomial $q(x)$ has two distinct sets of roots. This results in the following expressions for the coefficients c_j in Lemma 1:

$$\begin{aligned} c_j &= \frac{1}{q'(\theta(j,r))} = 2^{-r}\sin\left(\frac{2j-1}{4}\pi\right)\sin\left(\frac{2j-1}{2^{r+1}}\pi\right),\ j=1,2,\ldots,2^r,\\ c_{2^r+j} &= \frac{1}{q'(\theta(j,r-1))} = -2^{-r}\sin\left(\frac{2j-1}{2}\pi\right)\sin\left(\frac{2j-1}{2^r}\pi\right),\ j=1,2,\ldots,2^{r-1}. \end{aligned} \tag{20}$$

Thus the matrix $B_1^{(r)}$ has the following partial fraction:

$$\begin{aligned} B_1^{(r)} = 2^{-r}\sum_{j=1}^{2^r}\sin\left(\frac{2j-1}{4}\pi\right)\sin\left(\frac{2j-1}{2^{r+1}}\pi\right)(D-\theta(j,r)I)^{-1} - \\ 2^{-r}\sum_{j=1}^{2^{r-1}}(-1)^{j-1}\sin\left(\frac{2j-1}{2^r}\pi\right)(D-\theta(j,r-1)I)^{-1}. \end{aligned} \tag{21}$$

Another set of equations involve the matrix $B_2^{(r)}$, which is of the form

$$B_2^{(r)} = \left(D^{(r-1)}\right)^{-1}T^{(r-1)}\left(D^{(r)}\right)^{-1} = \left(\beta^{(r-1)}(D)\alpha^{(r)}(D)\right)^{-1}. \tag{22}$$

Now, $p(x)=S_{2^{r-1}-1}(x)$ and $q(x)=C_{2^r}(x)$. Thus, Lemma 1 leads to the expansion

$$B_2^{(r)} = 2^{-r}\sum_{j=1}^{2^r}(-1)^{j-1}\sin\left(\frac{2j-1}{4}\pi\right)(D-\theta(j,r)I)^{-1}. \tag{23}$$

The third set of equations has coefficient matrices $B_3^{(r)}$ having the structure

$$\begin{aligned} B_3^{(r)} &= \left(D^{(r-1)}\right)^{-1}T^{(r-1)}\left(D^{(r)}\right)^{-1}T^{(r-1)}\left(D^{(r-1)}\right)^{-1}\\ &= \left(\beta^{(r-1)}(D)\alpha^{(r)}(D)\alpha^{(r-1)}(D)\right)^{-1} \end{aligned} \tag{24}$$

In this case also, the denominator $q(x)=C_{2^r}(x)C_{2^{r-1}}(x)$ in Lemma 1 has two distinct sets of roots and thus

$$B_3^{(r)} = 2^{-r-1}\sum_{j=1}^{2^r}(D-\theta(j,r)I)^{-1} + 2^{-r}\sum_{j=1}^{2^{r-1}}(D-\theta(j,r-1)I)^{-1}. \tag{25}$$

4.3 Final Formulas

Now the radix-4 reduction stage can be rewritten by using the results of the previous subsection as follows: Let $n_1 = 4^k - 1$ for some positive integer k and $f^{(0)} = f$. Then the reduction steps $r = 1, 2, \ldots, k-1$ are

$$\begin{aligned} f_i^{(r)} = f_{4i}^{(r-1)} + 2^{1-2r} \sum_{j=1}^{2^{2r-1}} \sin\left(\frac{2j-1}{2^{2r}}\pi\right) (D - \theta(j, 2r-1)I)^{-1} \\ \left[(-1)^{j-1}\left(f_{4i-2}^{(r-1)} + f_{4i+2}^{(r-1)}\right) + \right. \\ \left. \sin\left(\frac{2j-1}{4}\pi\right)\left(f_{4i-3}^{(r-1)} + f_{4i-1}^{(r-1)} + f_{4i+1}^{(r-1)} + f_{4i+3}^{(r-1)}\right)\right] + \\ 2^{1-2r} \sum_{j=1}^{2^{2r-2}} (-1)^{j-1} \sin\left(\frac{2j-1}{2^{2r-1}}\pi\right) (D - \theta(j, 2r-2)I)^{-1} \cdot \\ \left(-f_{4i-3}^{(r-1)} + f_{4i-1}^{(r-1)} + f_{4i+1}^{(r-1)} - f_{4i+3}^{(r-1)}\right), \end{aligned} \tag{26}$$

where $i = 1, 2, \ldots, 4^{k-r} - 1$.

The same procedure can be performed for the back substitution stage formulas (16), (17) and (18). Let $r = k-1, k-2, \ldots, 0$ and $d = 0, 1, \ldots, 4^{k-r-1} - 1$. First, it is necessary to define vectors

$$\begin{aligned} g_{d,j}^{(r)} = (-1)^{j-1} f_{4d+2}^{(r)} + \sin\left(\frac{2j-1}{4}\pi\right)\left(f_{4d+1}^{(r)} + f_{4d+3}^{(r)}\right) + \\ \sin\left(\frac{2j-1}{2^{2r+2}}\right)\left(u_d^{(r+1)} + u_{d+1}^{(r+1)}\right), \\ h_{d,j}^{(r)} = (-1)^{j-1}\left(f_{4d+1}^{(r)} - f_{4d+3}^{(r)}\right) + \sin\left(\frac{2j-1}{2^{2r+1}}\right)\left(u_d^{(r+1)} - u_{d+1}^{(r+1)}\right), \end{aligned} \tag{27}$$

where $u_0^{(r+1)} = u_{4^{k-r-1}}^{(r+1)} = 0$. If $d \neq 4^{k-r-1} - 1$, then $u_{4d+4}^{(r)} = u_{d+1}^{(r+1)}$. The other components can be solved from

$$\begin{aligned} u_{4d+1}^{(r)} &= 2^{-2r-1} \sum_{j=1}^{2^{2r+1}} \sin\left(\frac{2j-1}{4}\pi\right) v_{d,j}^{(r)} + 2^{-2r-1} \sum_{j=1}^{2^{2r}} (-1)^{j-1} y_{d,j}^{(r)}, \\ u_{4d+2}^{(r)} &= 2^{-2r-1} \sum_{j=1}^{2^{2r+1}} (-1)^{j-1} v_{d,j}^{(r)}, \\ u_{4d+3}^{(r)} &= 2^{-2r-1} \sum_{j=1}^{2^{2r+1}} \sin\left(\frac{2j-1}{4}\pi\right) v_{d,j}^{(r)} - 2^{-2r-1} \sum_{j=1}^{2^{2r}} (-1)^{j-1} y_{d,j}^{(r)}, \end{aligned} \tag{28}$$

where

$$\begin{aligned} v_{d,j}^{(r)} &= (D - \theta(j, 2r+1)I)^{-1} g_{d,j,}^{(r)} \\ y_{d,j}^{(r)} &= (D - \theta(j, 2r)I)^{-1} h_{d,j}^{(r)}. \end{aligned} \tag{29}$$

Finally $u = u^{(0)}$.

On each reduction step (26) the algorithm handles $4^{k-r}-1$ block rows, each of which requires the solution of $3\cdot 2^{2r-2}$ sub-problems. During the back substitution stage (28) the algorithm handles 4^{k-r-1} groups per step, each of which requires the solution of $3\cdot 2^{2r}$ sub-problems. This amounts to a total of

$$N^4_{\text{count}}(k) = 2^{2k-1}(3k-2)+1, \tag{30}$$

sub-problems during the algorithm. Thus, the radix-4 PFCR method produces fewer sub-problems that the radix-2 PFCR method and

$$\lim_{k\to\infty}\frac{N^2_{\text{count}}(2k)}{N^4_{\text{count}}(k)} = \frac{4}{3}. \tag{31}$$

If the matrix D is block tridiagonal as was discussed in Section 2, then these same methods can be applied to sub-problems occurring in formulas (26) and (28). As a result,

$$\lim_{k\to\infty}\left(\frac{N^2_{\text{count}}(2k)}{N^4_{\text{count}}(k)}\right)^2 = \frac{16}{9}. \tag{32}$$

5 Error analysis

An error estimate [34] shows, by using similar techniques as in [36], that the radix-2 BCR method is linearly stable with respect to the size of the problem when $D = D^T$ and the smallest eigenvalue of the matrix D is at least two. This section extends this technique to the radix-4 BCR method.

Let $\lambda(D)$ be the spectrum of the matrix D. The reference [34] provides the following estimates for the spectral norms of the matrices $\left(D^{(r)}\right)^{-1}$ and $T^{(r)}\left(D^{(r)}\right)^{-1}$ when $D = D^T$ and $\lambda \geq 2$ for all $\lambda \in \lambda(D)$:

$$\begin{aligned}\left\|\left(D^{(r)}\right)^{-1}\right\| &\leq 2^{r-1},\\ \left\|T^{(r)}\left(D^{(r)}\right)^{-1}\right\| &\leq \frac{1}{2}.\end{aligned} \tag{33}$$

Since the spectral norm is sub-multiplicative, these lead to:

$$\begin{aligned}\left\|T^{(r-1)}\left(D^{(r-1)}\right)^{-1}T^{(r)}\left(D^{(r)}\right)^{-1}\right\| &\leq \frac{1}{4},\\ \left\|\left(D^{(r-1)}\right)^{-1}T^{(r-1)}\left(D^{(r)}\right)^{-1}\right\| &\leq 2^{r-2},\\ \left\|\left(D^{(r-1)}\right)^{-1}T^{(r-1)}\left(D^{(r)}\right)^{-1}T^{(r-1)}\left(D^{(r-1)}\right)^{-1}\right\| &\leq 2^{r-3}.\end{aligned} \tag{34}$$

Let $n_1 = 4^k - 1$, $k \geq 2$, $f_1, f_2, \ldots, f_{n_1}$ denote the exact right-hand side vector blocks, $f_{i,\varepsilon}$ be the floating point counterpart of f_i and $\varepsilon \geq 0$ be selected in such a way that

$$\|f_i - f_{i,\varepsilon}\|_2 \leq \varepsilon,$$

for all $i = 1, 2, \ldots, n_1$. And let $f_i^{(r)}$, $r = 0, 1, \ldots, k-1$, $i = 1, 2, \ldots, 4^{k-r} - 1$, denote the right hand side vector blocks produced by the radix-4 BCR method using exact arithmetic and $f_{i,\varepsilon}^{(r)}$ be the floating point counterpart of $f_i^{(r)}$. Substituting the spectral norm estimates (33) and (34) into the formula (14) gives the following error estimates for the reduction stage

$$\begin{aligned}
\left\| f_i^{(0)} - f_{i,\varepsilon}^{(0)} \right\|_2 &\leq \varepsilon = g^{(0)}(\varepsilon, \delta), \\
\left\| f_i^{(r)} - f_{i,\varepsilon}^{(r)} \right\|_2 &\leq g^{(r-1)}(\varepsilon, \delta) + 1/2 \cdot 2 \cdot g^{(r-1)}(\varepsilon, \delta) + \\
&\qquad 1/2 \cdot 2 \cdot g^{(r-1)}(\varepsilon, \delta) + 1/4 \cdot 4 \cdot g^{(r-1)}(\varepsilon, \delta) + \delta \\
&= 4^r \varepsilon + 1/3 \cdot (4^r - 1)\, \delta = g^{(r)}(\varepsilon, \delta),
\end{aligned} \tag{35}$$

where $\delta \geq 0$ denotes the upper limit for the roundoff error introduced at each step.

A similar analysis can be performed for the back substitution stage. Let $u_i^{(r)}$, $r = 0, 1, \ldots, k-1$, $i = 1, 2, \ldots, 4^{k-r} - 1$, denote the solution vector blocks produced by the radix-4 BCR method using exact arithmetic and $u_{i,\varepsilon}^{(r)}$ be the floating point counterpart of $u_i^{(k)}$. For the level $r = k-1, k-2, \ldots, 0$, using the formulas (16), (17) and (18) leads to

$$\begin{aligned}
\left\| u_{4d+1}^{(r)} - u_{4d+1,\varepsilon}^{(r)} \right\|_2 &\leq 3 \cdot 2^{2r-1} g^{(r)}(\varepsilon, \delta) + \mu^{(r+1)}(\varepsilon, \delta) + \delta < \mu^{(r)}(\varepsilon, \delta), \\
\left\| u_{4d+2}^{(r)} - u_{4d+2,\varepsilon}^{(r)} \right\|_2 &\leq 2^{2r+1} g^{(r)}(\varepsilon, \delta) + \mu^{(r+1)}(\varepsilon, \delta) + \delta = \mu^{(r)}(\varepsilon, \delta), \\
\left\| u_{4d+3}^{(r)} - u_{4d+3,\varepsilon}^{(r)} \right\|_2 &\leq 3 \cdot 2^{2r-1} g^{(r)}(\varepsilon, \delta) + \mu^{(r+1)}(\varepsilon, \delta) + \delta < \mu^{(r)}(\varepsilon, \delta) \\
\left\| u_{4d+4}^{(r)} - u_{4d+4,\varepsilon}^{(r)} \right\|_2 &\leq \mu^{(r+1)}(\varepsilon, \delta) < \mu^{(r)}(\varepsilon, \delta),
\end{aligned} \tag{36}$$

where $d = 0, 1, \ldots, 4^{k-r-1} - 1$ and $\mu^{(k)}(\varepsilon, \delta) = 0$.

At the final back substitution step ($r = 0$), the accumulative error is

$$\mu^{(0)}(\varepsilon, \delta) = \left[\sum_{r=0}^{k-1} 2^{2r+1} \left(4^r \varepsilon + 1/3 \cdot (4^r - 1)\, \delta \right) \right] + k\delta. \tag{37}$$

This leads to the following result

$$\| u_i - u_{i,\varepsilon} \|_2 \leq \mu^{(0)}(\varepsilon, \delta) \leq \frac{2}{15} (n_1 + 1)^2 \left(\varepsilon + \delta \right), \tag{38}$$

for all $i = 1, 2, \ldots, n_1$. Thus it can be concluded that, the radix-4 BCR method is linearly stable with respect to the size of the problem when $n_1 \approx n_2$. The numerical results shown in Figure 1 support this conclusion.

6 Connection to the Radix-4 PSCR method

As mentioned in [34], the radix-2 PFCR and PSCR methods are equivalent in the sense that both methods generate exactly the same sub-problems when applied to a linear systems of the form (1). The first subsection describes a simplified radix-q PSCR method and the second subsection shows that the radix-4 PFCR method has a similar connection to the radix-4 PSCR method.

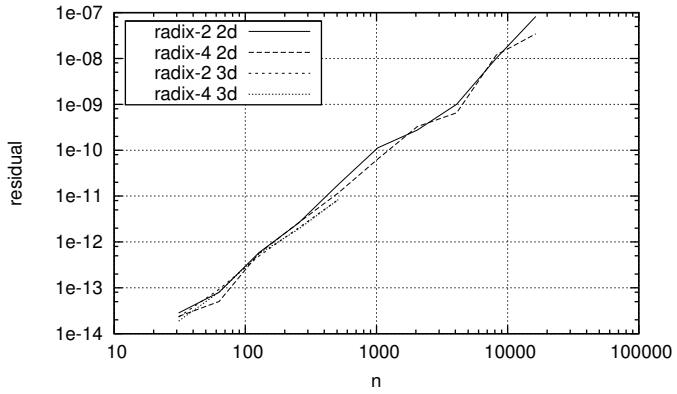

Figure 1: Measured error residuals for the radix-2 and radix-4 PFCR methods when using blockwise euclidean vector norm and double-precision floating-point arithmetic. The number of unknowns is n^2 for two-dimensional problems and n^3 for three-dimensional problems. $D = \text{tridiag}\{-1, 4, -1\}$ (or $B = \text{tridiag}\{-1, 6, -1\}$) and the elements of the right-hand side vector are randomly distributed on the interval $[-1, 1]$.

6.1 Simplified Radix-q PSCR Method

The PSCR method can be applied, under certain assumptions, to block tridiagonal linear systems of the form

$$Au = f, \quad A = A_1 \otimes M_2 + M_1 \otimes A_2 + c(M_1 \otimes M_2), \tag{39}$$

where $A_1, M_1 \in \mathbb{R}^{n_1 \times n_1}, A_2, M_2 \in \mathbb{R}^{n_2 \times n_2}, u, f \in \mathbb{R}^{n_1 n_2}, c \in \mathbb{R}$ and $\otimes$ denotes the matrix Kronecker (tensor) product[1]. The PSCR method includes two stages which are very similar to the reduction and back substitution stages in the BCR methods, and an initialization stage comprised of generalized eigenvalue problems.

If $n_1 = q^k - 1$ for some positive integers q and k, $2 \leq q$, and the matrix A is of the form

$$\tilde{A} \otimes I + I \otimes (D - 2I), \tag{40}$$

where $\tilde{A} = \text{tridiag}\{-1, 2, -1\} \in \mathbb{R}^{n_1 \times n_1}$, then the matrix A correspond to the coefficient matrix in the system (1) and the generalized eigenvalue problems reduce into

$$\tilde{A}^{(r)} w_j^{(r)} = \lambda_j^{(r)} w_j^{(r)}, \; j = 1, 2, \ldots, m_r, \tag{41}$$

where $m_r = q^{r+1} - 1$ and $\tilde{A}^{(r)} = \text{tridiag}\{-1, 2, -1\} \in \mathbb{R}^{m_r \times m_r}$.

The radix-q PSCR method can be described by using the same notation as was used in Sections 2 and 4. Let $f^{(0)} = f$. At first, for $r = 1, 2, \ldots, k-1$, a sequence of vectors is generated by using the formula

$$f_i^{(r)} = f_{qi}^{(r-1)} + \sum_{j=1}^{m_{r-1}} (w_j^{(r-1)})_{m_{r-1}} v_{i,j}^{(r-1)} + \sum_{j=1}^{m_{r-1}} (w_j^{(r-1)})_1 v_{i+1,j}^{(r-1)}, \tag{42}$$

[1] If $A \in \mathbb{R}^{n \times n}$ and $B \in \mathbb{R}^{m \times m}$, then $A \otimes B = \{A_{i,j} B\}_{i,j=1}^{n} \in \mathbb{R}^{nm \times nm}$.

where $i = 1, 2, \ldots, q^{k-r} - 1$ and the vector $v_{i,j}^{(r-1)}$ can be solved from

$$\left(D + (\lambda_j^{(r-1)} - 2)I\right) v_{i,j}^{(r-1)} = \sum_{s=1}^{q-1} (w_j^{(r-1)})_{sq^{r-1}} f_{(i-1)q+s}^{(r-1)}. \tag{43}$$

This operation includes so called partial solutions [30, 31], where the right-hand side vector is sparse and only certain components from the solution vector are needed, namely the first and the last component in the present case. Next, for $r = k-1, k-2, \ldots, 0$, a second sequence of vectors is generated by using the formula

$$\begin{aligned} u_{qd+i}^{(r)} &= \sum_{j=1}^{m_r} (w_j^{(r)})_{iq^r} y_{d,j}^{(r)}, \ i = 1, 2, \ldots, q-1, \\ u_{qd+q}^{(r)} &= u_{d+1}^{(r+1)}, \ d \neq q^{k-r-1} - 1, \end{aligned} \tag{44}$$

where $d = 0, 1, \ldots, q^{k-r-1} - 1$ and the vector $y_{d,j}^{(r)}$ can be solved from

$$\begin{aligned} \left(D + (\lambda_j^{(r)} - 2)I\right) y_{d,j}^{(r)} = \sum_{s=1}^{q-1} (w_j^{(r)})_{sq^r} f_{qd+s}^{(r)} + \\ (w_j^{(r)})_1 u_d^{(r+1)} + (w_j^{(r)})_{m_r} u_{d+1}^{(r+1)}. \end{aligned} \tag{45}$$

In addition, $u_0^{(r+1)} = u_{k-r-1}^{(r+1)} = 0$. Finally, $u = u^{(0)}$.

6.2 Connection to the Radix-4 PFCR Method

It is known that the matrix $\tilde{A}^{(r)}$ has the following eigenvalues and eigenvectors

$$\lambda_i^{(r)} = 2 - 2\cos\left(\frac{i\pi}{q^{r+1}}\right) \text{ and } (w_i^{(r)})_j = \sqrt{2 \cdot q^{-r-1}} \sin\left(\frac{ij\pi}{q^{r+1}}\right), \tag{46}$$

where $i, j = 1, 2, \ldots, m_r$. Let $q = 4$. Now

$$\begin{aligned} (w_j^{(r)})_1 &= \sqrt{2^{-2r-1}} \sin\left(j\pi/4^{r+1}\right) = (-1)^{j-1} (w_j^{(r)})_{m_r}, \\ (w_j^{(r)})_{1 \cdot 4^r} &= \sqrt{2^{-2r-1}} \sin\left(j\pi/4\right) = (-1)^{j-1} (w_j^{(r)})_{3 \cdot 4^r}, \\ (w_j^{(r)})_{2 \cdot 4^r} &= \sqrt{2^{-2r-1}} \sin\left(j\pi/2\right). \end{aligned} \tag{47}$$

It is easy to see that $(w_j^{(r)})_{2 \cdot 4^r} = 0$ when $j \in 2\mathbb{N}$ and $(w_j^{(r)})_{1 \cdot 4^r} = 0 = (w_j^{(r)})_{3 \cdot 4^r}$ when $j \in 4\mathbb{N}$. By taking this into account, the formulas (42) and (43) can be rewritten as

$$\begin{aligned} f_i^{(r)} = f_{4i}^{(r-1)} + \sum_{j=1}^{2^{2r-1}} \sqrt{2^{1-2r}} \sin\left(\frac{2j-1}{2^{2r}}\pi\right) v_{i,2j-1}^{(r)} + \\ \sum_{j=1}^{2^{2r-2}} \sqrt{2^{1-2r}} \sin\left(\frac{2j-1}{2^{2r-1}}\pi\right) v_{i,4j-2}^{(r)} \end{aligned} \tag{48}$$

and

$$\begin{aligned}\left(D-2\cos\left(j\pi/2^{2r}\right)I\right)v_{i,j}^{(r)} &= \sqrt{2^{1-2r}}\sin\left(j\pi/4\right)\left(f_{4i-3}^{(r-1)}+(-1)^{j-1}f_{4i+1}^{(r-1)}\right)+\\ &\sqrt{2^{1-2r}}\sin\left(j\pi/2\right)\left(f_{4i-2}^{(r-1)}+(-1)^{j-1}f_{4i+2}^{(r-1)}\right)+\\ &\sqrt{2^{1-2r}}\sin\left(j\pi/4\right)\left((-1)^{j-1}f_{4i-1}^{(r-1)}+f_{4i+3}^{(r-1)}\right).\end{aligned} \tag{49}$$

Thus the first stage of the radix-4 PSCR method is equivalent with the radix-4 PFCR reduction stage.

Similarly, the formulas (44) and (45) can be rewritten as

$$\begin{aligned}u_{4d+1}^{(r)} &= \sum_{j=1}^{2^{2r+1}}\sqrt{2^{-2r-1}}\sin\left(\frac{2j-1}{4}\pi\right)y_{d,2j-1}^{(r)}+\sum_{j=1}^{2^{2r}}\sqrt{2^{-2r-1}}\sin\left(\frac{2j-1}{2}\pi\right)y_{d,4j-2}^{(r)},\\ u_{4d+2}^{(r)} &= \sum_{j=1}^{2^{2r+1}}\sqrt{2^{-2r-1}}\sin\left(\frac{2j-1}{2}\pi\right)y_{d,2j-1}^{(r)},\\ u_{4d+3}^{(r)} &= \sum_{j=1}^{2^{2r+1}}\sqrt{2^{-2r-1}}\sin\left(\frac{2j-1}{4}\pi\right)y_{d,2j-1}^{(r)}-\sum_{j=1}^{2^{2r}}\sqrt{2^{-2r-1}}\sin\left(\frac{2j-1}{2}j\pi\right)y_{d,4j-2}^{(r)}\end{aligned} \tag{50}$$

and

$$\begin{aligned}\left(D-2\cos\left(j\pi/2^{2r+2}\pi\right)I\right)y_{d,j}^{(r)} &= \sqrt{2^{-2r-1}}\sin\left(j\pi/4\right)f_{4d+1}^{(r)}+\\ &\sqrt{2^{-2r-1}}\sin\left(j\pi/2\right)f_{4d+2}^{(r)}+\\ &\sqrt{2^{-2r-1}}\sin\left(j\pi/4\right)(-1)^{j-1}f_{4d+3}^{(r)}+\\ &\sqrt{2^{-2r-1}}\sin\left(j\pi/2^{2r+2}\right)\left(u_{d}^{(r+1)}+(-1)^{j-1}u_{d+1}^{(r+1)}\right).\end{aligned} \tag{51}$$

Clearly this formulation is identical with (28), and thus the second stage of the radix-4 PSCR method is equivalent with the radix-4 PFCR back substitution stage.

When the radix-2 PSCR method is applied to a Poisson problem of the form discussed in Section 2, about half of the sub-problems can be ignored because the eigenvector components corresponding these to sub-problems are zero [34]. The above analysis shows that about quarter of the sub-problems can be ignored in the case of the radix-4 PSCR method. More generally, if the radix-q PSCR method is applied to a problem where $n_1 = q^k - 1$, $k \geq 2$ and $(M_1)^{-1}A_1 = tridiag\{-1, b, -1\}$, $b \in \mathbb{R}$, then

$$(w_i^{(r)})_{j\cdot q^r} = \sqrt{2\cdot q^{-r-1}}\sin\left(\frac{ij\pi}{q}\right) = 0, \text{ for all } j \in \{1, 2, \ldots, q-1\}, \tag{52}$$

if, and only if, $i \in q\mathbb{N}$. Thus, the total number of sub-problems is

$$M_{\text{count}}^{q}(k) = \frac{k(q-1)(2q^k+1)-(q+2)(q^k-1)}{q-1}, \tag{53}$$

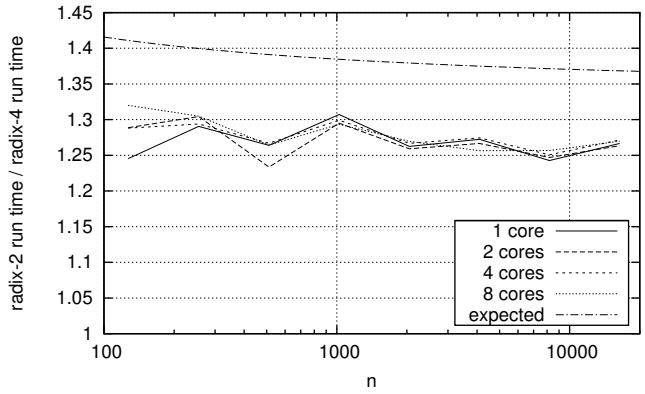

Figure 2: Relative run time difference between the radix-2 and radix-4 PFCR methods for the two-dimensional problems (n^2 unknowns). The expected-line shows the expected run time difference according to (13) and (30).

when all sub-problems are counted and

$$M^{q,0}_{\text{count}}(k) = 2k(q-1)q^{k-1} - q^k + 1, \tag{54}$$

when non-contributing sub-problems are ignored. As a result, the total number of sub-problems is reduced asymptotically by the factor

$$\lim_{k\to\infty} 1 - \frac{M^{q,0}_{\text{count}}(k)}{M^{q}_{\text{count}}(k)} = 1 - (1 - 1/q) = 1/q. \tag{55}$$

7 Numerical results

The implementations of the radix-2 and radix-4 PFCR methods are written using the OpenMP framework [37]. The implementations are applied to two- and three-dimensional problems of the form (1) and the run times are compared. The tests are carried out using a computer with two Intel Xeon 4-core CPUs. All measurements are done using double-precision floating-point arithmetic and the tridiagonal sub-problems are solved using the LU decomposition.

Figure 2 shows the results for the two-dimensional problems. The expected-line shows the expected relative run time difference according to the formulas (13) and (30). Since this estimate takes into account only the number of sub-problems, the real relative run time difference depends largely on the type of these sub-problems. For example, if the matrix D is a full matrix, then the algorithm used to solve the sub-problems dominates the overall run time of the algorithm and the real relative run time difference would track very closely to the expected-line. The results show a quite constant relative run time difference between the methods. That difference is reasonably close to the expected relative run time difference. Both methods scale similarly as the number of cores is increased. The moderate sawtooth pattern is due to the modifications discussed in Section 4.

Figure 3 shows the corresponding results for the three-dimensional problems. Again, the relative run time difference seem to correspond reasonably well to the

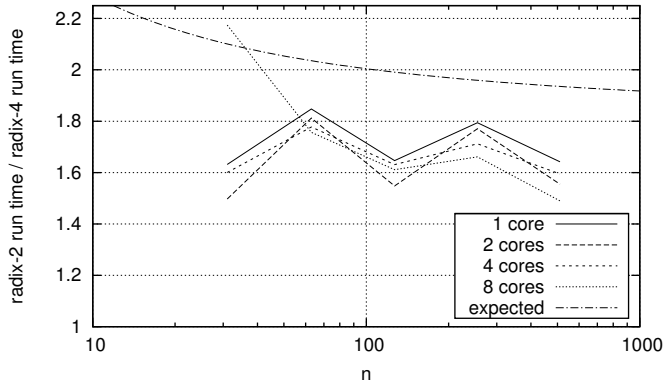

Figure 3: Relative run time difference between the radix-2 and radix-4 PFCR methods for the three-dimensional problems (n^3 unknowns). The expected-line shows the expected run time difference according to (13) and (30).

expectations. Figure 3 might suggest that the radix-4 method does not scale as well as the radix-2 method when the number of cores increases. However, this is likely due to the fact that the scaling of both methods varied more strongly depending on the used hardware and software configuration when compared to the two-dimensional implementation.

Additional comparisons can be found from [38], where GPU implementations were compared to each other and to equivalent CPU implementations. In addition to concluding that the BCR type methods are suitable for GPU computing, the paper also concluded that the radix-4 PFCR method is indeed better able to utilize the GPU's parallel computing resources. Figure 4 shows some of the results obtained in the paper. The implementations were simpler than the ones considered in this paper and utilized a simplified scalar cyclic reduction instead of the standard LU decomposition. As a result, the tridiagonal system solver constituted a smaller portion of the total run time, and thus the relative run time difference between the methods is expected to be slightly smaller.

The radix-4 PFCR method seems to perform well in the case of small and mid-sized problems. This is something that was to be expected since the number of threads required to fully utilize a GPU is very high. Thus, when the problem size is relative small, some of the computing units inside the GPU are left partially unused and the memory band can not be used effectively. However, the radix-4 PFCR method can use these computing units more effectively than the radix-2 PFCR method. A more generalized GPU implementation would use a conventional (parallel) cyclic reduction [39] or similar methods in order to solve the arising tridiagonal sub-problems and, thus the tridiagonal system solver would have taken a larger portion of the total run time. Then it would be expected that the relative run time difference would be even higher. In the case of large problems, both methods can use the computing units nearly equally, and the radix-4 PFCR method loses this additional benefit. Also, both GPU implementations change their behavior when the system size exceeds $n = 1023$ because the threads are arranged in groups with a maximum size of 1024. These are the most probable explanations for the sudden drop in the relative run time

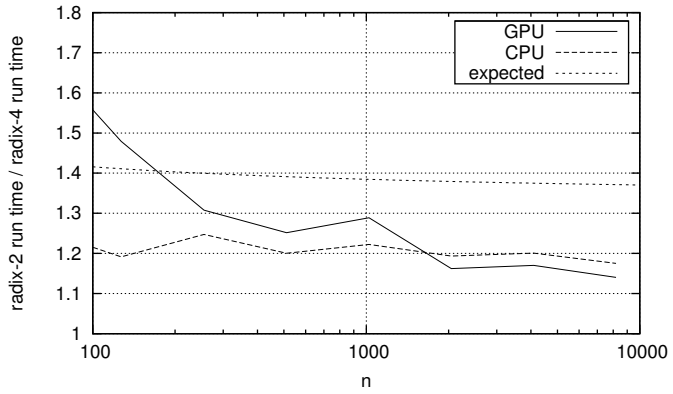

Figure 4: GPU comparison from [38]. Relative run time difference between the radix-2 and radix-4 PFCR methods for the two-dimensional problems (n^2 unknowns). Expected-line shows the expected run time difference according to (13) and (30).

difference when the problem size exceeds $n = 1023$.

8 Conclusions

This paper presents an alternative and intuitive way of deriving a radix-4 block cyclic reduction method for systems with a coefficient matrix of the form tridiag$\{-I, D, -I\}$. The presented method is numerically stable and highly parallel, allowing its efficient implementation on many-core platforms like GPUs. The higher radix number has the effect of reducing the total number of emerging sub-problems when compared to the radix-2 block cyclic reduction method. The method was shown to be equivalent with the radix-4 PSCR method. The measured run time difference between the radix-2 and the radix-4 methods correspond relatively well to the theoretical expectations based on the number of arising sub-problems.

Acknowledgments

The authors thank anonymous reviewers for their valuable feedback. The presentation of the 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.

References

[1] Hockney RW. A fast direct solution of Poisson's equation using Fourier analysis. *Journal of the Association for Computing Machinery* 1965; **12**:95–113, doi:10.1145/321250.321259.

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

[3] Hockney RW. Potential calculation and some applications. *Methods in Computational Physics* 1970; **9**:135–511.

[4] Swarztrauber PN. The methods of cyclic reduction, Fourier analysis and the FACR algorithm for the discrete solution of Poisson's equation on a rectangle. *SIAM Review* 1970; **19**:490–501, doi:10.1137/1019071.

[5] Sweet RA. A cyclic reduction algorithm for solving block tridiagonal systems of arbitrary dimension. *SIAM Journal on Numerical Analysis* 1977; **14**:706–719, doi:10.1137/0714048.

[6] Sweet RA. A parallel and vector variant of the cyclic reduction algorithm. *SIAM Journal on Scientific and Statistical Computing* 1988; **9**:761–765, doi:10.1137/0909050.

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

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

[9] Diamond MA, Ferreira DLV. On a cyclic reduction method for the solution of Poisson's equations. *SIAM Journal on Numerical Analysis* 1976; **13**(1):54–70, doi:10.1137/0713007.

[10] Heller D. Some aspects of the cyclic reduction algorithm for block tridiagonal linear systems. *SIAM Journal on Numerical Analysis* 1976; **13**:484–496, doi:10.1137/0713042.

[11] Swarztrauber PN. Approximation cyclic reduction for solving Poisson's equation. *SIAM Journal on Scientific and Statistical Computing* 1987; **8**(3):199–209, doi:10.1137/0908030.

[12] Swarztrauber PN, Sweet RA. Vector and parallel methods for the direct solution of Poisson's equation. *Journal of Computational and Applied Mathematics* 1989; **27**(1–2):241–263, doi:10.1016/0377-0427(89)90369-5. Special Issue on Parallel Algorithms for Numerical Linear Algebra.

[13] Schwandt H. Truncated interval arithmetic block cyclic reduction. *Applied Numerical Mathematics* 1989; **5**(6):495–527, doi:10.1016/0168-9274(89)90047-0.

[14] Amodio P, Paprzycki M. A cyclic reduction approach to the numerical solution of boundary value ODEs. *SIAM Journal on Scientific Computing* 1997; **18**(1):56–68, doi:10.1137/S1064827595287225.

[15] Reichel L. The ordering of tridiagonal matrices in the cyclic reduction method for Poisson's equation. *Numerische Mathematik* 1989; **56**:215–227, doi:10.1007/BF01409785.

[16] Yalamov P, Pavlov V. Stability of the block cyclic reduction. *Linear Algebra and its Applications* 1996; **249**(1–3):341–358, doi:10.1016/0024-3795(95)00392-4.

[17] Bini DA, Meini B. The cyclic reduction algorithm: from Poisson equation to stochastic processes and beyond. *Numerical Algorithms* 2008; **51**(1):23–60, doi:10.1007/s11075-008-9253-0.

[18] Bini D, Meini B. On cyclic reduction applied to a class of Toeplitz-like matrices arising in queuing problems. *Computations with Markov Chains*, Stewart WJ (ed.), Kluwer Academic Publishers, 1995; 21–38, doi:10.1007/978-1-4615-2241-6_2.

[19] Bini D, Meini B. On the solution of a nonlinear matrix equation arising in queueing problems. *SIAM Journal on Matrix Analysis and Applications* 1996; **17**(4):906–926, doi:10.1137/S0895479895284804.

[20] Bini DA, Meini B. Improved cyclic reduction for solving queueing problems. *Numerical Algorithms* 1997; **15**(1):57–74, doi:10.1023/A:1019206402431.

[21] Bini DA, Meini B. Effective methods for solving banded Toeplitz systems. *SIAM Journal on Matrix Analysis and Applications* 1999; **20**(3):700–719, doi:10.1137/S0895479897324585.

[22] Bini DA, Gemignani L, Meini B. Factorization of analytic functions by means of Koenig's theorem and Toeplitz computations. *Numerische Mathematik* 2001; **89**(1):49–82, doi:10.1007/PL00005463.

[23] Bini DA, Gemignani L, Meini B. Computations with infinite Toeplitz matrices and polynomials. *Linear Algebra and its Applications* 2002; **343/344**:21–61, doi:10.1016/S0024-3795(01)00341-X. Special issue on structured and infinite systems of linear equations.

[24] Bini DA, Fiorentino G, Gemignani L, Meini B. Effective fast algorithms for polynomial spectral factorization. *Numerical Algorithms* 2003; **34**(2–4):217–227, doi:10.1023/B:NUMA.0000005364.00003.ea.

[25] Bini DA, Meini B, Ramaswami V. Analyzing M/G/1 paradigms through QBDs: the role of the block structure in computing the matrix G. *Proceedings of the Third Conference on Matrix Analytic Methods*, Latouche G, Taylor P (eds.), Advances in Algorithmic Methods for Stochastic Models, Notable Publications: NJ, USA, 2000; 73–86.

[26] Bini DA, Iannazzo B, Meini B, Poloni F. Nonsymmetric algebraic Riccati equations associated with an M-matrix: recent advances and algorithms. *Numerical Methods for Structured Markov Chains*, Bini D, Meini B, Ramaswami V, Remiche MA, Taylor P (eds.), no. 07461 in Dagstuhl Seminar Proceedings, Internationales Begegnungs- und Forschungszentrum für Informatik (IBFI), Schloss Dagstuhl, Germany: Dagstuhl, Germany, 2008.

[27] Guo CH. Efficient methods for solving a nonsymmetric algebraic Riccati equation arising in stochastic fluid models. *Journal of Computational and Applied Mathematics* 2006; **192**(2):353–373, doi:10.1016/j.cam.2005.05.012.

[28] Vassilevski P. Fast algorithm for solving a linear algebraic problem with separable variables. *Comptes Rendus de Academie Bulgare des Sciences* 1984; **37**:305–308.

[29] Kuznetsov YA. Numerical methods in subspaces. *Vychislitel'-nye Processy i Sistemy II* 1985; **37**:265–350.

[30] Banegas A. Fast Poisson solvers for problems with sparsity. *Mathematics of Computation* 1978; **32**:441–446, doi:10.2307/2006156.

[31] Kuznetsov YA, Matsokin AM. On partial solution of systems of linear algebraic equations. *Soviet Journal of Numerical Analysis and Mathematical Modelling* 1978; **4**:453–468, doi:10.1515/rnam.1989.4.6.453.

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

[33] Bialecki B, Fairweather G, Karageorghis A. Matrix decomposition algorithms for elliptic boundary value problems: a survey. *Numerical Algorithms* 2011; **56**:253–295, doi:10.1007/s11075-010-9384-y.

[34] Rossi T, Toivanen J. A nonstandard cyclic reduction method, its variants and stability. *SIAM Journal on Matrix Analysis and Applications* 1999; **20**:628–645, doi:10.1137/S0895479897317053.

[35] Rossi T, Toivanen J. A parallel fast direct solver for block tridiagonal systems with separable matrices of arbitrary dimension. *SIAM Journal on Scientific Computing* 1999; **20**:1778–1796, doi:10.1137/S1064827597317016.

[36] Buzbee BL, Golub GH, Nielson CW. On direct methods for solving Poisson's equations. *SIAM Journal on Numerical Analysis* 1970; **7**(4):627–656.

[37] OpenMP (Open Multi Processing). http://openmp.org/.

[38] Myllykoski M, Rossi T, Toivanen J. Fast Poisson solvers for graphics processing units. *Applied Parallel and Scientific Computing*, *Lecture Notes in Computer Science*, vol. 7782, Manninen P, Öster P (eds.). Springer Berlin Heidelberg, 2013; 265–279, doi:10.1007/978-3-642-36803-5_19.

[39] Hockney RW, Jesshope CR. *Parallel computers : architecture, programming and algorithms*. Bristol : Adam Hilger, 1981.