JYVÄSKYLÄ STUDIES IN COMPUTING
155

Jyri Leskinen

Distributed Multi-Objective Optimization Methods for Shape Design using Evolutionary Algorithms and Game Strategies

JYVÄSKYLÄN YLIOPISTO

JYVÄSKYLÄ STUDIES IN COMPUTING 155

Jyri Leskinen

Distributed Multi-Objective Optimization Methods for Shape Design using Evolutionary Algorithms and Game Strategies

Esitetään Jyväskylän yliopiston informaatioteknologian tiedekunnan suostumuksella
julkisesti tarkastettavaksi yliopiston Agora-rakennuksen auditoriossa 1
syyskuun 22. päivänä 2012 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 1, on September 22, 2012 at 12 o'clock noon.

UNIVERSITY OF JYVÄSKYLÄ

JYVÄSKYLÄ 2012

Distributed Multi-Objective Optimization Methods for Shape Design using Evolutionary Algorithms and Game Strategies

JYVÄSKYLÄ STUDIES IN COMPUTING 155

Jyri Leskinen

Distributed Multi-Objective Optimization Methods for Shape Design using Evolutionary Algorithms and Game Strategies

UNIVERSITY OF JYVÄSKYLÄ

JYVÄSKYLÄ 2012

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

URN:ISBN:978-951-39-4844-3
ISBN 978-951-39-4844-3 (PDF)

ISBN 978-951-39-4843-6 (nid.)
ISSN 1456-5390

Jyväskylä University Printing House, Jyväskylä 2012

ABSTRACT

Leskinen, Jyri
Distributed Multi-Objective Optimization Methods for Shape Design using Evolutionary Algorithms and Game Strategies
Jyväskylä: University of Jyväskylä, 2012, 86 p.(+included articles)
(Jyväskylä Studies in Computing
ISSN 1456-5390; 155)
ISBN 978-951-39-4843-6 (nid.)
ISBN 978-951-39-4844-3 (PDF)
Finnish summary
Diss.

This research investigates innovative new methods for increasing the efficiency of shape design optimization are studied. Considerable improvements on algorithmic convergence can be achieved by splitting the geometry of a single-objective problem into an equivalent multi-objective problem and solving it by using competitive Nash games from the field of game theory. Further efficiency improvements can be achieved using a "distributed one-shot" method introduced in this work by decomposing the computational domain and solving both the geometry and domain simultaneously in a single "global" Nash game. The method is inherently parallel and is suitable for distributed platforms. Optimization is done using evolutionary algorithms allowing global and non-smooth optimization. Implementation of modern graphics processing units for the introduced methods is also studied. The methods are evaluated on academic model problems from the field of computational fluid dynamics, available in the Finnish Design Test Case Database. The numerical results presented validate the new approach and open the door to more complex optimization problems of industrial interest.

Keywords: competitive games, computational fluid dynamics, distributed optimization, domain decomposition, evolutionary algorithms, finite element method, GPGPU, Nash algorithms, shape optimization

Author	Jyri Leskinen Department of Mathematical Information Technology University of Jyväskylä Finland
Supervisors	Professor Jacques Périaux Department of Mathematical Information Technology University of Jyväskylä Finland International Center for Numerical Methods in Engineering (CIMNE) Polytechnic University of Catalonia Spain
	Professor Pekka Neittaanmäki Department of Mathematical Information Technology University of Jyväskylä Finland
Reviewers	Professor Marek Rudnicki Institute of Information Technology Faculty of Technical Physics, Applied Mathematics and Information Technology Technical University of Lodz Poland
	Professor David Greiner Institute of Intelligent Systems and Numerical Applications in Engineering (SIANI) University of Las Palmas de Gran Canaria Spain
Opponent	Professor Marko M. Mäkelä Department of Mathematics and Statistics University of Turku Finland

ACKNOWLEDGEMENTS

During my years as a graduate student I became familiar with the joys of mathematical optimization and the strange world of evolutionary algorithms, and I got hooked. In the beginning of my postgraduate studies I was introduced to computational simulation, and a whole new world opened to me. For making possible the adventure in this wonderland, I have to thank many people.

First of all, I would like to express my greatest gratitude to my supervisors, Professor Jacques Périaux and Professor Pekka Neittaanmäki for giving me this opportunity, and for all their guidance. I would also like to thank all the reviewers for giving me crucial feedback and new insights into my work.

I thank all the current and former postgraduate students in the University of Jyväskylä with whom I worked and had inspiring discussions. I would also like to thank all my coworkers in the Design Project team, Hong Wang, Anu Penttilä, Tero Tuovinen, Kati Valpe, and Tuomo Varis, for the opportunity to work in such an interesting project.

I would like to thank Professor Frédéric Hecht from the Paris 6 University for making available such a great scientific software tool for free use by anybody, and for all the instruction he personally provided.

I am also deeply grateful to all the secretaries in the Department of Mathematical Information Technology for taking care of practical matters. I thank the Department, the COMAS Graduate School and TEKES for their financial support.

I will be forever indebted to my parents, who have always inspired and supported me. I am most deeply to my wife Xiaohua, who never stopped believing in me even when I myself did, and to my son Tapio who has made the world such a better place.

LIST OF FIGURES

LIST OF TABLES

CONTENTS

LIST OF INCLUDED ARTICLES

PI Jyri Leskinen, Jacques Périaux and Frédéric Hecht. Nash Games and Adaptive Meshing in a Steady-State Navier–Stokes Shape Reconstruction Problem. Evolutionary Methods for Design, Optimization and Control. T. Burczynski and J. Périaux (eds.), pages 109–114, 2011.

PII Jyri Leskinen and Jacques Périaux. A New Distributed Optimization Approach for Solving CFD Design Problems Using Nash Game Coalition and Evolutionary Algorithms. *To appear in Domain Decomposition Methods in Science and Engineering. R. Bank, M. Holst, O. Widlund and J.C. Xu (eds.)*.

PIII Jyri Leskinen and Jacques Périaux. Distributed evolutionary optimization using Nash games and GPUs – Applications to CFD design problems. *To appear in Computer and Fluids (ParCFD 2011 conference special issue)*.

PIV Jyri Leskinen, Hong Wang and Jacques Périaux. Increasing Paralellism of Evolutionary Algorithms by Nash Games in Design Inverse Flow Problems. *To appear in Computational Methods in Engineering Design and Optimization (special issue of Engineering Computations)*.

PV Hong Wang, Jyri Leskinen, DongSeop Lee and Jacques Périaux. Active Flow Control of Airfoil Using Mesh/Meshless Method Coupled to Hierarchical Genetic Algorithms for Drag Reduction Design. *To appear in Computational Methods in Engineering Design and Optimization (special issue of Engineering Computations)*.

1 INTRODUCTION

Shape design is in central role in the design industry since the performance and the high-tech operability of a product quite often depends on its shape. Because of the ubiquitous nature of shape design problems, a large number of optimization solutions for different applications can be found in the literature. To illustrate this, several examples of applications are listed below.

In many cases, the mass of a supporting structure has to be minimized in order to reduce weight or lower construction costs. On the other hand, the strength of the structure is critical preventing catastrophic failures. Structural shape optimization problems has been studied e.g. in [48, 121, 124].

One of the most traditional fields involving shape optimization is aeronautics. A typical example is minimization of drag, and its importance in reducing fuel consumption [29, 73, 53]. Over the years, the increased efficiency of numerical simulations has steadily made them more realistic (and hence relevant from the industry point of view). Starting from physically simple compressible potential flows [58], the models have become steadily more complex involving three-dimensional, turbulent Navier–Stokes flows solved in massive cluster environments [59]. Other examples involving shape design optimization and computational fluid dynamics include car and ship industries [27], medicine (e.g. blood flow [98]), and paper machine technology [49].

Antenna optimization is another example in the area of design optimization, where the goal is maximization of signal strength [117, 61, 101]. Stealth aircraft design where the reflecting radar signal is minimized presents an opposite problem [72].

Noise generated by air traffic is a major issue. Sources of that noise include jet exhausts and sonic booms. The structure of the aircraft itself is also a major source of noise. Similarly, the rotating blades of wind turbines produce noise. Solutions to these problems benefit from aeroacoustic shape optimization, which consists of finding shapes that produce the minimum amount of noise [24].

It is well known now that in order to reach satisfactory results, a design engineer needs good optimization tools. Traditionally, shapes are built "by hand" based on engineers' experience and tested using physical prototypes. This requires high level of engineering experience. In addition, building and testing prototypes is very time-consuming and expensive.

Fortunately, recent advances in computer technology and improved simulation software have significantly reduced the need for costly prototyping and physical testing. In many single-objective design problems, optimization performed by an experienced engineer can rapidly produce good results. However, for more difficult problems, especially for problems involving multiple objectives, advanced computer-based optimization is needed.

From the introduction of integrated circuit technology until the mid-2000s, the efficiency of central processing units increased steadily. New processors clocked at higher frequencies were able to process larger amounts of data, speeding up the simulation "for free" without the need of updating the algorithms.

However, this trend no longer prevails. Due to limitations arising from the laws of physics, miniaturization of microcircuits can continue only so far. Traditional software does not run much faster in modern computers compared to the ones built nearly a decade earlier.

Facing this problem, CPU manufacturers created an alternative strategy. Instead of increasing the number of transistors or clock frequency, they added more computing cores to the CPU chip. If the CPU had two cores, two program instances could run simultaneously without an appreciable loss of speed.

This multi-core approach must now be taken into account in the development of new algorithms. In high-performance computing, algorithms well-suited for parallelism have been hard to come by for decades. In the case of shape design optimization, parallelism can be implemented in two ways. One way to tackle it is to divide the problem into smaller subproblems and to be solved, which results in an objective function value that is then passed to a sequential algorithm. Another approach is to use intrinsically parallel algorithms, such as evolutionary algorithms, which operate on populations of solution candidates.

Efficient parallel optimization algorithms are critical in modern optimization approaches. With intelligent methods, good solutions can be found even when the problem is difficult. On the other hand, fast optimization methods can reduce the computational time drastically, resulting in faster design cycles and reduced design costs. Furthermore, algorithmic speed-ups can provide reasonable alternatives to traditionally impractically slow methods.

The computer science discipline of parallel computing taken to the extreme can be seen in the recent development of graphics processing units (GPUs). Originally developed for fast 3D gaming, modern GPUs can also be used for solving more general inverse and optimization problems of higher complexity modeled by nonlinear partial differential equations. The often impressive results produced by GPUs have made them a popular alternative in high-performance computing. However, since these GPUs were designed for processing graphics, they are not as versatile as traditional CPUs. Bandwidth and memory issues coupled with the fact that GPU cores are designed to perform simple instructions can reduce the usability of GPU processing in general shape optimization cases.

The importance of shape design and increasingly ubiquitous parallelism coupled with emerging technologies demonstrate the need for new optimization methods. This research is one step further in developing new efficient methods for distributed shape design optimization.

1.1 Objectives of the research

This study has three main goals. First, the speed-up of evolutionary algorithms is studied using ideas from the field of multi-objective optimization and game theory. Efficiency of the so-called “virtual” Nash approach for separable functions (Sefrioui and Périaux [107]) is evaluated on model problems. In this study, a binary-coded Nash genetic algorithm was replaced by a modern adaptive variant of the popular differential evolution algorithm.

The second goal of the study is the development of a new “distributed one-shot” method. It combines the Nash evolutionary algorithms with domain decomposition methods and geometry decomposition methods. Unlike in traditional shape optimization methods, where the direct problem is solved completely for each new design, one-shot methods solve the state equations which model the physical problem simultaneously with shape optimization. However, because of the nature of the algorithms, gradient-based methods are unlikely to produce good results for complex problems where global optimization is needed.

The third goal of the study is utilization of GPUs and evaluation of their associated performances in the context of a distributed design cycle. The proposed innovative methods were tested on selected simple inverse shape design problems in the field of computational fluid dynamics.

1.2 Organization of the thesis

The content of the thesis is organized in the following manner. In Chapter 2, a review of various optimization methods used in the study is given. The state-of-the-art methods are also briefly described.

Chapter 3 consists of game-theoretical aspects of shape optimization. An approach where evolutionary algorithms are enhanced by breaking down the complexity of the detailed design problem into subproblems coupled with Nash game strategies is described and tested. Depending on the physical modeling, a distributed approach is introduced by splitting the decision vector and the objective function into separate subvectors and subfunctions, respectively, and associating a so-called “virtual” multi-objective problem to the original single-objective problem. In this research, this breakdown strategy is achieved using intensively virtual competitive Nash games. The advantages of the geometry decomposition (or in a more general case, split of territory of the design variables) used by the Nash method are presented and discussed.

The three test cases used in this study are listed in Chapter 4. Two of them are from the Finnish Design Test Case Database (TA3, three-ellipse element geometry and TA10, Bi-NACA0012 geometry, available online at <http://jucri.jyu.fi>). The third test case, multi-disk element geometry, was designed for this thesis.

An innovative approach that combines the Nash games with parallel methods such as evolutionary algorithms and domain decomposition methods is introduced in Chapter 5. The flow field and geometry are reconstructed simultaneously with the help of a new "distributed one-shot" strategy using a single global Nash game for both the flow analyzer and the geometry optimizer. The performance of the approach is evaluated on simple 2D geometries in potential flow described in Chapter 4. The results are analyzed and discussed.

Graphics processing units and their performances in shape design optimization are discussed in Chapter 6. Sample results are introduced including the integration of the methods in the distributed Nash approach described in the previous chapter. The conclusions and future prospects are stated in the final chapter. The results for the proposed method in the hybrid CPU/GPU environment studied in Section 6.3 are yet to be published.

1.3 Author's contributions in included papers

Paper I is a study of the efficiency of the Nash geometry decomposition method based on Nash genetic algorithm (Nash GA) introduced in [107]. The author applied the method for an adaptive variant of the differential evolution algorithm (jDE, Brest et al. [12]). The approach was tested on mathematical test functions and the three-ellipse element position reconstruction problem in laminar Navier–Stokes flow (full test case definition in Section 4.1). The approach was found to improve algorithmic efficiency. The observed improvement is purely due to the algorithm, as parallelization was not used. In addition, in order to acquire good results, the mesh employed was adapted during the iterative flow computation process. This mesh regeneration process causes massive numerical noise, for which the authors proposed a solution by including layers of a constant mesh around the ellipses. The reduced noise resulted in improved algorithmic convergence.

In Paper II, a novel "distributed one-shot" method developed by the author is introduced. Based on the original idea of hierarchical Nash algorithms by Périaux et al. [95], the proposed *global Nash game coalition algorithm* (GNGCA) combines domain and geometry decomposition into a single global Nash game. The approach was tested on the multi-disk element geometry position reconstruction problem (Section 4.3) in compressible potential flow. The results show that by reconstructing the shape on subdomains and repairing the global flow field simultaneously, considerable speed-ups can be achieved. It was also found that the reconstruction of flow plays a major role in the efficiency of the method.

Paper III describes the implementation of a legacy compressible Navier–

Stokes flow solver for Nash geometry decomposition methods and GPUs. The original code was developed by B. Mohammadi (NSC2KE [82]) and implemented for GPUs by the author. The Bi-NACA0012 geometry reconstruction problem (Section 4.2) using three different Euler flow conditions was tested. The jDE algorithm was tested with and without the Nash geometry decomposition method. The algorithms were evaluated using two GPU cards, two CPU cores, and 20 CPU cores in the comparisons. The Nash geometry decomposition method brought clear improvements in the efficiency in low-Mach cases. However, the selected geometry decomposition did not work in cases where shocks were present. The performance of the GPUs on a dense mesh produced massive speed-ups (up to 26) compared with the standard CPU approach, indicating that a single GPU card should be superior to the 20 CPU configuration used in the testing. Tests using a synchronous differential evolution algorithm produced identical results, confirming the numerical stability of GPUs in evolutionary computing.

Paper IV is a study conducted by the author to compare different levels of parallelization of evolutionary algorithms. The selected algorithms are jDE, jDE using the island model (based on the idea of Mühlenbein [84]), Nash-jDE using geometry decomposition, and GNGCA. The methods were tested on Bi-NACA0012 geometry reconstruction and three-ellipse element geometry position reconstruction problems in incompressible potential flow. The results show GNGCA is the most efficient and comparable to the Nash approach using two masters and two slaves even though the threads responsible for the flow reconstruction were idle for most of the time. For simulating an optimal load balancing, GNGCA was also run with two slave threads, leading into further improvement in efficiency. The island model did not bring any algorithmic improvement, probably because of the relative simplicity of the objective function landscape.

In Paper V, the authors explore the use of hierarchical genetic algorithms, using models of different levels of fidelity for speeding up the optimization process. The idea is based on the work by Sefrioui and Périaux [106]. The top layer consists of a single subpopulation operating on a precise model. On the middle layer, two subpopulations operate on a model of intermediate accuracy. The bottom layer, consisting of four subpopulations (two for each middle layer populations), operates on a coarse model. The method was validated using a real-world optimization problem consisting of an RAE5243 airfoil in transonic compressible Euler flow. The precise model employs an accurate but computationally intensive meshless method based on the finite volume method. The intermediate model uses a hybrid mesh/meshless method and the coarse model fast but less accurate mesh-based method. The hierarchical approach brought considerable improvements in both the CPU cost and in the quality of the final results. The meshless method and the solver were developed by Ms. Hong Wang. The hierarchical genetic algorithm was implemented by the author.

The numerical tests in Papers I–IV and Chapters 5 and 6 were conducted by the author. In addition, all test cases listed in Chapter 4 were designed by the author.

2 SHAPE DESIGN OPTIMIZATION

Shape optimization is a branch of optimization where the aim is to find the best shape of a structure [52]. In broad sense, shape optimization deals with all problems in which the geometry is subject to optimization. It can be divided into three main types. In *sizing optimization* only the size is modified, otherwise the shape is kept unchanged. Optimization of the thickness of a supporting beam is an example of sizing optimization. In a more strict sense, *shape optimization* consists of finding the optimal geometry by modifying the shape. The shape can be highly deformed, but the topology of the object does not change. A typical example is minimization of drag of an airfoil by deforming the original shape. Finally, *topology optimization* allows even larger changes because the underlying topology is no longer constant. Holes in the structure can be added or removed. Topology optimization has been used for instance in constructing optimal internal supporting structures.

2.1 Mathematical optimization

Optimization in general is a process which consists of finding the optimal solution $\mathbf{x}^* = (x_1, x_2, \ldots, x_N)$ from a set of admissible solutions S that minimizes (or maximizes) a real-valued function $f : S \to \mathbb{R}$ subject to some additional constraints. Mathematically this can be formulated as follows:

$$\begin{aligned} \min_{\mathbf{x} \in S} f(\mathbf{x}) & & \\ \text{subject to } g(\mathbf{x}) &\leq 0 \\ h(\mathbf{x}) &= 0, \end{aligned} \tag{1}$$

where $g : S \to \mathbb{R}$ and $h : S \to \mathbb{R}$ are given functions.

The set of parameters $\mathbf{x}$ is called *design vector*. Maximization problems can be trivially transformed into minimization problems by simply changing the sign of the objective function.

In shape design optimization, a mathematical model that describes the physical behavior of the system must be constructed. This is done with the help of partial differential equations (PDEs) called *state equations* which are solved in a given region (*domain*, Ω).

Shape design problems can be considered as optimization problems constrained by state equations,

$$\begin{aligned} \min_{\Omega \in S_\Omega} f\left(u(\Omega), \Omega\right) & \\ \text{subject to } c\left(u(\Omega), \Omega\right) &= 0 \end{aligned} \tag{2}$$

where $u(\Omega)$ is the solution to the PDE in the domain $\Omega \in S_\Omega$ (the set of admissible domains) and the constrain $c = (u(\Omega), \Omega)$ consists of the governing state equations. The shape of the domain Ω depends on the selected shape parametrization and the values of design vector $\mathbf{x}$.

2.2 Design process

Shape design is an iterative process typically consisting of the phases described below. First, a mathematical model for the optimization problem is constructed. The appropriate state equations are selected. The range of geometry changes is determined. Depending on the properties of the objective function, a suitable optimization algorithm is selected.

The optimization algorithm controls the optimization process (illustrated in Figure 1). It produces new solution candidates (i.e. design vector values) which describe the given geometry. In industrial design, shape parameterization is typically done using parameterized CAD models.

In the second phase, the computational domain is discretized using a mesh. If the changes in geometry are sufficiently small, the mesh can be updated by deformation instead of time-consuming remeshing. Massive mesh regeneration should be avoided because it introduces numerical noise into the objective function.

The resulting mesh is passed to the matrix assembler, which constructs a new linearized system of equations $A\mathbf{u} = \mathbf{f}$ based on the mesh and the PDEs. The system of equations is passed to the solver which sends the computed solution $\mathbf{u}$ to the analyzer for computing the corresponding objective function value. The new objective function value is passed back to the solver. Finally, once the optimal solution has been found, it is visualized and analyzed.

Since all the aspects of the problem are rarely known *a priori*, the problem formulation and optimization process itself may require several iterations before a satisfying design can be found.

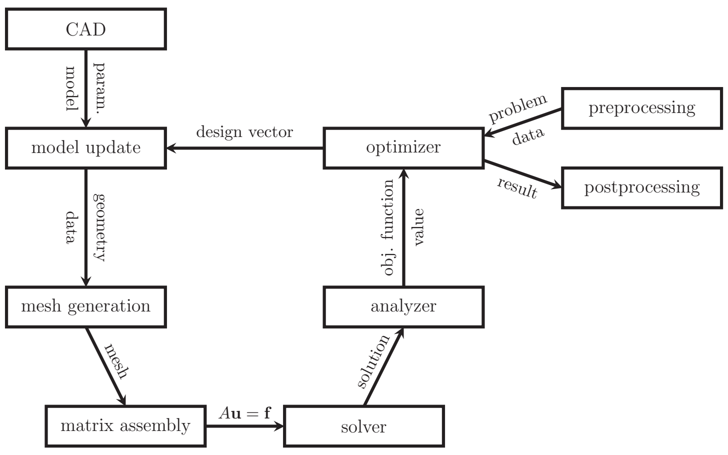


FIGURE 1 Design process cycle.

2.3 Overview of optimization algorithms

Successful optimization depends on the selected algorithm. As the so-called "no free lunch" theorem states, there cannot be a single algorithm that is optimal for all kinds of problems [125]. Instead, the most suitable algorithm must be selected based on the *a priori* information of the problem.

If the shape design problem does not involve very large geometric changes (i.e. the objective function is probably simple) and if the function evaluation is computationally very intensive, gradient-based methods should be used. In more general cases, for example in developing completely new designs, global search methods are better in finding new solutions.

In the following, optimization methods used in shape design optimization are discussed. The methods used in this work are described in detail.

2.3.1 Gradient-based methods

Classical optimization methods employ gradient information. They are the most efficient methods in continuous optimization because of comparatively small number of function evaluations needed. Convergence to a local optimum is guaranteed, depending on the accuracy of the gradient computation and smoothness of the function.

Shape optimization methods can be divided into two groups [69]. *Loosely coupled optimization* refers to methods that do not require specific information on the function gradients. Instead, they are calculated using finite differences. This "black box" approach is straightforward but can be prohibitively slow if there are many design parameters. For example, if n_{dim} is the number of dimensions of the problem, and n_{eval} is the number of function evaluations needed to achieve the desired result, the algorithm will need the order of $n_{dim} \times n_{eval}$ function evaluations. If the problem is computationally intensive, this may be too expensive to calculate.

In contrast, the *adjoint methods* use tightly coupled optimization. Instead of computing the gradients separately, they use the adjoints of the original state equations. This reduces the number of evaluations to $2n_{eval}$ (the factor 2 comes from the doubling of governing equations). The efficiency can be further improved by so-called *one-shot methods* which combine the state equations, their adjoints and design equations into one large system of equations removing the need of iterative optimization [5, 67].

2.3.2 Evolutionary algorithms

Neither the gradient or direct local search methods can work adequately in the case of multi-modal objective functions. Therefore, search methods that operate globally in the whole search space are required. These methods are stochastic in nature and their convergence to an optimum cannot be guaranteed.

There are a number of different global methods developed. The most popular are undoubtedly the so-called *evolutionary algorithms* (EAs) which mimic natural evolution. These include genetic algorithms [56], evolutionary strategies [100, 105] and evolutionary programming [33], which all were originally introduced in the 1960s. A more recent method is the popular differential evolution, developed in the 1990s [115, 116]. Other population-based methods include particle swarm optimization where the solution candidates move as “swarms” around the most promising areas [63], and ant colony optimization methods which simulate the movement of ants using “pheromones” [26]. Examples of other commonly used global heuristic methods include simulated annealing [64] and tabu search [40].

Due to their inherently parallel nature, parallelization of evolutionary algorithms is straightforward. Indeed, parallelization of EAs is considered an “embarrassingly parallel” problem [13]. The master process sends the solution candidates to slaves, and they send the corresponding objective function values back to the master (Figure 2). If the objective function computation takes a fixed time, the implementation of a parallel evolutionary algorithm is trivial. However, in shape optimization the evaluation time may be highly variable because of the variable number of nodes in the mesh and the convergence rates of the solver for example. Therefore, non-generational approaches should be considered.

Evolutionary algorithms for shape design optimization have been under intensive research because of their global scope and problem-independent approach. Several approaches have been proposed in order to improve the accuracy and high computational cost of the methods.

The amount of computation required can be reduced by replacing the costly function evaluation. One group of popular methods is the *metamodels* [122]. A

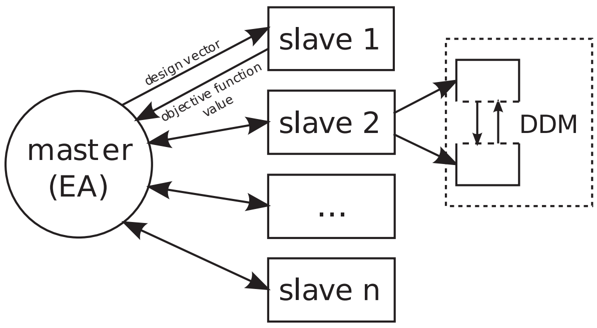


FIGURE 2 Parallelized evolutionary algorithm. The work is distributed to slave processes which can solve the objective function using further levels of parallelization.

mathematical model is constructed based on a limited set of actual objective function values. This computationally inexpensive model is then optimized. Until the algorithm has converged, a new set of function values are produced and the optimizer is restarted. The process is repeated until an adequate solution has been found. There exists a variety of different methods for constructing metamodels, for example neural networks which are especially suitable for curve fitting. For example, in [38] an advanced metamodel-based evolutionary algorithm using neural networks was implemented.

Multi-level algorithms operate on two or more levels in order to minimize computational cost [39, 62]. In multi-level evaluation, a high-fidelity (accurate) model is coupled with a faster low-fidelity model. For example, by using simpler, less accurate state equations low-fidelity models can be produced. Alternatively, different mesh densities can be used. One genetic algorithm using a hierarchical tree of populations, with each level operating an objective function of different accuracy was introduced in [106]. Other examples of multi-level algorithms include multi-level search (e.g., gradient-based search coupled with global metaheuristics) and multi-level parameterization, where the number of design parameters are reduced at lower accuracy levels.

Another approach is to hybridize the evolutionary approach by employing local search methods in order to improve algorithmic convergence. Following the idea of "memes" by R. Dawkins [19], *memetic algorithms* are a class of evolutionary algorithms that do not directly follow the idea of Darwinian evolution [83, 51]. At its simplest, a memetic algorithm is a hybrid evolutionary algorithm which implements local search within the search cycle. More advanced methods use operators, for example for maintaining population diversity in order to prevent premature convergence [87].

Genetic algorithm (GA) First introduced in the 1960s by J. Holland [56], genetic algorithms became popular after the publication of the seminal book of D. Goldberg in 1989 [46]. Of the evolution-based methods, genetic algorithms follow most closely the natural evolution. The algorithm is simple: first, a set (*population*) of solution candidates (*individuals* or *chromosomes*) is constructed and the candidates are evaluated. Some of the individuals, preferably but not only the fittest ones, are selected for parents. Using a process called *crossover* to combine the parent chromosomes, new offspring are formed. Some of the offspring are also subjected to *mutation*. The best offspring, or the best individuals from the combined population of parents and offspring, are selected for the next population. Usually the fittest individual, called *elite*, is automatically selected in order to prevent the loss of the most promising candidate solution. Crossover and mutation generate new solution candidates, whereas selection controls the convergence of the algorithm. A general genetic algorithm is listed below.

1. Generate initial population $pop^{(0)}$.
2. Until the termination criteria has been satisfied, compute a new generation:
   (a) Select parent chromosomes $pop_p^{(i)} \in pop^{(i)}$.
   (b) Generate initial offspring population $pop_o^{(i)}$ by recombining the parents (crossover).
   (c) Mutate some of the offspring in $pop_o^{(i)}$.
   (d) Select individuals for next generation, $pop^{(i+1)}$, from the offspring and parent populations.
   (e) Continue from step (2).

Because genetic algorithms are extremely versatile, they have been applied to a vast number of problems, ranging from continuous problems such as shape optimization to combinatorial problems such as finding minimum travel distances in the traveling salesman problem. The chromosomes were originally encoded using binary strings, each binary value representing a "gene". The accuracy of the binary-valued algorithm can be adjusted by changing the length of the binary string. In continuous optimization, real-valued chromosomes differ from the traditional idea of genes, but can be more efficient because the crossover and mutation operators can be selected based on the geometry of the problem. For example, the blend crossover (BLX [30]) produces new offspring that are often located between the parents. Binary string based methods such as the one-point crossover produce offspring that can be located far from both of the parents.

Differential evolution (DE) Differential evolution, introduced by Storn and Price in 1995, was designed for continuous global optimization [115, 116]. The idea of the algorithm is very simple yet efficient. In traditional genetic algorithms, crossover plays the major role in constructing a new individual, mutation being

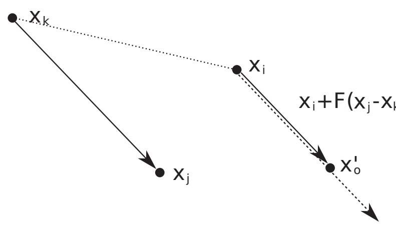


FIGURE 3 Mutation operator in differential evolution.

a minor component. In differential evolution, like in evolutionary strategies, new individuals are primarily constructed with the help of the mutation operator. One of the strengths of the algorithm is that it is very simple and requires only three control parameters.

The algorithm proceeds as follows. As in the case of genetic algorithms, the initial population of size NP is formed randomly. An intermediate individual $\mathbf{x}_o^{'}$ is generated by differential mutation. First, three different random individuals are selected ($\mathbf{x}_{i,j,k}$). The difference between $\mathbf{x}_j$ and $\mathbf{x}_k$ is multiplied using the mutation factor F. The result is added to the third individual $\mathbf{x}_i$ (Figure 3).

The final offspring $\mathbf{x}_o$ is produced by randomly mixing the variables of the intermediate offspring and the individual $\mathbf{x}_n$ in the parent population using the crossover rate CR. In order to prevent the final individual from being identical to $\mathbf{x}_n$, one gene is copied directly from the intermediate offspring. Finally, the new offspring is evaluated and compared to $\mathbf{x}_n$. The fitter one is selected for the next generation. A sketch of the algorithm is listed below.

```
1. Generate initial population $pop$ of size $NP$.
2. Until the termination criteria has been satisfied
    (a) Repeat for each $n \in \{1, \ldots, NP\}$:
         i. Select unique random values $i, j, k \in NP$.
        ii. Mutation: $\mathbf{x}_o^{'} := \mathbf{x}_i + F \cdot (\mathbf{x}_j - \mathbf{x}_k)$
       iii. Crossover:
            – Select random value $m \in D$ ($D$ is the number of dimensions).
            – For each $d \in \{1, \ldots, D\}, d \neq m$:
              If rand(0,1) $\leq CR$, $x_o[d] := x_n[d]$, else $x_o[d] := x_o^{'}[d]$.
        iv. Selection: If $f(\mathbf{x}_o) < f(\mathbf{x}_n)$, replace $\mathbf{x}_n$ with $\mathbf{x}_o$.
    (b) Continue from step (2).
```

There are many variants of the original algorithm. The one described above is referred to as the $DE/rand/1/bin$ variant. Other variants introduced by the developers replace the individual x_i by the elite individual, resulting in faster convergence in some functions ($DE/best/1/bin$), or use more than one vector difference in producing new individuals (e.g. $DE/rand/2/bin$). The crossover operator can also be replaced (e.g. $DE/rand/1/exp$). For comparison of various differential evolution variants, please refer for example to [80].

A large number of further modifications can be found in the literature [88]. One of the variants (jDE [12]), is used extensively in this thesis. Instead of using fixed values for the control parameters F and CR, each individual is assigned with its own value, $F_i \in [F_{\min}, F_{\max}]$ and $CR_i \in [0,1]$, $i = 1, \ldots, NP$. For the initial population, the values are selected randomly. When new individuals are introduced, the old value is either retained or replaced by a new random value,

$$\begin{cases} F_i^{k+1} &= (F_{\max} - F_{\min}) \cdot \text{rand}(0,1) + F_{\min} & \text{if rand}(0,1) < \tau_1 \\ F_i^{k+1} &= F_i^k & \text{otherwise} \end{cases}$$

and similarly

$$\begin{cases} CR_i^{k+1} &= \text{rand}(0,1) & \text{if rand}(0,1) < \tau_2 \\ CR_i^{k+1} &= CR_i^k & \text{otherwise} \end{cases}$$

The authors use the value $\tau_1 = \tau_2 = 0.1$ for the probability adjustment parameters. This value was adopted in the present work.

The island model In nature, organisms rarely form a single population where the genetic material is transferred freely. Geographical and other obstacles cause a single species to split into several subpopulations with limited genetic interaction between them. Prolonged isolation leads to a high genetic difference between the different populations. To illustrate the idea, one can consider an archipelago which is settled by a new species. Originally the populations on different islands are fairly homogeneous, but over time the populations diverge to different species, each of which is specialized to the environment of their particular island.

The *island model genetic algorithm* is based on this idea [84]. An evolutionary algorithm that operates on a single population is in constant danger of losing genetic diversity leading to premature convergence. If the population is divided into several groups (*demes*) which have limited communication between them, the algorithm can retain higher diversity. If the objective function is highly multimodal, this approach is especially useful because the subpopulations can explore different local optima.

The populations are not completely separate, however. After a certain period, for example after a fixed number of generations, some of the individuals, usually the elites migrate to other populations replacing some of the original individuals. The migration process requires a topology between the subpopulations. Popular topologies include hypercube, two or three-dimensional grid and torus [90]. An example torus structure is illustrated in Figure 4.

The island model provides an additional layer for parallelization. For example, each population can be operated on a different computer or cluster of computers. Compared to the standard approach, the method is more robust against high latency since the information exchange between the populations is limited.

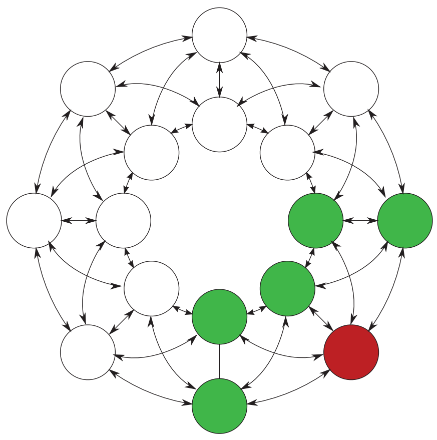


FIGURE 4 Example of the island model topology (torus, [84]). The neighbors of the dark gray population are marked with lighter gray.

2.4 Solving state equations

Except for the very simplest of cases, PDEs cannot be solved analytically. Numerical methods are used instead. In order to be solvable in computer, the original problem must be discretized. Several different discretization methods applicable to the purpose have been developed. Of these, the most commonly used are the *finite differences* (FDM), *finite elements* (FEM), and *finite volumes* (FVM) [15]. Each of these methods discretize the domain using a *mesh* (or a *grid*), which consists of nodes and edges connecting them. Methods that do not require meshing include the *boundary element method* (BEM [6]) and meshless methods that operate on clouds of points [9, 123].

The finite difference method is the most straightforward of the methods. It is easy to implement and, compared to the other methods, is not mathematically complex. The main drawback is that it can operate only on regular grids, which limits the use of the method to relatively simple geometries.

The finite element method is a very versatile PDE discretization method. It can operate on unstructured meshes allowing complex geometries. However, in order to make them solvable, the original PDEs have to be transformed into a weak form.

The finite volume method is based on the evaluation of volumes around the mesh nodes. Because the flux remains constant in adjacent volumes, the method is conservative and especially suitable for computational fluid dynamics problems. The method can also be implemented for unstructured meshes.

In this study, both finite element and finite volume methods are used. For an in-depth description of the methods, please refer to [55, 96, 15].

Depending on the method used, discretization results in a system of linear

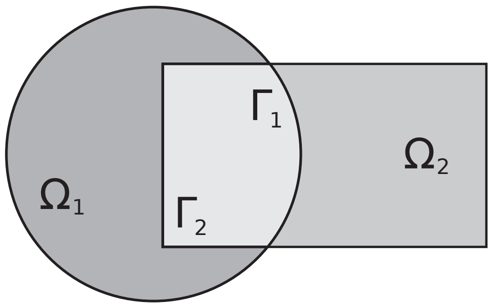


FIGURE 5 A sample domain decomposition problem.

equations

$$A\mathbf{u} = \mathbf{f}, \tag{3}$$

where A is a sparse matrix, $\mathbf{u}$ is the value of the approximate solution and $\mathbf{f}$ is the right-hand side containing values from the boundary conditions and forces.

There are many methods for solving linear systems with large sparse matrices. Direct methods solve the system of equations using factorization. Iterative methods compute approximate results iteratively until the target accuracy has been achieved. The former kinds of methods are efficient on moderate-sized matrices, but because of higher memory requirements, they are not suitable for large-scale problems. Iterative methods are more memory-efficient. The most relevant methods for this work are the UMFPACK unsymmetric multifrontal sparse LU factorization package [18], and the preconditioned conjugate gradient method [54].

2.5 Domain decomposition methods

The term *domain decomposition method* (DDM) refers to a group of methods for splitting a domain or a discretized system of equations into smaller problems. There are several reasons for doing this [109]. The classical application, introduced by H. Schwarz in 1870, was to solve elliptical boundary value problems analytically for complex shapes [104] (Figure 5 depicts this classic example).

Domain decomposition methods are commonly used as preconditioners either in order to speed up the solution time or to make a poorly conditioned system better behaving [79]. For example, if the solver can solve a matrix one-tenth the size of the original one much more than ten times faster, solving the problem using DDM with ten subdomains could result in faster convergence.

The most relevant application of domain decomposition for this study is the parallel solution of a system of equations (3). In addition to obvious increases in solution time due to parallelism, DDM makes it possible to solve massive problems too large for a single machine.

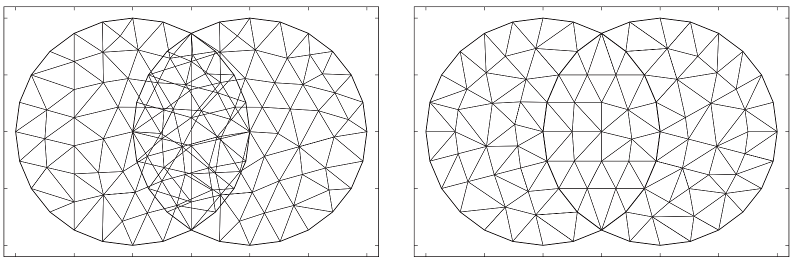


FIGURE 6 Examples of nonmatching and matching meshes.

Two common classes of domain decomposition methods are the *Schwarz methods*, which use overlapping domains, and the *Schur complement methods* (also known as *substructuring methods*) where the domains are connected by an internal boundary and do not have common nodes elsewhere [109]. The former method is straightforward, while the latter one is more efficient in parallel computing because less information needs to be passed when the global domain is reconstructed. Examples of more advanced methods include the FETI method (*finite element tearing and interconnect* [31]), which further reduces communication overhead between subdomains.

The overlapping meshes can be either matching or non-matching (Figure 6). In the matching case the meshes share the nodes on the overlap. In the non-matching case, extra caution has to be taken when the global domain is reconstructed. In this work, only overlapping subdomains which use matching grids are studied.

2.5.1 Alternating Schwarz method

Let us consider the following Poisson boundary value problem:

$$\begin{array}{rcll} -\Delta\varphi & = & f & \text{in} \quad \Omega \\ \varphi & = & g & \text{on} \quad \partial\Omega \end{array} \tag{4}$$

With two overlapping subdomains ($\Omega_1 \cup \Omega_2 = \Omega$, overlap $\Omega_{1,2} = \Omega_1 \cap \Omega_2 \neq \emptyset$), the equation becomes

$$\begin{array}{rcll} -\Delta\varphi_1 & = & f & \text{in} \quad \Omega_1 \\ \varphi_1 & = & g & \text{on} \quad \partial\Omega_1 \setminus \Gamma_1 \\ \varphi_1 & = & \varphi_2 & \text{on} \quad \Gamma_1 \end{array} \tag{5}$$

and

$$\begin{array}{rcll} -\Delta\varphi_2 & = & f & \text{in} \quad \Omega_2 \\ \varphi_2 & = & g & \text{on} \quad \partial\Omega_2 \setminus \Gamma_2 \\ \varphi_2 & = & \varphi_1 & \text{on} \quad \Gamma_2 \end{array} \tag{6}$$

where Γ_1 and Γ_2 are the overlap boundaries (Figure 5).

The original domain decomposition method, known as the *alternating Schwarz method*, operates in the following manner. The process is started by selecting the initial values for the overlapping boundary Γ_1. The equation (5) is solved and the new values of φ_1 are passed to Γ_2. Similarly, the values on the boundary Γ_1 are replaced with the updated value of φ_2. The process is iterated until the discrepancy between φ_1 and φ_2 on the boundaries Γ_1, Γ_2 and in the overlap $\Omega_{1,2}$ is minimized.

One alternating Schwarz iteration is

$$\begin{aligned} -\Delta\varphi_1^{n+\frac{1}{2}} &= f && \text{in} \quad \Omega_1 \\ \varphi_1^{n+\frac{1}{2}} &= g && \text{on} \quad \partial\Omega_1 \setminus \Gamma_1 \\ \varphi_1^{n+\frac{1}{2}} &= \varphi_2^n && \text{on} \quad \Gamma_1 \end{aligned} \tag{7}$$

and

$$\begin{aligned} -\Delta\varphi_2^{n+1} &= f && \text{in} \quad \Omega_2 \\ \varphi_2^{n+1} &= g && \text{on} \quad \partial\Omega_2 \setminus \Gamma_2 \\ \varphi_2^{n+1} &= \varphi_1^{n+\frac{1}{2}} && \text{on} \quad \Gamma_2 \end{aligned} \tag{8}$$

With a minor modification, the Schwarz method can be parallelized:

$$\begin{aligned} -\Delta\varphi_1^{n+1} &= f && \text{in} \quad \Omega_1 \\ \varphi_1^{n+1} &= g && \text{on} \quad \partial\Omega_1 \setminus \Gamma_1 \\ \varphi_1^{n+1} &= \varphi_2^n && \text{on} \quad \Gamma_1 \end{aligned} \tag{9}$$

and

$$\begin{aligned} -\Delta\varphi_2^{n+1} &= f && \text{in} \quad \Omega_2 \\ \varphi_2^{n+1} &= g && \text{on} \quad \partial\Omega_2 \setminus \Gamma_2 \\ \varphi_2^{n+1} &= \varphi_1^n && \text{on} \quad \Gamma_2 \end{aligned} \tag{10}$$

3 GAME THEORY FOR MULTI-OBJECTIVE DESIGN OPTIMIZATION

In this chapter several different game strategies that can be applied to shape design optimization are described. The Nash algorithms are discussed in detail. The Nash approach can be applied on single-objective optimization problems using a process called "virtualization". Finally, the efficiency of the method is tested on several mathematical functions.

3.1 Overview of game strategies

This section provides a brief introduction to the game theory. A *game* is played between a group of players (from one to infinity) who each have a set of *strategies* with certain *payoffs*. In a typical game, each player maximize his/her payoff. Normally it is assumed that the players are rational, i.e. they always choose the best available strategy. However, they may not have all the information available, and the game may be affected by chance. The players' strategies are typically conflicting: for example, if Player 1 increases his payoff by selecting strategy A, Player 2 correspondingly loses.

There are several types of games. For example, the players may form a *coalition* to try to maximize their combined payoff, or they could act sequentially. In this section, three types of games relevant to this thesis are introduced: cooperative, competitive, and hierarchical games.

3.1.1 Cooperative games

A *cooperative game*, or *Pareto game* (after V. Pareto who introduced the idea [94]), is a game where a coalition of players try to maximize their combined payoff. Multi-objective optimization problems consisting of two or more conflicting criteria can be considered as Pareto games.

The idea of a multi-objective problem can be illustrated with a simple exam-

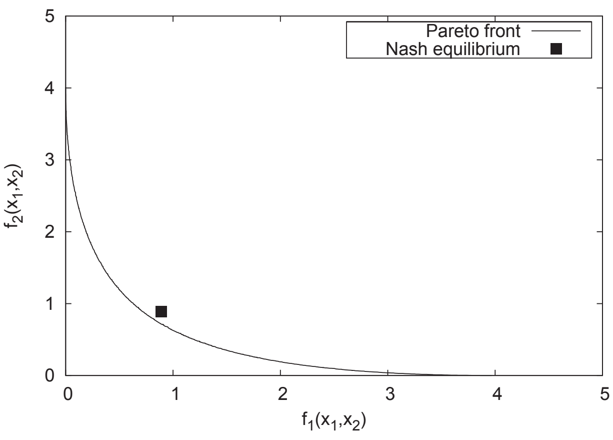


FIGURE 7 Example Pareto front and (dominated) Nash equilibrium.

ple [21]. Let us consider a person who is going to buy a new car. The person has two criteria, quality and price. Obviously, he is interested in buying a car which is as inexpensive as possible and of as good quality as possible. Unfortunately, these objectives are in conflict, and he will not be able to find a car which is both inexpensive and of good quality. Therefore, he has to pick several candidates with different prices and qualities. Some of the cars are both more expensive and worse in quality than others. Those he can reject immediately. The rest form a set of cars from which he can pick the one he prefers. Based on these two criteria, both being equal, he cannot select the best solution automatically without additional information about the problem.

More formally, a decision vector $\mathbf{x}^* \in S$ (S is the set of available strategies) is *Pareto optimal* if there does not exist another decision vector $\mathbf{x} \in S$ such that $f_i(\mathbf{x}) \leq f_i(\mathbf{x}^*)$ for all $i = 1, \ldots k$ and there exists at least one j such that $f_j(\mathbf{x}^*) < f_j(\mathbf{x})$ [81]. A decision vector that is not Pareto optimal is *dominated*. The set of optimal points is called a *Pareto front*. In higher dimensions, it is called a *Pareto surface*. An example of a simple two-dimensional convex and continuous front is illustrated in Figure 7.

3.1.1.1 Multi-objective optimization

Multi-objective optimization usually consists of capturing the Pareto front. In real life cases, a satisfactory solution has to be selected, using multi-criteria decision making, from the set of Pareto optimal solutions [11].

In industrial design, multi-objective problems are the norm. Typically several coupled physical phenomena affect the system leading to conflicting criteria (*multi-physics optimization*). When a complex system such as an aircraft is being developed, several engineering disciplines have to be taken into account (*multi-disciplinary optimization*). Examples from the field of aeronautics include diverse problems such as minimization of sonic boom and maximization of lift [68], optimization of lift on multi-element airfoils for different flow regimes [97], maximization of lift and minimization of radar visibility [78], to name a few.

A wide variety of multi-objective optimization algorithms has been developed. Some early approaches were based on weighted sums, i.e. the multi-objective problem was transformed into a single-objective problem and then solved using single-objective algorithms [112]. This approach is very limited because it cannot recover the front properly if it is not convex.

Evolutionary algorithms are particularly well-suited for multi-objective optimization because they operate on populations of solution candidates. In addition to the global approach, the parallelism provided by populations help the algorithm in capturing the global Pareto front [35]. Examples range from the early *vector evaluated genetic algorithm* (VEGA) [103] to more sophisticated algorithms such as *multiple objective genetic algorithm* (MOGA) [34], *non-dominated sorting genetic algorithm* (NSGA) [110], *strength Pareto evolutionary algorithm* (SPEA) [126], *Pareto archived evolution strategy* (PAES) [66], and the popular NSGA-II [22]. For a comprehensive review of evolutionary multi-objective algorithms, please refer to [17, 16]. A survey on a wide variety of multi-objective methods for design problems can be found in [3].

Hybrid evolutionary algorithms for speeding up algorithmic convergence are also common in multi-objective optimization [119, 108]. Various memetic multi-objective algorithms are described in [45].

3.1.2 Competitive games

The nature of the game changes dramatically if the players do not cooperate. *Competitive games*, also known as *Nash games* (after J. Nash who formalized them in 1950 [86]) are games where the players maximize their payoffs by taking into account the available strategies of their opponents.

Competitive games can be illustrated using the classical example known as the *Prisoners' dilemma* [99]: Two men are arrested, but there is not enough evidence for a conviction. Therefore, both the men are given two options: either betray your partner or remain silent. The person committing betrayal walks free, and his partner will serve a ten-month sentence. If they both remain silent, they both have to serve for two months for a minor charge. If they both betray each other, they have to serve six months each. The men are separated and cannot be aware each other's decisions.

The payoff matrix of the game is shown in Table 1. In the first case, a prisoner assumes his accomplice will not betray him. If he himself betrays his partner, he will avoid the prison sentence. On the other hand, if he correctly assumes that his accomplice will betray him, he would save four months of prison by doing likewise. Again, the "betray" strategy would be better.

Since both of them are assumed to be rational and the game is symmetric, they will betray each other, and both will serve a six-month prison term even though they could save four months by selecting the "remain silent" strategy. This situation, called the *Nash equilibrium*, is stable because after reaching it nei-

TABLE 1 Payoff matrix of the Prisoner's Dilemma. The Nash equilibrium is in bold.

		Prisoner 2	
		silent	betray
Prisoner 1	silent	(2, 2)	(10, 0)
	betray	(0, 10)	**(6, 6)**

ther of them can improve their payoffs any further.

Let us define the Nash equilibrium mathematically. Consider a two-player game. Let S_1 and S_2 be the set of strategies and J_1 and J_2 the payoff functions of Player 1 and 2, respectively. A strategy pair (x_1, x_2), $x_1 \in S_1$, $x_2 \in S_2$ is a Nash equilibrium if and only if

$$\begin{aligned} J_1(\bar{x}_1, \bar{x}_2) &= \inf_{x_1 \in S_1} J_1(x_1, \bar{x}_2) \\ J_2(\bar{x}_1, \bar{x}_2) &= \inf_{x_2 \in S_2} J_2(\bar{x}_1, x_2) \end{aligned} \tag{11}$$

This can be trivially expanded into an n-player game,

$$\begin{aligned} J_1(\bar{x}_1, \bar{x}_2, \ldots, \bar{x}_n) &= \inf_{x_1 \in S_1} J_1(x_1, \bar{x}_2, \ldots, \bar{x}_{n-1}, \bar{x}_n) \\ J_2(\bar{x}_1, \bar{x}_2, \ldots, \bar{x}_n) &= \inf_{x_2 \in S_2} J_2(\bar{x}_1, x_2, \ldots, \bar{x}_{n-1}, \bar{x}_n) \\ &\vdots \\ J_n(\bar{x}_1, \bar{x}_2, \ldots, \bar{x}_n) &= \inf_{x_n \in S_n} J_n(\bar{x}_1, \bar{x}_2, \ldots, \bar{x}_{n-1}, x_n) \end{aligned} \tag{12}$$

The idea of Nash games can be applied to optimization. A multi-objective optimization problem can be solved as a Nash game where the different criteria are optimized by different players. For example, wing shape optimization involving minimization of drag and maximization of lift can result in an adequate compromise at the Nash equilibrium if it is located close to the Pareto front, avoiding the time consuming Pareto search. Likewise, hybridizing Pareto games with competitive games can provide substantial speed-ups because the algorithm can be seeded with good solutions from the Nash game [70, 71].

3.1.3 Hierarchical games

The last game type described in this manuscript is called *hierarchical game*, or *Stackelberg game* (after H. F. von Stackelberg who introduced it [111]). It is a strategic game where one player, called *leader*, moves first, and the other players, *followers*, react [77]. The game can have perfect information, the players knowing, beforehand, all the possible information and the situation they currently are in.

A typical Stackelberg game situation arises when a new company appears into a market dominated by another company. The new company has to select the optimal strategy based on the actions of the market leader. Distributed shape design optimization, where the computational domain is decomposed into subdomains and solved for every design, could be considered a kind of Stackelberg game where the shape optimizer is the leader and the domain decomposition method is the follower [95].

Hierarchical games result in *Stackelberg equilibria* [25]. Let f_1 be the criterion of the leader and $f_i, i = 1, \ldots, N$ the criteria of the followers. Then the Stackelberg equilibrium is a result where the leader optimizes all the controllable variables according to his own strategy based on the Nash equilibrium of the followers,

$$\min_{\mathbf{x} \in S} f_1(\mathbf{x}), \tag{13}$$

where $\mathbf{x} = (x_1, x_2, \ldots, x_N)$ and $S = S_1 \times S_2 \times \cdots \times S_N$ is the space of admissible strategies. The values $(x_2, x_3, \ldots, x_N)$ come from the Nash equilibria of the followers,

$$f_i\left(x_1, \bar{x}_2, \ldots, \bar{x}_N\right) = \inf_{x_i \in S_i} f_i\left(x_1, \bar{x}_2, \ldots, \bar{x}_N\right) \text{ for } i = 2, \ldots, N \tag{14}$$

3.2 Nash genetic algorithms

If the objective function is smooth and unimodal, locating Nash equilibria is straightforward with gradient-based methods. However, as is frequently the case, in real life the situation is often more complicated. In order to solve Nash equilibria for difficult objective functions, Sefrioui and Périaux introduced the *Nash genetic algorithm* [107].

Being a multi-population algorithm, it shares similarities with the island model genetic algorithms. However, there are several notable differences. Each of the objective function is assigned with a player and a population. Players are allowed to manipulate only the design parameters assigned to them. This is done using standard genetic operators. After each epoch, such as one generation, the players exchange their elite individual values. The algorithm converges into a Nash equilibrium. The migration process is illustrated in Figure 8.

In order to demonstrate the Nash approach, let us consider the following simple multi-objective problem:

$$\left\{ \begin{array}{rcl} f_1(x_1, x_2) & = & (x_1 - 1)^2 + (x_1 - x_2)^2 \\ f_2(x_1, x_2) & = & (x_2 - 3)^2 + (x_1 - x_2)^2 \end{array} \right. \tag{15}$$

The Nash equilibrium can be found analytically using *rational sets*. Let D_1 be the rational set of Player 1 and D_2 be the rational set of Player 2:

$$\begin{array}{llcl} D_1(\bar{x}_1, x_2) \in \bar{S}_1 \times \bar{S}_2 & \text{such that} \quad f_1(\bar{x}_1, x_2) & \leq & f_1(x_1, x_2) \\ D_2(x_1, \bar{x}_2) \in \bar{S}_1 \times \bar{S}_2 & \text{such that} \quad f_2(x_1, \bar{x}_2) & \leq & f_2(x_1, x_2) \end{array} \tag{16}$$

The reaction sets consist of solutions that satisfy the following conditions

$$\left\{ \begin{array}{rcl} D_1 & = & \left\{ x_1 \left| \frac{\partial f_1(x_1, x_2)}{\partial x_1} = 0 \right. \right\} \\ D_2 & = & \left\{ x_2 \left| \frac{\partial f_2(x_1, x_2)}{\partial x_2} = 0 \right. \right\} \end{array} \right. \tag{17}$$

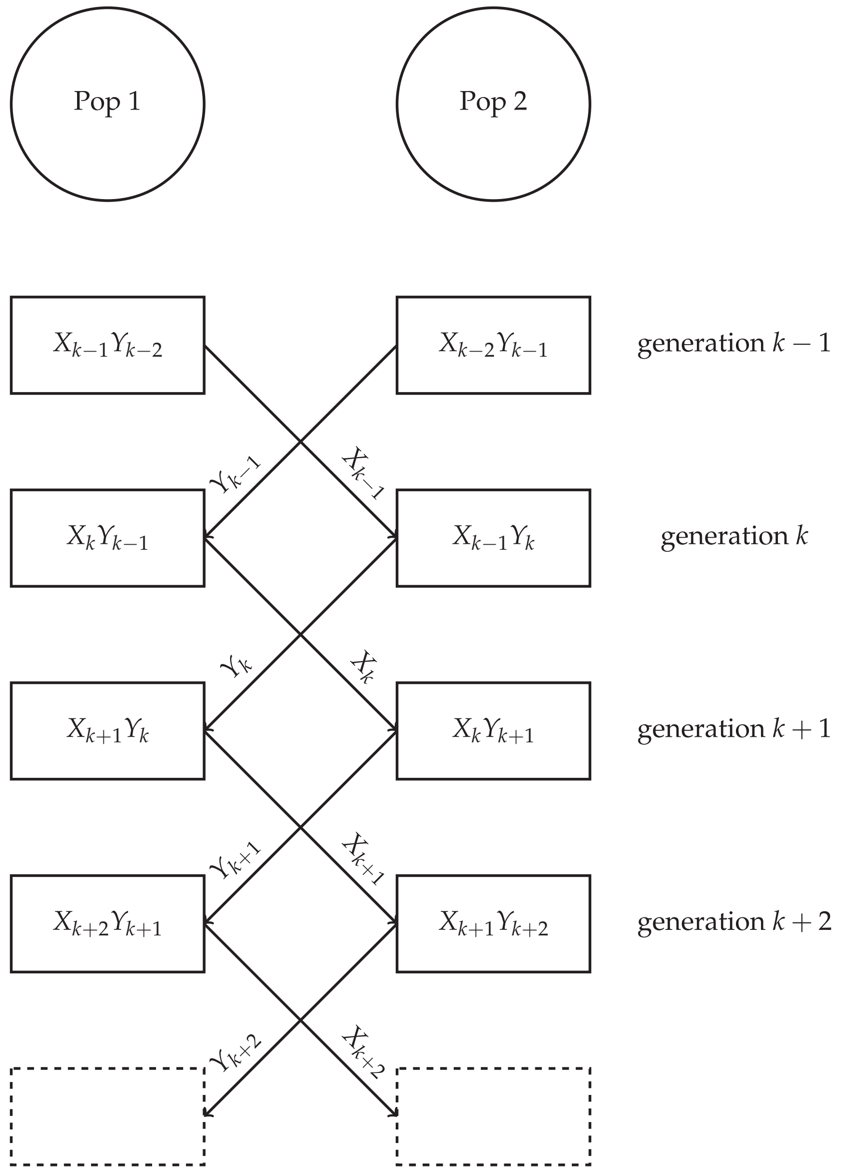


FIGURE 8 Migration process in Nash genetic algorithms. Player 1, who controls population 1, sends its elite values X_k of generation k to Player 2, and vice versa.

By applying (17) to the original system of equations (15), the rational reaction sets can be computed:

$$\begin{array}{lllll} \frac{\partial f_1(x_1,x_2)}{\partial x_1} = 0 & \Leftrightarrow & 2(x_1-1)+2(x_1-x_2)=0 & \Leftrightarrow & x_2 = 2x_1 - 1 \\ \frac{\partial f_2(x_1,x_2)}{\partial x_2} = 0 & \Leftrightarrow & 2(x_2-3)-2(x_1-x_2)=0 & \Leftrightarrow & x_2 = \frac{x_1+3}{2} \end{array} \tag{18}$$

The Nash equilibrium is the intersection of the two sets. Solving (18) yields the result $x_1 = \frac{5}{3}$ and $x_2 = \frac{7}{3}$. The Nash equilibrium exists and is unique, $(x_1, x_2) = (\frac{5}{3}, \frac{7}{3})$. The corresponding solution is $(f_1, f_2) = (\frac{8}{9}, \frac{8}{9})$.

The validity of the Nash approach is tested using differential evolution with a population size of $NP = 10$ per player. The elite values are listed in Figure 9. The algorithm is able to converge into the Nash equilibrium successfully in a few dozen generations.

3.3 Virtual Nash algorithms

In some cases, the objective function can be divided into subfunctions so that

$$f(\mathbf{x}) = \sum_{i=1}^{N} f_i(\mathbf{x}_i) \tag{19}$$

These functions function are called *(linearly) separable*. Separated functions can be divided into a set of subfunctions forming a *"virtual" multi-objective problem*. If

$$f(\mathbf{x}) = 0, \ \mathbf{x} = 0 \tag{20}$$

is the global minimum of the function, and

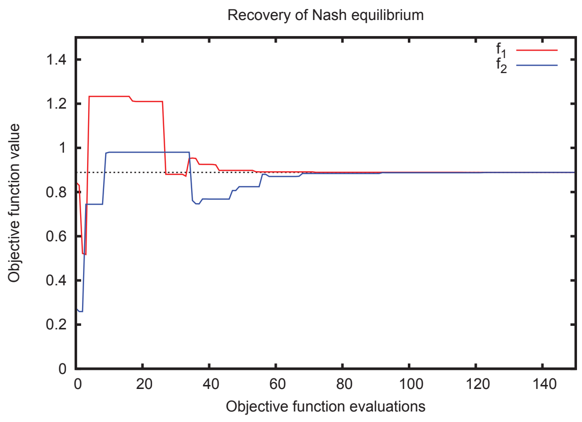
$$\min f(\mathbf{x}) = \sum_{i=1}^{N} \min f_i(\mathbf{x}) = 0 \tag{21}$$

it is known that the global Nash equilibrium is located there [14].

Since the search space depends on the dimensionality of the problem, it is apparent that the complexity of the problem can be reduced if each function is solved separately because separating the search space restricts the movement of the algorithm. One effect of the reduced movement is that the number of possible local optima where the algorithm can get stuck is reduced. The idea of handling a single-objective problem using multi-objective methods is explored for example in [65, 60].

3.3.1 Mathematical examples

To illustrate the effect of the Nash approach, it is tested on selected additively separable mathematical functions listed below (the functions are illustrated in Figure 10). The number of dimensions is $n_{dim} = 12$ allowing five different Nash



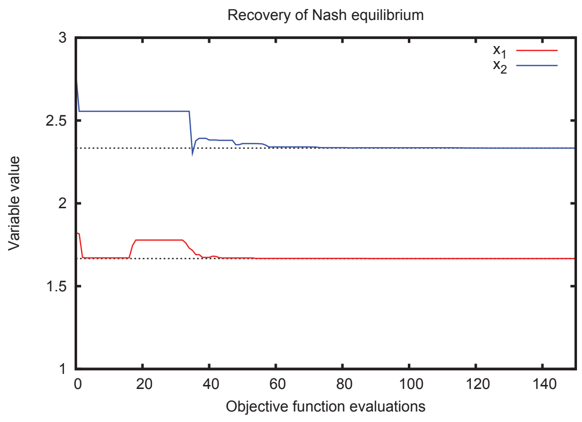


FIGURE 9 Convergence of the differential evolution algorithm into the Nash equilibrium $(f_1, f_2) = (\frac{8}{9}, \frac{8}{9})$, $(x_1, x_2) = (\frac{5}{3}, \frac{7}{3})$.

TABLE 2 Selected parameter values for the jDE algorithm.

population size	NP	$\max\left(\frac{5n_{dim}}{n_{players}}, 15\right)$
mutation factor	F	[0.1, 1.0]
crossover rate	CR	[0.0, 1.0]
F replacement probability	τ_1	0.1
CR replacement probability	τ_2	0.1

decompositions, $n_{players} = \{2, 3, 4, 6, 12\}$. A differential evolution variant with adaptive control parameters is used (jDE, described in the Section 2.3.2). The parameters listed in Table 2 are used for each test function. The algorithms are terminated when the best value is within the machine precision from the optimum, or when the maximum number of combined objective function evaluations $n_{it} = 10^5$ is reached. The algorithms are run in parallel with $n_{players}$ controlling processes and a total of 12 slave processes. In order to prevent premature convergence due to the loss of diversity, the non-elite individuals of the population are disturbed using Gaussian mutation if the normalized standard deviation of the fitness values of the population has decreased below the given threshold of $\sigma_{div} = 10^{-1}$.

De Jong's sphere function. One of the simplest test functions,

$$f_a(\mathbf{x}) = \sum_{i=1}^{n_{ndim}} x_i^2, x_i \in [-5.12, 5.12], \tag{22}$$

is both smooth and has only one global minimum, $x_i = 0, i = 1, \ldots, n_{dim}$ [20].

Rastrigin's function. The function

$$f_b(\mathbf{x}) = 10n_{dim} + \sum_{i=1}^{n_{dim}} \left(x_i^2 - 10\cos(2\pi x_i)\right), x_i \in [-5.12, 5.12] \tag{23}$$

is highly multi-modal, and there is a total of $11^{n_{dim}}$ local optima [85]. The global optimum $f(\mathbf{x}) = 0$ is located at $x_i = 0, i = 1, \ldots, n_{dim}$.

Schwefel's function. The function

$$f_c(\mathbf{x}) = 418.9829n_{dim} + \sum_{i=1}^{n_{dim}} x_i \sin\left(\sqrt{|x_i|}\right), x_i \in [-500, 500] \tag{24}$$

has several deceptive local minima [105]. The global optimum $f(\mathbf{x}) = 0$ is located far from the origin at $x_i = -420.9687, i = 1, \ldots, n_{dim}$.

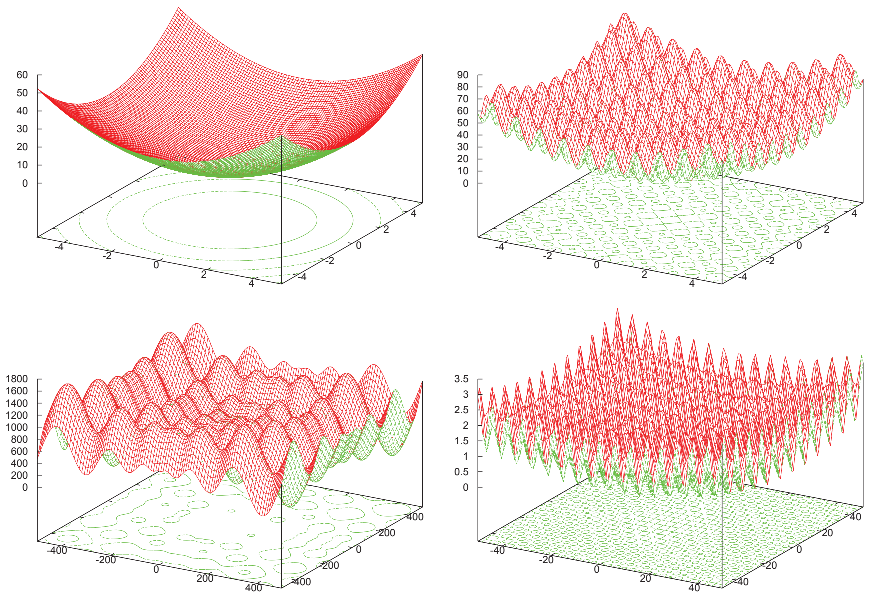


FIGURE 10 Test function landscapes ($n_{dim} = 2$). *Top left:* De Jong's sphere function. *Top right:* Rastrigin's function. *Bottom left:* Schwefel's function. *Bottom right:* Griewank's function (close-up).

Griewank's function. In order to test the efficiency of the methods in the case of nonseparable functions, the Griewank function was included [47].

$$f_d(\mathbf{x}) = 1 + \sum_{i=1}^{n_{dim}} \frac{x_i^2}{4000} - \prod_{i=1}^{n_{dim}} \cos\left(\frac{x_i}{\sqrt{i}}\right), x_i \in [-600, 600] \tag{25}$$

It is a highly multimodal function. The product introduces dependency between the variables. For multi-objective cases, the subfunctions were formulated as

$$f_{d_k}(\mathbf{x}_k) = \sum_{i \in I_k} \frac{x_i^2}{4000} + \frac{1}{n_{pl}} \left(1 - \prod_{i=1}^{n_{dim}} \cos\left(\frac{x_i}{\sqrt{i}}\right)\right) \tag{26}$$

where k is the index of the player, I_k is the set of design variable indexes that the player is allowed to operate, $n(I_k) = \frac{n_{dim}}{n_{pl}}$. The global optimum $f(\mathbf{x}) = 0$ is located at the origin, $x_i = 0, i = 1, \ldots, n_{dim}$.

The averaged results for 10 runs are listed in Tables 3–6 and the convergence curves are shown in Figure 11. Based on the results, it is evident that the "multi-objectivization" of the functions has a strongly positive effect on algorithmic convergence, accuracy and robustness. In the case of separable functions, every instance managed to find the optimum. Increasing the number of players (and thus decreasing the dimensionality) reduced the average algorithmic convergence rate in each case. Only in the case of Schwefel's function the six-player case had some difficulties in converging. As expected, the Griewank function

TABLE 3 Results for the algorithms in the case of De Jong's function. The number of players, the best and the worst objective function values and the total number of objective function evaluations are listed with the mean final objective function values and corresponding standard deviations. The number of successful runs is also listed.

De Jong					
n_{pl}	f_{min}	f_{max}	f_{mean}	σ_f	
1	8.9491E-61	1.4950E-54	1.8725E-55	1.9922E-108	
2	1.9956E-92	3.7196E-84	4.1370E-85	1.2206E-167	
3	3.0678E-121	2.5941E-109	2.5954E-110	6.0557E-218	
4	5.6038E-151	8.0443E-120	8.0443E-121	5.8240E-239	
6	2.8402E-150	7.7971E-135	8.6737E-136	5.3964E-269	
12	7.6899E-161	1.6489E-135	1.7071E-136	2.4308E-270	
	it_{min}	it_{max}	it_{mean}	σ_{it}	success
1	18276	19795	1.8807E+04	4.9446E+02	10/10
2	11479	15051	1.3181E+04	1.0573E+03	10/10
3	9049	10945	9.9960E+03	6.8587E+02	10/10
4	7430	10669	8.8830E+03	9.6376E+02	10/10
6	7329	8854	8.0620E+03	5.7772E+02	10/10
12	6326	7334	6.8450E+03	3.8969E+02	10/10

was the most difficult to solve. Compared to the Nash algorithms, each of which managed to converge at least once, the standard approach failed to converge to the global optimum. The 12-player algorithm was the most robust, converging most often.

TABLE 4 Results for the algorithms (Rastrigin's function).

Rastrigin					
n_{pl}	f_{min}	f_{max}	f_{mean}	σ_f	
1	0.0000E+00	0.0000E+00	0.0000E+00	0.0000E+00	
2	0.0000E+00	0.0000E+00	0.0000E+00	0.0000E+00	
3	0.0000E+00	0.0000E+00	0.0000E+00	0.0000E+00	
4	0.0000E+00	0.0000E+00	0.0000E+00	0.0000E+00	
6	0.0000E+00	1.2434E-14	1.9540E-15	1.4798E-28	
12	0.0000E+00	5.4179E-13	5.5245E-14	2.6308E-25	
	it_{min}	it_{max}	it_{mean}	σ_{it}	success
1	32373	45603	3.8334E+04	3.4432E+03	10/10
2	20705	26049	2.3317E+04	1.7828E+03	10/10
3	14257	21165	1.8295E+04	2.3320E+03	10/10
4	12518	17619	1.5363E+04	1.5529E+03	10/10
6	12150	23455	1.5205E+04	3.1932E+03	10/10
12	9302	11868	1.0957E+04	7.2137E+02	10/10

TABLE 5 Results for the algorithms (Schwefel's function).

Schwefel					
n_{pl}	f_{min}	f_{max}	f_{mean}	σ_f	
1	0.0000E+00	0.0000E+00	0.0000E+00	0.0000E+00	
2	0.0000E+00	0.0000E+00	0.0000E+00	0.0000E+00	
3	0.0000E+00	0.0000E+00	0.0000E+00	0.0000E+00	
4	0.0000E+00	0.0000E+00	0.0000E+00	0.0000E+00	
6	0.0000E+00	5.6843E-14	1.1369E-14	5.1698E-27	
12	0.0000E+00	1.7053E-13	2.8422E-14	2.7465E-26	
	it_{min}	it_{max}	it_{mean}	σ_{it}	success
1	23801	27459	2.5589E+04	1.2613E+03	10/10
2	14874	18950	1.6635E+04	1.5429E+03	10/10
3	10683	13444	1.2266E+04	9.7536E+02	10/10
4	9632	15005	1.1018E+04	1.4224E+03	10/10
6	9551	19521	1.2675E+04	3.4319E+03	10/10
12	8054	10378	9.0150E+03	6.7358E+02	10/10

TABLE 6 Results for the algorithms (Griewank's function).

Griewank					
n_{pl}	f_{min}	f_{max}	f_{mean}	σ_f	
1	2.7134E-03	2.6325E-02	1.4776E-02	6.4229E-04	
2	4.3891E-47	4.0681E-02	1.3768E-02	1.6092E-03	
3	2.5541E-55	2.5682E-02	6.3574E-03	7.5716E-04	
4	4.6981E-93	2.8089E-02	1.1327E-02	1.1476E-03	
6	2.1575E-94	2.6974E-02	1.1129E-02	1.4481E-03	
12	9.2516E-148	2.4820E-02	4.6771E-03	6.5575E-04	
	it_{min}	it_{max}	it_{mean}	σ_{it}	success
1	100000	100000	1.0000E+05	0.0000E+00	0/10
2	57118	100000	9.5711E+04	1.2865E+04	1/10
3	50704	100000	8.6121E+04	1.8138E+04	4/10
4	35293	100000	7.9561E+04	2.6339E+04	4/10
6	29653	100000	6.7449E+04	3.2792E+04	5/10
12	11243	100000	4.1567E+04	3.8487E+04	7/10

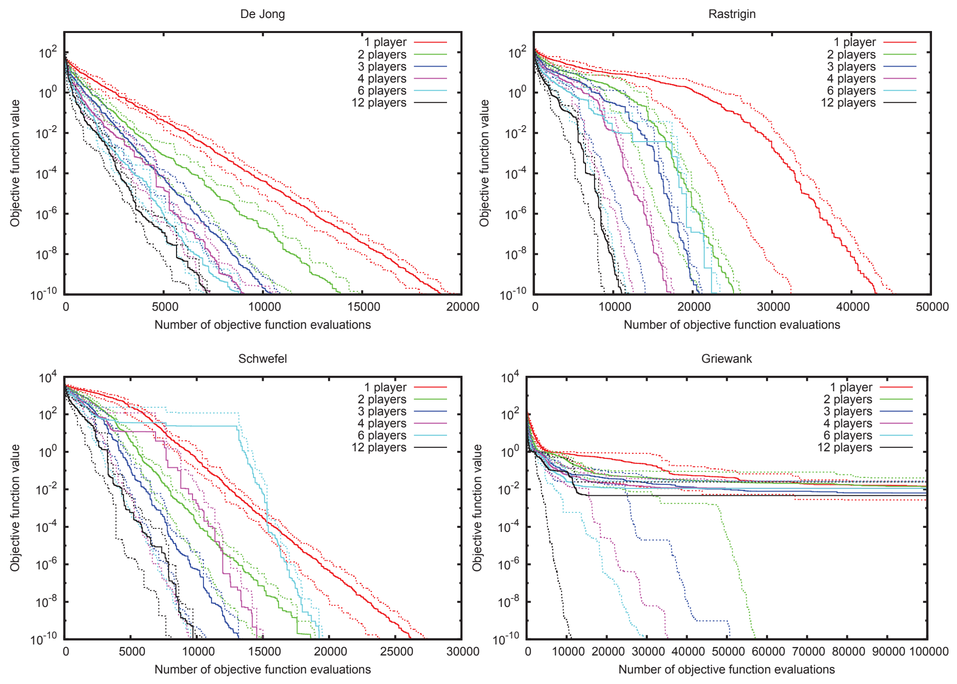


FIGURE 11 Convergence of the algorithms. The dotted lines depict the best and the worst values at the corresponding objective function iteration.

4 MODEL PROBLEMS

In this chapter, the optimization test cases used in this study are described. Most of them are simple academic geometry reconstruction problems from more realistic cases of aeronautical interest. The first and the second test case are available in the Finnish Design Test Case Database[1]. All the test cases have been developed by the author.

4.1 Inverse or optimization problems for multiple (ellipse) ellipsoid configurations

This academic test case[2] was developed in order to study algorithmic convergence by splitting the inverse problem (recovery of the target pressure on a surface) into smaller subproblems. It also provides a way to study the behavior of algorithms with meshes of different quality. Finally, it can be expanded into a simple test platform for multiphysics optimization (computational fluid dynamics, computational electromagnetism, and aeroacoustics), both in 2D and 3D.

The test case includes three different problems: an aerodynamic reconstruction problem, a radar wave problem, and an aeroacoustics problem. The aerodynamic reconstruction problem, which was used in this work, consists of recovery of the original position of two ellipse (2D) or ellipsoid (3D) elements using potential, Euler, or Navier-Stokes flows for $Re = 100$ and $Re = 500$.

The computational domain and geometry are defined by the parameters listed in Table 7. The geometry is shown in Figure 12. The allowed ranges of design variables are listed in Table 8.

The flow is described by incompressible laminar flow. At upstream entrance the flow has the velocity components $v_x = \cos(\alpha)$ and $v_y = \sin(\alpha)$, where $\alpha =$

[1] The database with test cases and computed results is available to the public at the address <http://jucri.jyu.fi>.

[2] Full definition of the test case TA2 available at the address <http://jucri.jyu.fi/?q=testcase/5>.

TABLE 7 Parameters for the geometry and computational domain.

s_1	=	40	height (and width) of the bounding box
s_2	=	80	length of the bounding box
(x_0, y_0, z_0)	=	$(-30, -20, -20)$	front lower left corner of the bounding box
l_1	=	2.0	length of the ellipse/ellipsoid 1
h_1, w_1	=	0.5	height (and width) of the ellipse/ellipsoid 1
x_1, y_1, z_1	=	$(-7.0, -0.5, 0.0)$	reference position of the ellipse/ellipsoid 1
α_1	=	$-3.0°$	reference angle of the ellipse/ellipsoid 1 (clockwise)
l_2	=	10.0	length of the ellipse/ellipsoid 2
h_2, w_2	=	1.0	height (and width) of the ellipse/ellipsoid 2
x_2, y_2, z_2	=	$(0.0, 0.0, 0.0)$	position of the ellipse/ellipsoid 2
α_2	=	$0.0°$	angle of the ellipse/ellipsoid 2 (clockwise)
l_3	=	3.5	length of the ellipse/ellipsoid 3
h_3, w_3	=	0.5	height (and width) of the ellipse/ellipsoid 3
x_3, y_3, z_3	=	$(7.5, -0.5, 0.0)$	reference position of the ellipse/ellipsoid 3
α_3	=	$3.0°$	reference angle of the ellipse/ellipsoid 3 (clockwise)

TABLE 8 Allowed ranges for design variables for the three-ellipse element geometry reconstruction problem.

	x_s	y_s	α_s [°]	x_f	y_f	α_f [°]
min	-10.00	-1.50	-10.00	7.25	-1.50	0.00
max	-6.50	0.00	0.00	10.00	0.00	10.00
target	-7.00	-0.50	-3.00	7.50	-0.50	3.00

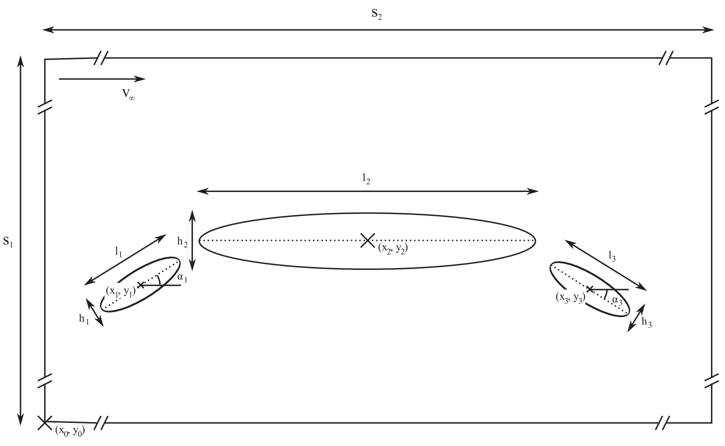


FIGURE 12 Three-ellipse element geometry.

5.0° is the angle of attack. At downstream exit there are free boundary conditions and on the surfaces of the ellipse/ellipsoid elements, no-slip conditions apply. The kinematic viscosity $\nu = \frac{1}{10}$ or $\nu = \frac{1}{50}$, corresponding the Reynolds number values $Re = 100$ and $Re = 500$.

The objective function is defined as the L^2 error norm of the computed and reference surface pressure,

$$J_i(\mathbf{x}_i) = \int_{\Gamma_i} \left| \mathbf{p}_i - \mathbf{p}_i^{target} \right|^2 \tag{27}$$

where $\mathbf{x}_i$ is the local design vector of the ellipse/ellipsoid $i = 1, \ldots, 3$ and $\mathbf{p}$ and $\mathbf{p}^{target}$ are the computed and target surface pressure distributions. Γ_i is the boundary of an ellipse element. The global objective function is the sum of the subfunctions,

$$J(\mathbf{x}) = \sum_{i=1}^{3} J_i(\mathbf{x}_i) \tag{28}$$

where the vector $\mathbf{x}_i = \{x_i, y_i, \alpha_i\}$ refers to the design parameter of the ellipse element $i = 1, \ldots, 3$.

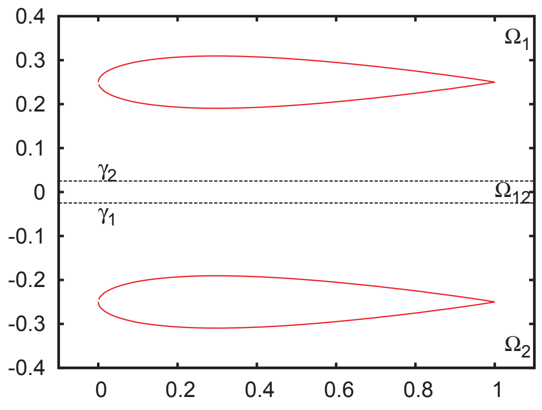


FIGURE 13 Bi-NACA0012 airfoil geometry with example domain decomposition.

TABLE 9 Allowed ranges for design variables for the Bi-NACA0012 geometry reconstruction problem.

		min	max	target
upper	x_u	-1.00	-0.50	-0.50
airfoil	y_u	0.15	0.35	0.25
lower	x_ℓ	-0.50	0.00	-0.50
airfoil	y_ℓ	-0.35	-0.15	-0.25

4.2 Reconstruction of BINACA0012 geometry using discrete and continuous optimization

This test case[3] consists of the recovery of the positions of two stacked NACA0012 airfoil geometries using (1) discrete and (2) continuous search space (Figure 13). In this work, the continuous case was studied. The distance of the airfoil is 0.5 chord lengths measured at the leading edge.

The original test case was introduced in [23]. The challenge of the geometry is that the proximity of the airfoils induces a shock between them and the solver and mesh quality must be good enough to capture it. When a non-zero angle of attack is introduced, an additional shock appears above the upper airfoil.

The target of the problem is to recover the original positions of the airfoils by minimizing the pressure difference between the computed and target pressure. The objective function is

$$\begin{aligned} J_u\left(x_u, y_u\right) &= \tfrac{1}{2}\int_{\Gamma_1}\left|\mathbf{p}_u - \mathbf{p}_u^{target}\right|^2 \\ J_\ell\left(x_\ell, y_\ell\right) &= \tfrac{1}{2}\int_{\Gamma_\ell}\left|\mathbf{p}_\ell - \mathbf{p}_\ell^{target}\right|^2 \end{aligned} \tag{29}$$

and

$$J = J_u + J_\ell \tag{30}$$

where $\Gamma_{u,\ell}$ are the solid boundaries of the airfoils and $\mathbf{p}_{u,\ell}$ and $\mathbf{p}_{u,\ell}^{target}$ are the respective pressure distributions.

The continuous problem consists of four design parameters (coordinates of the leading edges of the airfoils). The target positions and allowed ranges are listed in Table 9.

The test case can be run using two flow conditions. In the first case, the angle of attack $\alpha = 0.0°$, free-flow Mach number $M_\infty = 0.55$ and Reynolds number $Re = 5000$. A shock should form between the airfoils. In the second case, the angle of attack is increased to $\alpha = 6.0°$, resulting in a more complex pressure distribution. In the case of Euler flow, slip boundary conditions apply on the solid boundaries and in the case of Navier–Stokes flow, no-slip boundary conditions apply. In the far field, non-reflection boundary conditions apply.

4.2.1 Shape reconstruction problem

In Paper III, the test case was expanded into a shape reconstruction problem where (1) the positions were kept constant, the airfoils were deformed and (2) the airfoils were allowed to both move and deform. The shapes of the airfoils were parameterized using Bézier curves (Figure 14). The thicknesses of the airfoils were modified using $n - 2$ control points for each airfoil, where n is the

[3] Full definition of the test case TA10 is available at the address <http://jucri.jyu.fi/?q=testcase/49>.

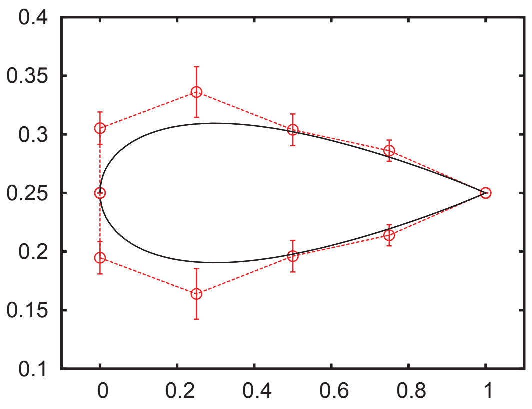


FIGURE 14 An example 6th order Bézier curve parameterization (four control parameters for symmetrical airfoil).

TABLE 10 Allowed ranges for design variables for the multi-disk element geometry reconstruction problem. The positions are described using polar coordinates. Index $k = \{1, \dots, n_c\}$, where n_c is the number of disks. The radii are $s_0 = 0.5$ for the central element, $s_k = 0.125$ for the surrounding elements, and $s_\infty = 20$ for the far field boundary.

	r_k	θ_k
target	2.0	$-k\frac{2\pi}{n_c} - \frac{\pi}{n_c}$
min	0.6375	$\theta_k^{target} - \frac{\pi}{4n_c}$
max	2.5	$\theta_k^{target} + \frac{\pi}{4n_c}$

order of the Bernstein polynomial. The control points were allowed to move in y-direction.

4.3 Multi-disk element geometry

This purely academic problem was developed in order to test the efficiency of the global Nash method described in detail in Chapter 5. The problem consists of a disk element (with the radius $s_0 = 0.5$) located at the origin. It is surrounded by $n_c = 2, \dots, 6$ smaller disk elements ($s_k = 0.125$, Figure 15). Except for the central element, the elements are allowed to move freely within their confined spaces. The range of movement depends on the number of elements. The allowed ranges and target positions are listed in Table 10.

The target is to minimize the difference between the prescribed and the cal-

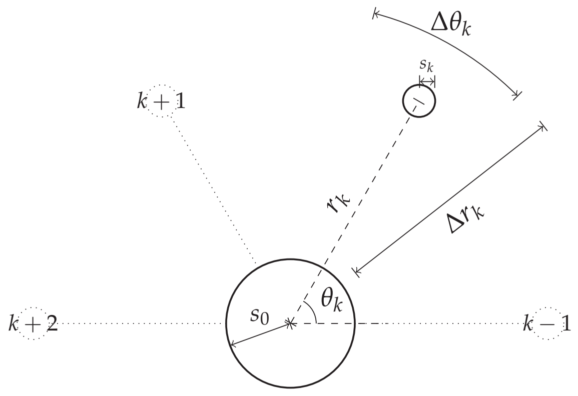


FIGURE 15 Multi-disk element geometry parameterization. Element k can move freely within the region defined by Δr_k and $\Delta\theta_k$.

culated surface pressure distributions:

$$J_k\left(r_k, \theta_k\right) = \frac{1}{np_{tot}} \sum_{j=0}^{np_k} \left(\mathbf{p}_j - \mathbf{p}_j^{target}\right)^2 \tag{31}$$

where np_k is the number of surface pressure points of the element k and np_{tot} is the total number of pressure points. The global objective function is the sum of subfunctions,

$$J\left(\mathbf{r}, \boldsymbol{\theta}\right) = \sum_{k=1}^{NP} J_k\left(r_k, \theta_k\right) \tag{32}$$

The arrangement of the disks allows the study of different domain and geometry decompositions. The problem can be made more challenging by introducing element deformation.

5 NASH GAMES COALITIONS

In this chapter, which forms the core of the thesis, the Nash approach is applied to distributed shape optimization. The methods are described and performance increases due to splitting the shape reconstruction problem using the virtual Nash approach illustrated using academic test problems. Nash games employing domain and geometry decomposition are combined into a single global game as a "distributed one-shot" method, which is validated using the Bi-NACA0012 geometry.

5.1 Domain decomposition using virtual Nash approach

Domain decomposition can be seen as an inverse problem in which the objective function is the discrepancy between the subdomain state solutions in the subdomains of the overlapping area,

$$JF\left(\mathbf{g}^{(1)},\mathbf{g}^{(2)}\right) = \left\|\varphi_1(\mathbf{g}^{(1)}) - \varphi_2(\mathbf{g}^{(2)})\right\| \tag{33}$$

where $\|\cdot\|$ is the appropriate norm and φ_1 and φ_2 are the solutions of Ω_1 and Ω_2 in the overlapping domain Ω_{12}, respectively (Figure 16). The vectors $\mathbf{g}^{(1)}$ and $\mathbf{g}^{(2)}$ refer to Dirichlet boundary condition values on the interface boundaries. The vectors are modified by the flow optimizer. The objective function JF tends to zero once the state solution in the global domain has been successfully reconstructed.

This problem can be solved using the virtual Nash approach where the problem consists of two functions [95],

$$\begin{aligned}\min_{\mathbf{g}^{(1)}} JF_1\left(\mathbf{g}^{(1)},\bar{\mathbf{g}}^{(2)}\right) &= \left\|\varphi_1(\mathbf{g}^{(1)}) - \varphi_2(\bar{\mathbf{g}}^{(2)})\right\| \\ \min_{\mathbf{g}^{(2)}} JF_2\left(\bar{\mathbf{g}}^{(1)},\mathbf{g}^{(2)}\right) &= \left\|\varphi_1(\bar{\mathbf{g}}^{(1)}) - \varphi_2(\mathbf{g}^{(2)})\right\|\end{aligned} \tag{34}$$

The most straightforward way to select $g_i^{(1,2)}$ is to use the values of φ on the boundary nodes (if P1 finite elements are used). However, this leads to a

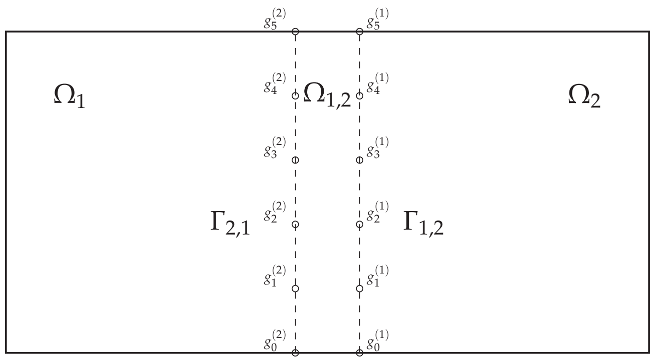

Ω_1
$\Omega_{1,2}$
Ω_2
$\Gamma_{2,1}$
$\Gamma_{1,2}$
$g_5^{(2)}$ $g_4^{(2)}$ $g_3^{(2)}$ $g_2^{(2)}$ $g_1^{(2)}$ $g_0^{(2)}$
$g_5^{(1)}$ $g_4^{(1)}$ $g_3^{(1)}$ $g_2^{(1)}$ $g_1^{(1)}$ $g_0^{(1)}$

FIGURE 16 Domain decomposition using Nash games. The overlap is minimized by modifying the boundary point values $g_i^{(1,2)}$.

high-dimensional problem which is slow to solve. In [95] an internal flow with a parabolic profile in a nozzle is studied. The simple problem allowed parameterization using only one control variable. In complex external flow cases with non-trivial geometries this kind of simplification is not possible. A more sophisticated approach would be to apply an optimizer to improve the convergence of a standard domain decomposition method.

5.2 Nash games and geometry decomposition

The virtual Nash approach is well-suited for distributed shape optimization. The decision vector can be split between the players, making it a "geometry decomposition" method. By dividing the geometry into a set of subgeometries, the dimensionality of the problem can be reduced, leading to improved algorithmic convergence. In addition, the virtual games can be used for parallelization.

The efficiency of the method is strongly coupled to the geometry of the problem. A poor choice can easily lead the algorithm into a deceptive minimum and, therefore, prevent it from converging.

5.2.1 Geometry decomposition example

To illustrate the effect of a selected geometry decomposition, a simplification of the Bi-NACA0012 geometry is tested. Instead of NACA0012 airfoils, the shapes of two elliptical elements are recovered in a potential flow with an angle of attack of $\alpha = 6.0°$. The following geometry decompositions were tested (Figure 17):

1. Standard approach: no geometry decomposition.
2. The ellipses are optimized separately, one player for each element.
3. Player 1 optimizes the extrado and Player 2 the intrado surface.

TABLE 11 Results for the different geometry decompositions; n_{it} is the number of objective function evaluations and t is the wall-clock time in seconds.

case	n_{it}	t
1	3553	449.02s
2	4326	317.29s
3	2551	151.02s
4	3447	228.84s
5	1511	88.75s
6	1768	108.47s
7	2185	140.26s

4. The control points alternate between the players. Player 1 operates the odd and Player 2 the even control points.
5. Four players. Each player optimizes one side of an ellipse.
6. Four players. Player 1 optimizes the leading side of the upper ellipse and Player 2 the trailing side and so on.
7. Four players, alternating control points.

All cases are tested using the jDE algorithm with populations of $5 \times n_{dim}$ individuals where n_{dim} is the number of design variables per player ($n_{dim} = \{16, 8, 4\}$). The algorithms are terminated when the threshold objective function value of $f_{min} = 10^{-3}$ is reached. In order to ensure fair comparison, all algorithms use equal number of slave processors (a total of 24). The results are listed in Table 11 and the convergence curves in Figure 18. The example results are illustrated in Figure 19.

It can be seen from the results that in most of the cases geometry decomposition improves convergence. Although the standard approach spent most wall-clock time, case (2) required the largest number of objective function evaluations, suggesting that the simplest geometry decomposition does not produce an algorithmic speed-up in this case. However, other decompositions are clearly better, case (5) being the most efficient.

5.3 Global Nash Game Coalition Algorithm

Geometry and domain decomposition using the Nash approach can be combined. The idea of *hierarchical Nash games* [95] is the following. As described in the previous section, a flow can be reconstructed using a Nash game. Similarly, using geometry decomposition a shape can be reconstructed as a Nash game. Each time the geometry players produce a new candidate geometry, the flow players repair the flow. In the game theoretical sense, geometry reconstruction can be considered as the leader in the Stackelberg game, and flow reconstruction as the

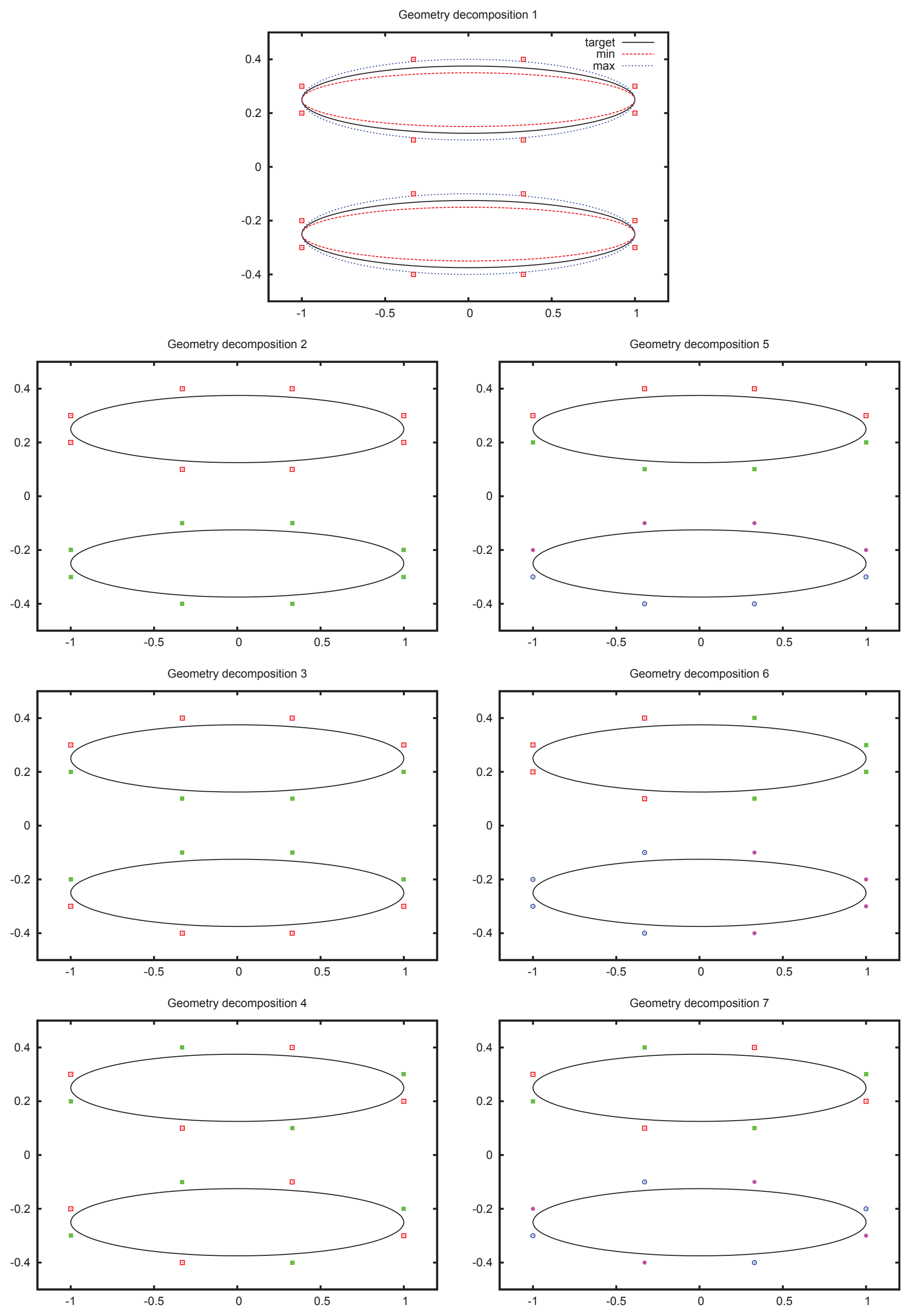
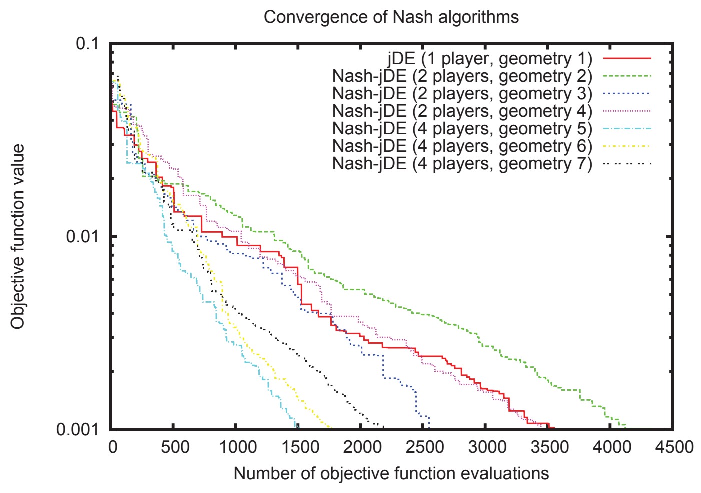


FIGURE 17 Different geometry decompositions. Single player: (1) No geometry decomposition. Two players: (2) Separate surfaces. (3) Extrado/intrado surfaces. (4) Alternating control points. Four players: (5) Top/bottom surfaces. (6) Leading/trailing surfaces. (7) Alternating control points.



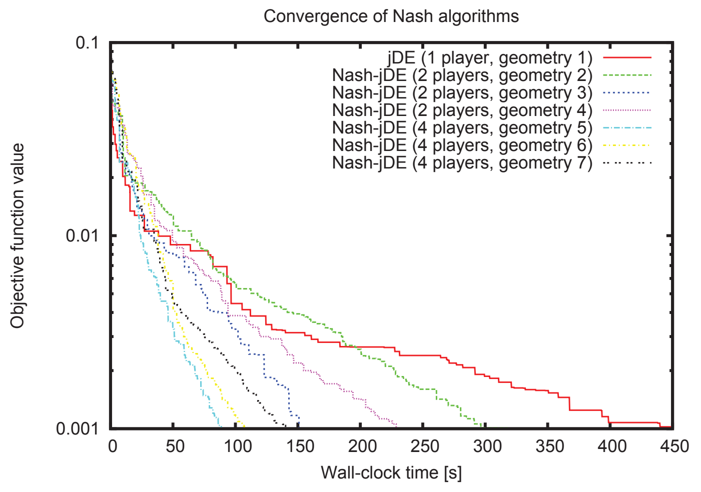


FIGURE 18 Convergence of the algorithms in the different geometry decomposition cases.

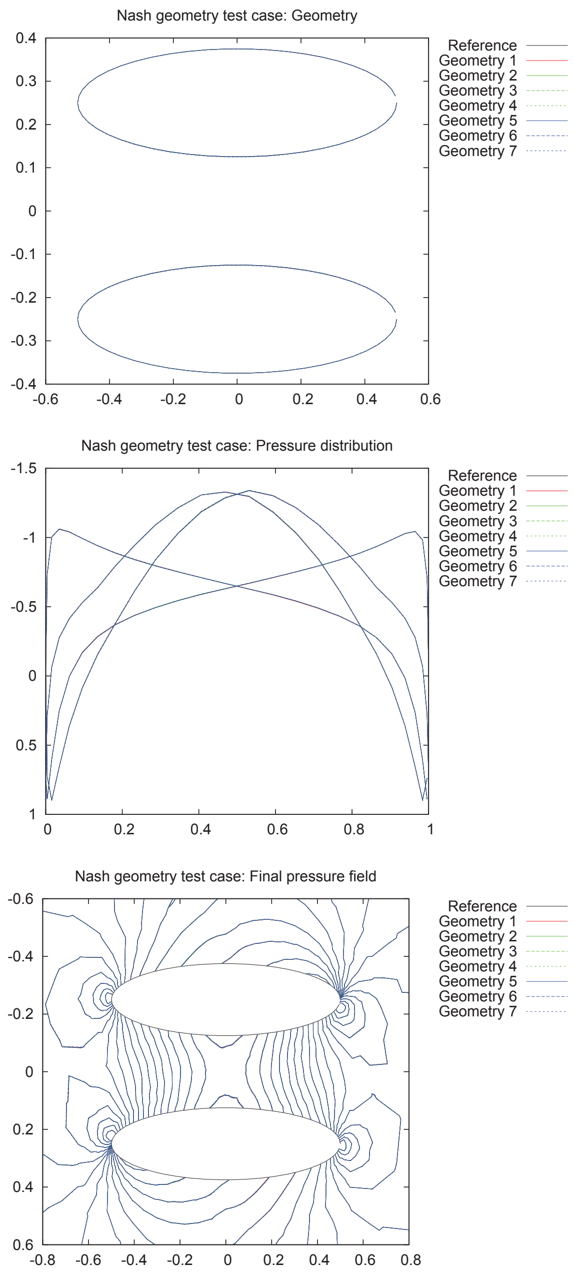


FIGURE 19 Example final results for geometry decomposition.

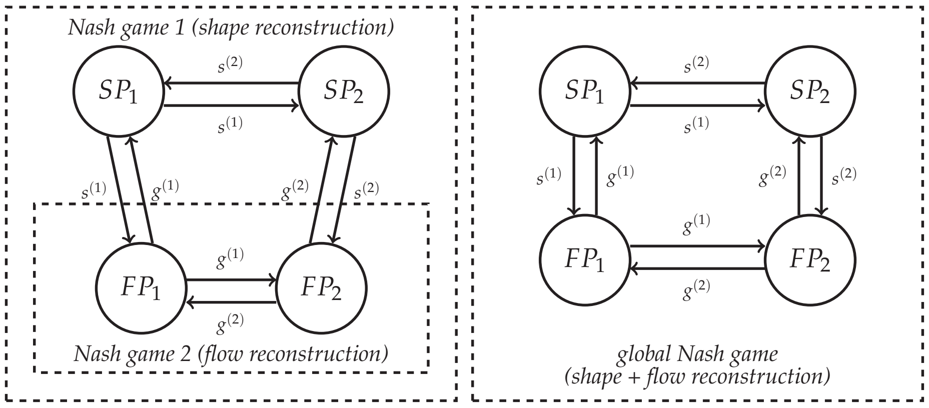


FIGURE 20 Hierarchical and global Nash game structures. In the case of hierarchical game, the flow reconstruction is done completely by flow players FP_i for each shape candidate generated by the shape players SP_k. In the global case, all players act simultaneously.

follower that reacts to the changes of the leader. The hierarchical Nash game structure is illustrated in Figure 20.

The hierarchical Nash game can be seen as an enhancement to the traditional approach, where optimization is done by distributing the evaluation of the objective function by means of domain decomposition. In both cases, the flow has to be completely reconstructed for each new geometry candidate. This sequential process imposes a serious bottleneck.

In order to avoid reconstructing the flow field completely for each new geometry candidate, a new method is introduced in this study. In the proposed method, the flow and geometry players operate simultaneously in a single global Nash game [75, 76]. Unlike in the hierarchical approach, the flow does not need to be completely reconstructed when the geometry is being optimized. In this sense, the method resembles one-shot methods.

The proposed method, the *global Nash game coalition algorithm* (GNGCA), which forms the core of this thesis, operates in the following manner. The domain is divided into two or more subdomains. The geometry to be optimized is distributed to the Nash shape players. Simultaneously, the flow players start to minimize the discrepancy on the overlapping areas. After each epoch, the shape players distribute their elite geometry information and flow players their updated boundary values. In this work, each subdomain has one or more Nash players. Cases where the subgeometries span two or more subdomains were not studied. In such situations, local flow could be solved with the help of standard domain decomposition methods or a Nash game leading to a locally hierarchical approach. The process is illustrated in Figure 20.

Although standard one-shot methods converge very quickly, GNGCA has several advantages. First, the proposed method is inherently parallel. The flow and shape players can be implemented on different CPU cores, computers, or computing clusters. Second, the method inherits advantages from domain decomposition methods. Instead of computing the whole domain, computation can be limited to a local subdomain for geometry players. Finally, the approach is very flexible when it comes to selecting the algorithms. For simple linear elliptic state equations, standard domain decomposition methods can be applied; for nonlinear cases, local or global search methods can be used. If the objective function is relatively simple and smooth, local search methods can be applied; global methods can be used in more difficult cases where standard one-shot methods cannot produce good results.

Interaction between shape and flow players is limited because flow information is only updated after each epoch. This is important in the cases where minimization of information transfer is critical. Examples of such situations include distributed computing over large distances (high latency) and GPU computing where the data transfer between the system memory and the graphics card should be minimized.

The method was validated with the help of inverse shape and position reconstruction problems. However, the method should be applicable to direct optimization problems in cases where the global Nash equilibrium is located at the optimum.

5.3.1 Implementation

The structure of the global Nash algorithm is highly distributed and allows a wide variety of implementations. In this study, both distributed and centralized approaches were used. The former allows a more flexible architecture, but in order to make sure that each player has up-to-date information, some sort of controlling process is preferable, especially when the system grows large. The "master" process holds the information on the current best design vector, as well as the boundary information. The information is passed through the master node. Examples of centralized and distributed architectures are illustrated in Figure 21.

In a theoretical utopian situation where the flow is always fully reconstructed, GNGCA should perform identically to the jDE algorithm except for the faster solution rate provided by DDM. In practice, the flow always remains more or less incomplete, misleading the shape players. Deterministic approaches, such as gradient-based algorithms, may not be able to converge. In the case of evolutionary algorithms, it is important that the shape players do not converge too early since the algorithm could get stuck in some suboptimal region. Therefore, maximizing the efficiency of flow players is important for a successful convergence of the method.

5.3.2 Shape reconstruction example

In order to illustrate the efficiency of the global Nash approach, a two-element airfoil geometry reconstruction problem is considered (Bi-NACA0012, described in Section 4.2). In order to increase the solver solution time difference between

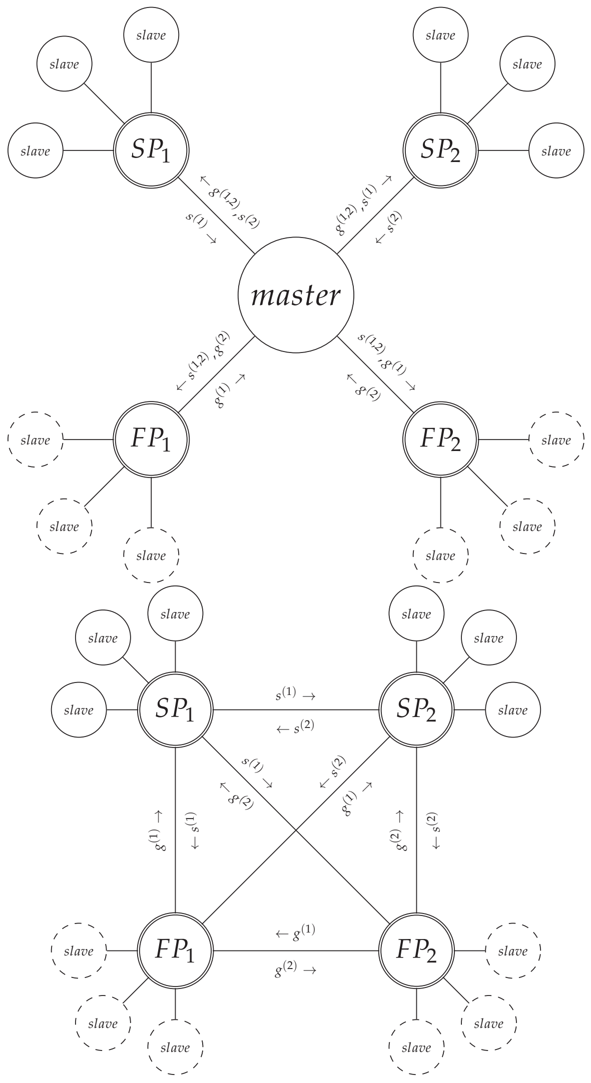


FIGURE 21 Example GNGCA architectures consisting of two flow and two shape players. *Top:* Centralized architecture where the information is passed through a master node which retains up-to-date flow ($\mathbf{g}^{(i)}$) and geometry ($\mathbf{s}^{(i)}$) data. *Bottom:* Distributed architecture where the information is passed between the players. The $0, \ldots, N$ slave processes, which perform the actual flow computations, communicate only with the player nodes, which in turn control the optimization and domain decomposition algorithms.

the global and the decomposed domain, compressible potential flow is used:

$$\begin{array}{rcll} \nabla \cdot \rho \nabla \varphi_{u,\ell} & = & 0 & \text{in} \quad \Omega_{u,\ell} \\ \varphi_{u,\ell} & = & \mathbf{v}_\infty & \text{on} \quad \Gamma_\infty \\ \frac{\partial \varphi_{u,\ell}}{\partial \mathbf{n}} & = & 0 & \text{on} \quad \Gamma_{u,\ell} \\ \varphi_u & = & \varphi_\ell & \text{on} \quad \Gamma_{u,\ell} \\ \varphi_\ell & = & \varphi_u & \text{on} \quad \Gamma_{\ell,u} \end{array} \tag{35}$$

where u and ℓ refer to the upper and lower subdomain, respectively. The pressure ρ is computed using the formula

$$\rho = \left\{ 1 + \frac{\gamma - 1}{2} M_\infty^2 \left(1 - |v|^2 \right) \right\}^\beta \tag{36}$$

The constant $\gamma = 1.4$ is the ratio of specific heats for air and $\beta = \frac{1}{\gamma - 1} = 2.5$. The free-flow Mach number is $M_\infty = .3$, guaranteeing the flow being subsonic in the whole domain. Because of the sharp edges present, the angle of attack is kept at $\alpha = 0.0^\circ$ in order to minimize the effect of singularities at the trailing edges. The shape deformation is limited to 8 design variables. In order to see the effect of parallelization, 2^k, $k = \{1, \ldots 5\}$ slave processes are used. For the shape reconstruction, the jDE algorithm is used with the threshold objective function value of $f_{min} = 10^{-7}$.

Algorithmic convergence rates are listed in Table 12 and convergence of the algorithms in Figure 23. Decomposed mesh is illustrated in Figure 22 and example results in Figure 24. It can be seen that with a small number of processors GNGCA is clearly the fastest algorithm by a large margin, even though only two subdomains were used. The performance of GNGCA is better than described in Paper II. This can be explained by the fact that the overlap is located far away from the elements, unlike in the multi-disk element geometry problem where the overlap touches element surface (Neumann boundary). Of the tested algorithms, the geometry decomposition approach (Nash-jDE) scales best when the number of processors are increased (Figure 23). The effect of incomplete flow reconstruction becomes apparent on GNGCA when a large number of slaves are used.

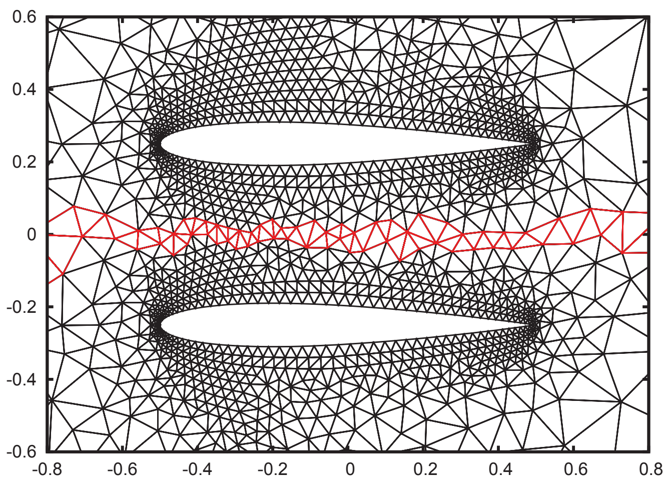


FIGURE 22 Decomposed mesh.

TABLE 12 Algorithmic performance in the Bi-NACA0012 geometry reconstruction case. The parameter n_{sl} refers to the number of slaves, t is the wall-clock time in seconds, and n_{it} is the combined number of objective function evaluations. The first speed-up column refers to the speed-up gained from adding slaves, the second column to the performance difference compared to GNGCA with the same number of slaves.

	n_{sl}	t	n_{it}	speed-up	
DE	2	1.1786E+03s	4974	1.0000	7.8594
	4	6.3631E+02s	4394	1.8522	9.4935
	8	2.3521E+02s	3053	5.0108	4.5220
	16	1.7497E+02s	3819	6.7360	5.0377
	32	1.8397E+02s	4368	6.4065	5.3054
Nash-DE	2	4.3434E+02s	1279	1.0000	2.8964
	4	2.5813E+02s	1521	1.6826	3.8512
	8	1.7307E+02s	2034	2.5096	3.3273
	16	7.1452E+01s	1683	6.0788	2.0572
	32	4.1446E+01s	1924	10.4797	1.1952
GNGCA	2	1.4996E+02s	1150	1.0000	–
	4	6.7026E+01s	1026	2.2373	–
	8	5.2015E+01s	1567	2.8830	–
	16	3.4732E+01s	2039	4.3176	–
	32	3.4676E+01s	3078	4.3246	–

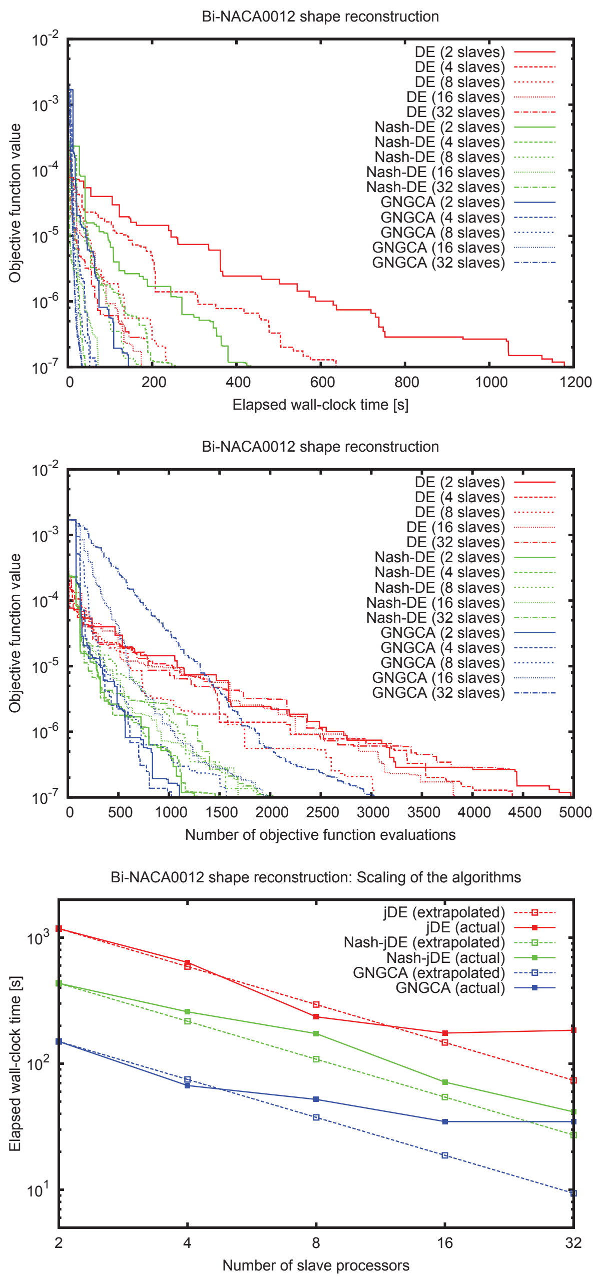


FIGURE 23 Convergence curves for the Bi-NACA0012 geometry reconstruction case. *Top:* Convergence as wall-clock time in seconds. *Middle:* Convergence as the number of objective function evaluations. *Bottom:* Scaling of the algorithms as the function of the number of slaves. The two-slave result for each of the algorithms was used as the baseline for extrapolation.

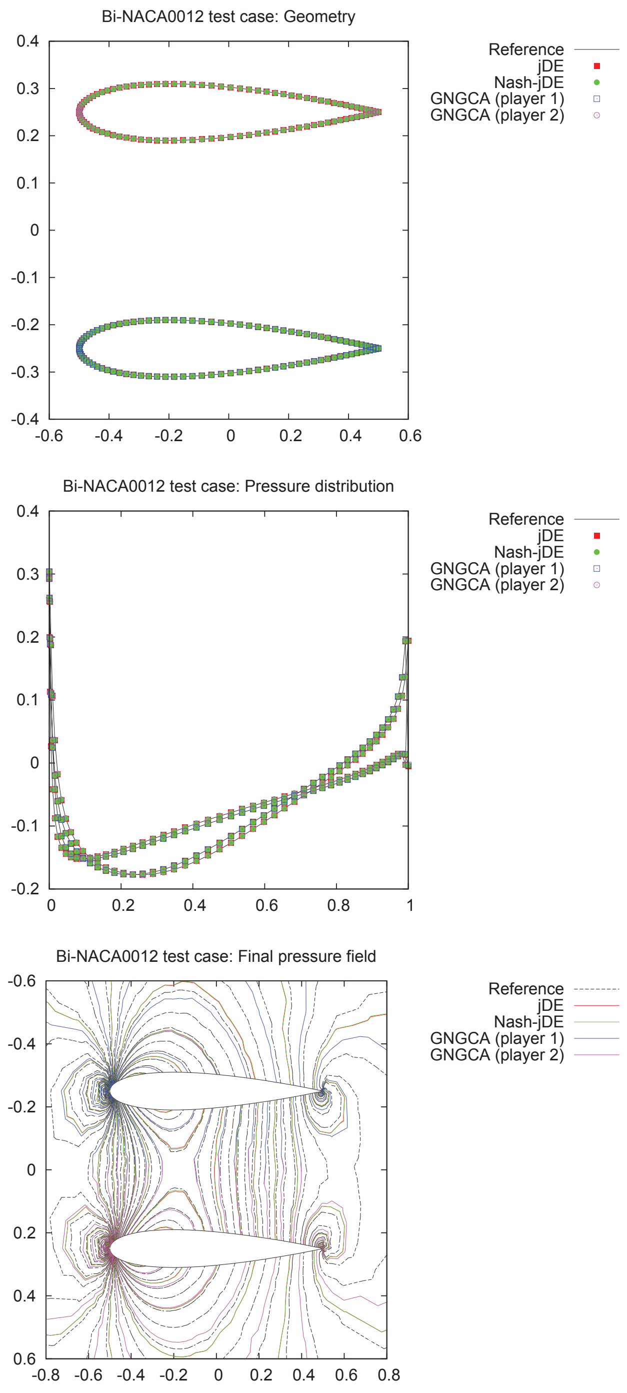


FIGURE 24 Example final ressults.

6 INCREASE OF PERFORMANCE WITH GPUS FOR SOLVING OPTIMIZATION PROBLEMS

In order to satisfy the growing demands of the nascent 3D gaming industry, hardware-based 3D graphics acceleration was introduced in the mid-1990s. Later generations of graphics hardware have become more sophisticated and compared to CPUs, far more efficient in certain tasks. The dramatic increase in the efficiency of GPUs is illustrated in Figure 25.

Drawing a single pixel on the computer screen does not need much information about the surrounding pixels. Therefore the process can be easily done in parallel. Following Flynn's taxonomy, GPUs are massively parallel SIMD machines, not unlike the vector processors which were popular in supercomputers from the 1970s to the 1990s [7, 120]. The vast increase in GPU processing power has been made possible by increasing the number of computing cores, which has no physical limitations similar to those arising from transistor miniaturization.

In order to be able to accelerate complex graphical features in computer games, GPU processors were made programmable, opening their processing capabilities to applications they were not originally designed for. Early GPU applications were done using OpenGL and required in-depth understanding of the system. Modern development platforms such as CUDA by NVIDIA [91] and platform-independent OpenCL [1] have considerably simplified GPU software development making it a cost-effective alternative for high-performance computing.

The advantages of general purpose GPU computing were realized by the high-performance computing community, and early on it was adopted for scientific computing. Applications of GPU computing include MRI reconstruction [114], data mining [36], molecular modeling [113], and quantum physics simulation [2] to mention a few. A review of early applications can be found in [37]. Flow simulation is computationally intensive and can be easily implemented on a parallel environment, making it a good application for GPUs [50]. Flow computations based on the finite difference method can be done node-wise on GPU cores [28]. Computations based on the finite element and the finite volume methods have also been implemented [41, 102]. The Lattice-Boltzmann method is especially

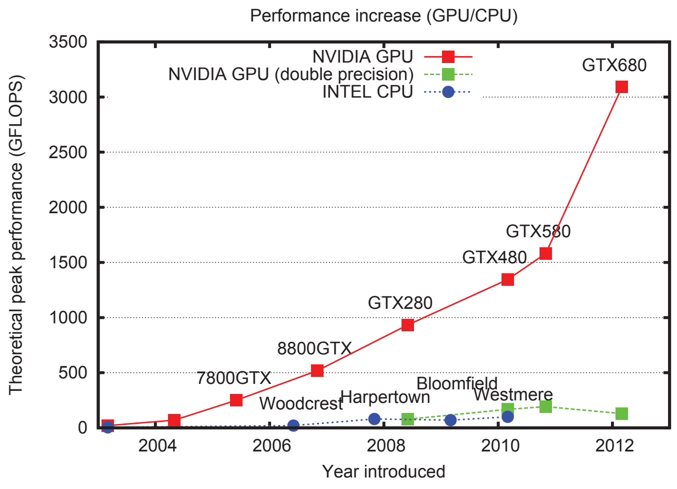


FIGURE 25 Increase of the efficiency of NVIDIA GPUs compared to Intel CPUs [93].

suitable for GPUs [10].

Scientific GPU computing has found its way to supercomputers, and some of the fastest machines in the world now include clusters of GPUs [118]. The results are often impressive: speed-ups in the range of 100$\times$ or more have been reported in the literature [89]. However, critics have pointed out that the comparisons have not always been fair to the traditional CPUs [74].

Despite the advantages, GPUs have serious limitations that must be considered. First of all, since GPUs are massive parallel systems, the problem that GPUs are applied to must be such that it can be split into a large number of simple computations. Standard issues arising from parallel computing, such as race conditions, must also be avoided. The memory structure poses its own limitations, which have to be taken into account when the software is being developed. In order to achieve an optimal speed-up, the computing cores should be kept as occupied as possible during computations. Compared to the traditional CPU programming, implementation of GPU code has much higher impact on efficiency.

6.1 GPU architecture

In order to understand GPU computing, some knowledge of the underlying hardware and memory structure is required. In this work, the CUDA development platform was used and therefore the terms used here are those introduced by the NVIDIA Corporation [91].

The CPU is called *host* whereas *device* refers to the graphics card. Functions that are run on GPU are called *kernels*. Each computing core on the device can run a single instance of kernel, called *thread*. The threads which are run on physically connected cores form a *block*. The threads in a block are run simultaneously. The collection of blocks is called *kernel grid*, the size of which is determined by the program during the launch of a kernel. Figure 26 depicts the thread hierarchy in NVIDIA graphics cards.

The main memory on the device is called *global memory*. Any thread can access it and it is the location where the data to be processed is sent from the host. Random access to the global memory is time-consuming because fetching the data may require several clock cycles since memory accessing is done synchronously. If the data is organized along the threads, the number of cycles can be minimized. This property, called *coalescence*, had a major impact on the efficiency of older GPUs. Fortunately, this has been corrected in modern GPUs.

Each of the blocks has a common *shared memory*, which only the threads within a block can access. It is much faster than the global memory, and in order to maximize the efficiency of the code it is recommended that it should be used as much as possible. Unfortunately, the size of the shared memory is very limited, up to 48 kilobytes in modern GPUs [92].

Other memory types include read-only *constant memory* where global constants are stored. Thread-specific scalar variables are primarily stored into the fast but limited *register memory*. If the register memory becomes full, or local arrays are used, the excess data is stored into a *local memory* which despite its name is actually a part of the global memory that has been reserved to a single thread. Excessive use of local memory has a negative impact on the efficiency of the code. Finally, *texture memory* is a form of cached global memory and is less used in general purpose GPU computing. The GPU memory structure in NVIDIA cards is illustrated in Figure 27.

Memory bandwidth between the host and the device is shown in Figure 28. The most striking aspect is how limited the bandwidth between the host and device memory is. Minimizing data transfer between the two types of memory is one of the most important considerations in producing efficient GPU code.

Another limitation of GPUs conserns the double-precision floating point arithmetic. Single-precision floating point arithmetic is much faster and accurate enough for most graphics processing. However, it is often not enough when scientific computing is concerned. Older graphics cards did not have the capacity to process double-precision floating point numbers which necessitated the use of much slower software-based emulation [44]. Modern GPUs based on brands such as NVIDIA Tesla and Fermi have cores capable of doing double-precision arithmetic although less efficiently [32]. For example, the NVIDIA GeForce GTX 480 graphics card operates at $\frac{1}{8}$th of the single-precision capability when double precision is used [4]. Best speed-ups on GPUs can be achieved using mixed precision, where the fast single-precision computations are corrected using double precision [42].

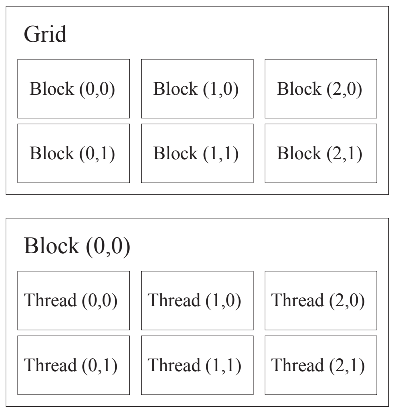


FIGURE 26 Thread/block structure.

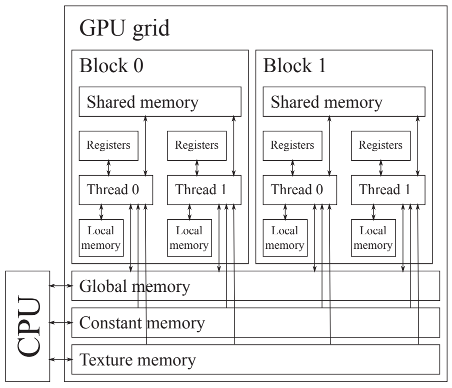


FIGURE 27 Device memory structure.

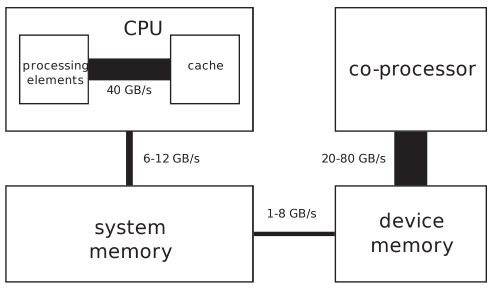


FIGURE 28 Device/host memory bandwidths [43].

6.2 Efficiency of GPUs

From the point of view of solving partial differential equations, the two main aspects that affect the speed-up of GPU computing are the size and the structure of the mesh. Sparse matrices, especially those produced using finite elements on unstructured meshes, are problematic for GPU computing because with them efficient utilization of threads and memory becomes complicated. Therefore many GPU applications operate on grids or dense matrices.

In order to illustrate the speed-up of GPU computing on unstructured meshes of varying size, a simple potential flow problem is solved using the CUSP preconditioned conjugate gradient solver [8]. Consider the three-ellipse element geometry domain Ω, where the Laplace equation is used as the state equation,

$$\begin{aligned} -\Delta\varphi &= 0 \quad \text{in} \quad \Omega \\ \varphi &= x \quad \text{on} \quad \Gamma_\infty \\ \frac{\partial\varphi}{\partial\mathbf{n}} &= 0 \quad \text{on} \quad \Gamma_{1,2,3} \end{aligned} \tag{37}$$

A mesh with four different levels of refinement is tested using a single GeForce GTX 580 graphics card. The original mesh ($\mathcal{T}_h$) is divided 2, 3, and 4 times ($\mathcal{T}_{\frac{h}{2}}$, $\mathcal{T}_{\frac{h}{4}}$, and $\mathcal{T}_{\frac{h}{8}}$). The meshes are illustrated in Figure 29. The results for computations are listed in Table 13 and the speed-ups compared to a CPU-optimized conjugate gradient method are shown in Figure 30. For a fair comparison, the times, including overhead caused by data transfer between the computers and between the CPU and GPU, are also included. It can be seen that with the use of a small mesh the speed-up is nonexistent, but it becomes dramatic (nearly 25) when the mesh becomes large, illustrating the behavior typical in GPU computing. When the overhead is taken into account, the benefit of the GPU approach is much less evident. This is in part due to the fact that the test problem is very simple. For example, if the matrix has to be solved repeatedly (as in the case of nonlinear partial differential equations), the difference in efficiency between GPUs and CPUs could become more apparent. Also, a proper GPU cluster environment can reduce the transfer time. Furthermore, the overhead can be hidden by performing the data transfer when the GPUs are computing the result (cf., for example, [57]).

In this work, GPU computing is limited to solving the linearized system of equations ($A\mathbf{u} = \mathbf{f}$). The matrix A and vector $\mathbf{f}$ are computed on the master computer and transferred to separate computers accommodating the GPUs. After finishing the computation, vector $\mathbf{u}$ containing the computed solution is sent back to the master. A more sophisticated approach would be to minimize the data transfer and overhead by doing as much as possible on-the-fly on the GPUs. However, efficient implementation of this is beyond the scope of this study.

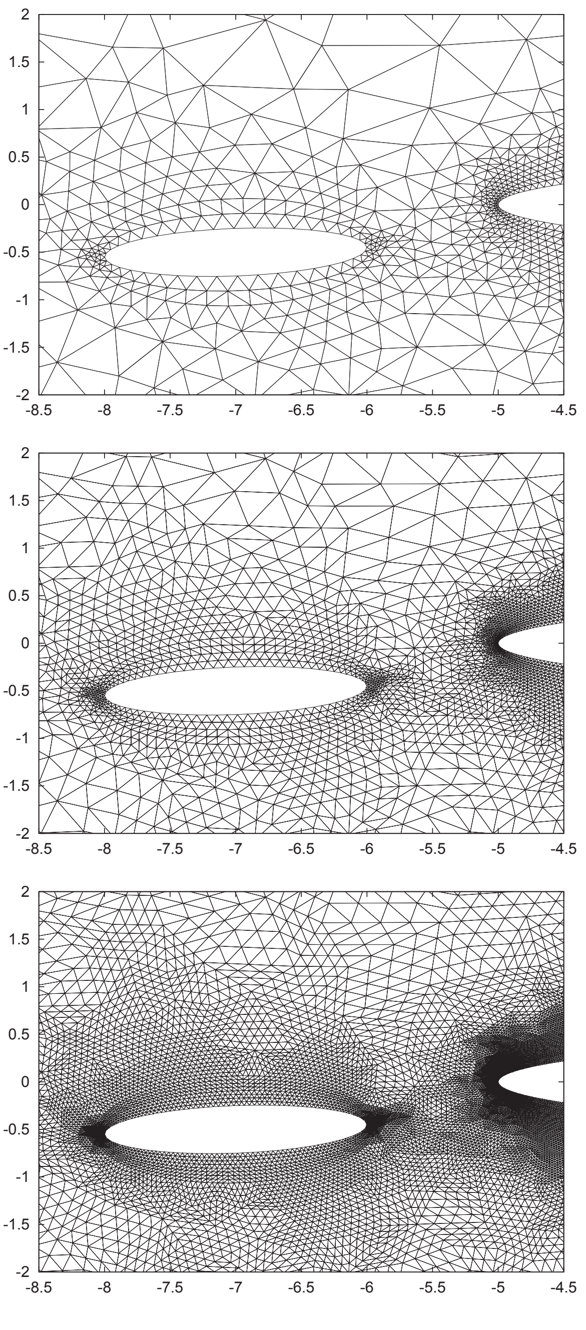
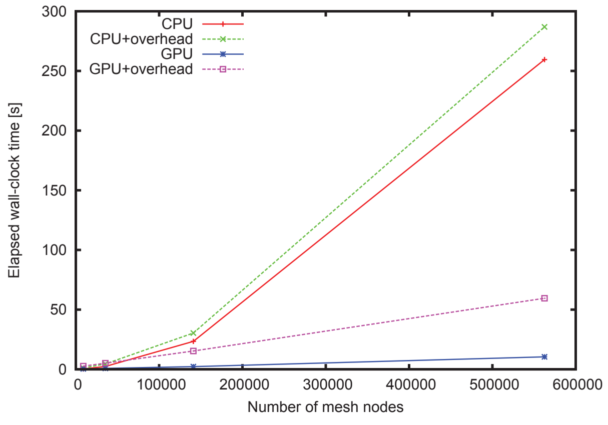
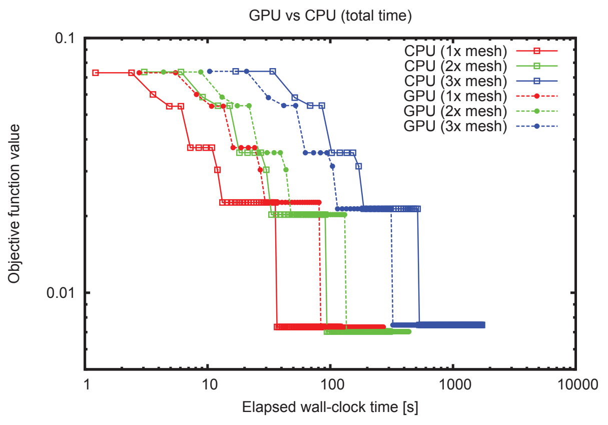


FIGURE 29 Mesh refinement in the case of three-ellipse element geometry. *Top:* Original mesh $\mathcal{T}_h$. *Middle:* Mesh where the elements are divided once ($\mathcal{T}_{\frac{h}{2}}$). *Bottom:* Mesh where the elements are divided twice ($\mathcal{T}_{\frac{h}{4}}$).





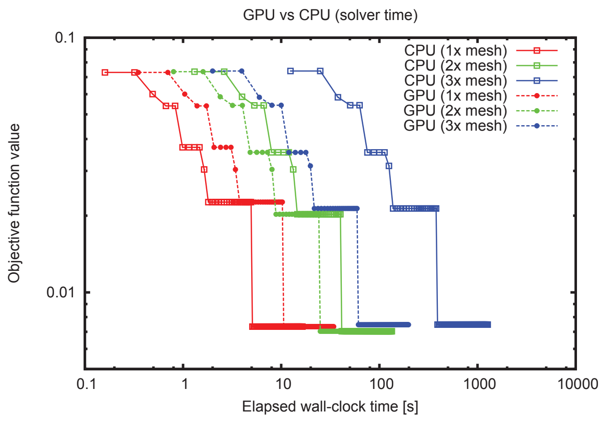


FIGURE 30 *Top:* Efficiency of GPU compared to CPU. For a more fair comparison, the overhead is included. *Middle:* Initial convergence of the jDE algorithm using meshes of different refinement levels. For comparing the optimal efficiency of GPUs, the solver-only time is included (*bottom*).

TABLE 13 Comparison between CPU and GPU when the conjugate gradient method is used.

	nodes	CPU [s]	CPU [s] +overhead	GPU [s]	GPU [s] +overhead	solver performance [×CPU]	actual performance [×CPU]
$\mathcal{T}_h$	8967	0.278374	1.14831	0.330020	2.66052	0.84	0.43
$\mathcal{T}_{\frac{h}{2}}$	35464	2.36992	4.45027	0.782007	5.08469	3.03	0.88
$\mathcal{T}_{\frac{h}{4}}$	141042	23.3545	30.4248	2.23112	15.3214	10.47	1.99
$\mathcal{T}_{\frac{h}{8}}$	562534	259.542	286.906	10.4392	59.5071	24.86	4.82

6.3 Implementation of GPUs for global Nash games

In this section, the impact of GPU computing on the efficiency of GNGCA is demonstrated. Because of the limited number of graphics cards available, the study was limited to a multi-disk element geometry problem using two disks in incompressible potential flow,

$$\begin{array}{rcll} -\Delta\varphi_{1,2} & = & 0 & \text{in } \Omega_{1,2} \\ \varphi_{1,2} & = & \mathbf{v}_\infty & \text{on } \Gamma_\infty \\ \frac{\partial\varphi_{1,2}}{\partial n} & = & 0 & \text{on } \Gamma_{0,1,2} \\ \varphi_1 & = & \varphi_2 & \text{on } \Gamma_{1,2} \\ \varphi_2 & = & \varphi_1 & \text{on } \Gamma_{2,1} \end{array} \tag{38}$$

where k is the index of subdomain, j the right-hand side and ℓ the left-hand side neighbor. The free-flow velocity $\mathbf{v}_\infty = (\cos\alpha, \sin\alpha)$ and the angle of attack $\alpha = 0°$.

In order to minimize the effect of overhead and the difference between implementations, both the CPU and GPU use the same CUSP solver giving somewhat unrealistic results for the CPU. However, this is justified since this test only considers the effect of GPUs on algorithmic convergence. The different configurations tested are listed in Table 14. All computations were done on two remote computers consisting of AMD Athlon II X4 645 quad-core processors clocked at 3.1 GHz. The first computer has a GeForce GTX 580 graphic card and the second one a slightly less efficient GeForce GTX 480 card. Since only the initial convergence was of interest, the algorithms were terminated when they reached the threshold objective function value of $f_{min} = 10^{-4}$. The elements of the meshes were divided once in order to increase the computational cost and to make the performance difference between GPU and CPU more apparent.

The results for the different configurations are listed in Table 15. The convergence curves are shown in Figure 31. Based on the results, it is evident that the convergence of the algorithm is defined by the efficiency of the flow players, confirming the results obtained in Paper II. This is dramatically illustrated in

TABLE 14 Tested CPU/GPU configurations. The number of processors used by each of the algorithms.

case	n_{FP}	n_{SP}
1	2	2
2	2	4
3	2	6
4	2	2 (GPU)
5	2 (GPU)	2
6	2 (GPU)	4
7	2 (GPU)	6

TABLE 15 Comparison of hybrid CPU/GPU implementations for the global Nash approach where t is the wall-clock time and n_{it} is the combined number of objective function evaluations by both players.

case	t	n_{it}
1	5.0088E+03s	642
2	4.1798E+03s	1063
3	3.7532E+03s	1419
4	4.7862E+03s	1462
5	4.4846E+03s	541
6	1.5670E+03s	391
7	1.2577E+03s	461

case (4) where the shape players are operated on GPUs. Even though the actual GPU performance is $\approx 2.5\times$ that of the CPU, the algorithm was not better than the CPU-only case. This can be explained by the fact that the shape players cannot converge if the flow players do not feed them with fast enough updated flow information.

On the other hand, when the flow players use GPUs, the performance is rapidly increased. Although no apparent improvement in the wall-clock time was made in the case of two slaves, the wall-clock time was more than halved when the number of slaves was doubled. Because the speed-up provided by GPUs was limited, further increase of slave processes did not bring more speed-ups.

TABLE 16 The average solution time and the average time needed to perform a single objective function evaluation.

	CPU	GPU	speed-up
solver	13.7782s	1.3542s	10.1744×
total	17.7068s	7.2038s	2.4580×

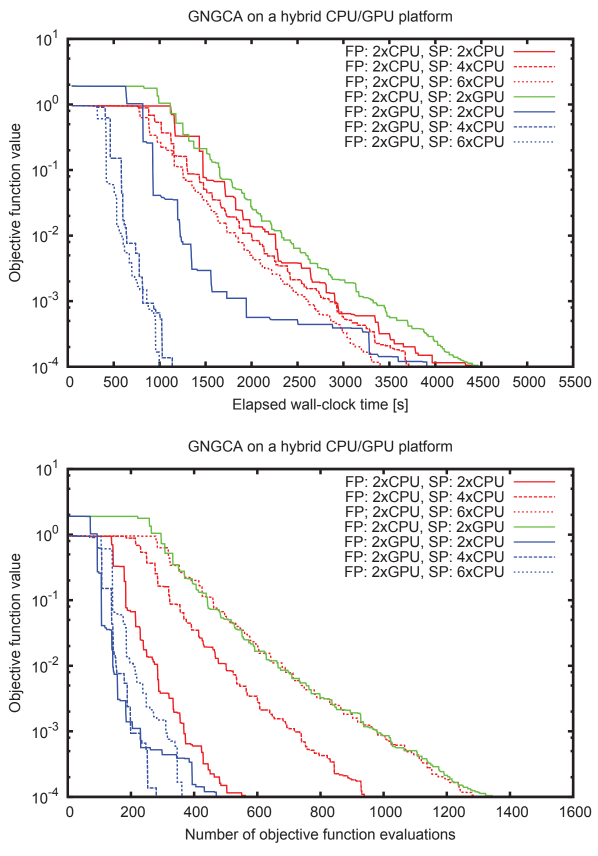


FIGURE 31 Convergence of GNGCA in different hybrid environments. Convergence is compared using wall-clock time (*left*) and number of required iterations (*right*).

7 CONCLUSIONS AND FUTURE WORK

In this research, multi-objective optimization algorithms coupled with Nash strategies have been studied, leading to the innovative concept of distributed optimization.

A competitive game approach well-suited to parallelization was implemented to provide significant algorithmic speed-ups. A novel "distributed one-shot" method that couples Nash games with domain and geometry decomposition was introduced. The approach was tested on various test cases with different number of processors. It was also tested on a hybrid CPU/GPU environment.

The numerical results on various optimization test cases complete previous studies in showing that by decomposing a design geometry into a distributed system with virtual game strategies the required amount of function evaluations can be efficiently reduced. The introduced method, the global Nash game coalition algorithm (GNGCA), was found to be superior compared to other methods in many cases due to reduction of geometrical complexity and local computational domain size. The method is inherently parallel, which makes its implementation on large computing clusters straightforward.

The results obtained also indicate the importance of geometry and domain decomposition. Poor choice of geometry splitting can result in degraded performance or failure of convergence. Also, the performance of GNGCA depends on the efficiency of flow players. In the cases where shape players operated faster than flow players could repair the flow, the algorithm did not bring massive increases in efficiency. However, the results obtained suggest that in cases where there are a large number of subdomains, the speed-ups could become impressive. Alternatively, if the flow reconstruction can be done much faster on graphic processing units, the overall algorithm efficiency can be greatly increased.

The utilization of GPUs on PDE-based analyzers had two main issues to be considered in regard to the speed-up of evaluation. When sufficiently large mesh sizes are used, the improvements in efficiency are impressive. On the other hand, the effort of implementing efficient GPU code for efficiency improvements remains very time consuming and problem-specific.

The performance of the new "distributed optimization" approach was validated only on simple flow problems with a small number of subdomains. Further studies using large-scale problems and advanced domain and geometry decomposition methods are needed.

The next step is to extend these new efficient distributed shape design optimization methods to industrial design collaborative platforms on which a large spectrum of optimization software abounds. This advanced technology transfer goal requires distributed optimization of simple 3D configurations with more complex nonlinear PDEs. These more challenging problems can be computed using intelligent domain and geometry decomposition techniques coupled with game strategies. One step further in complexity is to take into account more realistic physics modeling via robust design with uncertainties.

Resulting from the outcomes of the TEKES Design project, the scientific computing laboratory of the University of Jyväskylä is now well-equipped with its collaborative platform and the Finnish Design Test Case Database. A set of generic geometries for complex physics optimization problems can now be defined in order to evaluate performances of distributed optimization algorithms in situations much closer to the industry. The results can be published and compared in International Open Database Workshop events.

The results obtained combining distributed evolutionary optimization and GPU techniques illustrate that the new method is potentially highly parallelizable. This research is the first step on an innovative roadmap to design optimization with complex mathematical modeling and geometries. It is expected from the results of this study that efficiency obtained on simple optimization model test cases will be significantly increased on large size optimization problems in industrial design environments.

YHTEENVETO (FINNISH SUMMARY)

Muodon optimointi on tärkeä alue teollisessa suunnittelussa, koska varsin usein kehitettävän tuotteen tehokkuus ja toiminta riippuu sen muodosta. Perinteisesti optimointi on toteutettu rakentamalla fyysisiä malleja, mikä on hidasta ja kallista. Tietokoneiden tehostuminen on mahdollistanut uusien laskennallisten menetelmien kehittämisen, jolloin tuotteen muoto voidaan optimoida tehokkaammin. Valitettavasti realististen mallien optimoiminen on hyvin haastavaa suuren laskentamäärän vuoksi, joten kehittyneempien menetelmien kehittämiselle on tarvetta.

Väitöskirjassa "Hajautetut evoluutioalgoritmeja ja peliteoriaa hyödyntävät monitavoiteoptimointimenetelmät muodon optimoinnille" tutkitaan menetelmiä, joilla muodon optimointia pystytään tehostamaan. Ensimmäinen tutkituista menetelmistä perustuu peliteoriaan. Muuttamalla yksitavoitteinen ongelma monitavoitteiseksi ja ratkaisemalla se non-kooperatiivisena Nash-pelinä, voidaan varsinkin evoluutioon perustuvien optimointialgoritmien tehokkuutta parantaa. Tästä on hyötyä muodon optimoinnissa, sillä usein esiintyy tilanteita, joissa monimutkainen geometria voidaan jakaa osiin. Menetelmää on tutkittu ensimmäisessä väitöskirjaan sisällytetyssä artikkelissa.

Aluehajoitusmenetelmiä (DDM) on käytetty menestyksekkäästi raskaassa laskennassa. Laskettava alue rajataan osiin, jolloin laskenta kyetään jakamaan usean eri tietokoneen tai prosessorin kesken. Toinen väitöskirjassa esitettävä menetelmä perustuu osin tähän. Yhdistämällä geometria- ja aluehajoitusmenetelmät ja ratkaisemalla ne yhtenä globaalina Nash-pelinä, voidaan laskentaa tehostaa huomattavasti (GNGCA-algoritmi). Menetelmä on julkaistu toisessa väitöskirjaan sisällytetyssä artikkelissa, ja sitä vertaillaan muihin rinnakkaistettuihin evoluutioalgoritmeihin neljännessä artikkelissa.

Tietokonegrafiikan laskennassa käytettävien grafiikkaprosessorien (GPU) kehitys on ollut hyvin nopeaa viime vuosina. Vaikka ne onkin suunniteltu lähinnä pelikäyttöön, on niiden mahdollistama huomattava laskentapotentiaali otettu käyttöön myös tieteellisessä laskennassa. Väitöskirjan kolmannessa artikkelissa tutkitaan, kuinka Nash-algoritmien toimintaa voidaan tehostaa grafiikkaprosessoreiden avulla. Johdanto-osiossa esitellään lisäksi tuloksia GNGCA-algoritmin tehostamisesta GPU-laskennan avulla.

Viidennessä ja viimeisessä artikkelissa tutkitaan hierarkkisten evoluutioalgoritmien käyttöä verkollisten ja verkottomien menetelmien avulla.

Väitöskirjassa käytetyt testitapaukset ovat yksinkertaisia muodon ja geometrian rekonstruktiotehtäviä lähinnä potentiaalivirtauksen tapauksessa. Tulevaisuudessa väitöskirjassa esiteltyjä menetelmiä voidaan soveltaa haastavampiin ongelmiin, joissa tutkitaan monimutkaisia geometrioita sekä realistisempia epälineaarisia virtausmalleja. Lisäksi GPU-laskenta voidaan integroida entistä tiiviimmin optimointiprosessiin. Päämääränä on kehittää menetelmä, josta on käytännön hyötyä teollisessa optimoinnissa.

REFERENCES

[1] M. Aaftab. OpenCL specification version 1.0. URL: <http://www.khronos.org/registry/cl/>, 2011.

[2] A. G. Anderson, W. A. Goddard III, and P. Schröderb. Quantum Monte Carlo on graphical processing units. *Computer Physics Communications*, 177(3):298–306, August 2007.

[3] J. Andersson. A survey of multiobjective optimization in engineering design. Technical report, Department of Mechanical Engineering, Linköping University, Linköping, Sweden, 2000.

[4] H. Anzt, T. Hahn, V. Heuveline, and B. Rocker. GPU accelerated scientific computing: Evaluation of the NVIDIA Fermi architecture; elementary kernels and linear solvers. Technical Report 2010-4, Karlsruhe Institute of Technology, 2010.

[5] E. Arian and S. Ta'asan. Shape optimization in one shot. *Progress in Systems and Control Theory*, 19, 1995.

[6] P. K. Banerjee and R. Butterfield. *Boundary element methods in engineering science*. McGraw–Hill Book Co., UK, 1981.

[7] G. Bell and J. Gray. What's next in high-performance computing? *Communications of the ACM*, 45(2):91–95, February 2002.

[8] N. Bell and M. Garland. Cusp: Generic parallel algorithms for sparse matrix and graph computations. URL: <http://cusp-library.googlecode.com>, 2010. Version 0.1.0.

[9] T. Belytschko, Y. Krongauz, D. Organ, M. Fleming, and P. Krysl. Meshless methods: An overview and recent developments. *Computer Methods in Applied Mechanics and Engineering*, 139(1–4):3–47, December 1996.

[10] M. Bernaschi, M. Fatica, S. Melchionna, S. Succi, and E. Kaxiras. A flexible high-performance Lattice Boltzmann GPU code for the simulations of fluid flows in complex geometries. *Concurrency and Computation: Practice and Experience*, 22:1–14, 2010.

[11] J. Branke, K. Deb, K. Miettinen, and R. Slowiński, editors. *Multiobjective Optimization: Interactive and Evolutionary Approaches*. Springer–Verlag, Berlin Heidelberg, 2008.

[12] J. Brest, S. Greiner, B. Bošković, M. Mernik, and V. Žumer. Self-adapting control parameters in differential evolution: A comparative study on numerical benchmark problems. *IEEE Transactions on Evolutionary Computation*, 10(6):646–657, December 2006.

[13] E. Cantú-Paz and D. E. Goldberg. Efficient parallel genetic algorithms: theory and practice. *Computer Methods in Applied Mechanics and Engineering*, 186(2–4):221–238, June 2000.

[14] H. Q. Chen, R. Glowinski, and J. Périaux. A domain decomposition/Nash equilibrium methodology for the solution of direct and inverse problems in fluid dynamics with evolutionary algorithms. In U. Langer, M. Discacciati, D.E. Keyes, O.B. Widlund, and W. Zulehner, editors, *Domain Decomposition Methods in Science and Engineering XVIII*, volume 60 of *Lecture Notes in Computational Science and Engineering*, Heidelberg, 2006. Springer. Proceedings of the 17th International Conference on Domain Decomposition Methods held at St. Wolfgang / Strobl, Austria, July 3–7, 2006.

[15] T. J. Chung. *Computational Fluid Dynamics*. Cambridge University Press, Cambridge, 2002.

[16] C. A. Coello Coello. Evolutionary multi-objective optimization: A historical view of the field. *IEEE Computational Intelligence Magazine*, 1(1):28–36, February 2006.

[17] C. A. Coello Coello, D. A. Van Veldhuizen, and G. B. Lamont. *Evolutionary Algorithms for Solving Multi-Objective Problems*. Kluwer Academic Publishers, New York, 2002.

[18] T. A. Davis. Algorithm 832: UMFPACK v4.3-an unsymmetric-pattern multifrontal method. *ACM Transactions on Mathematical Software*, 30(2):196–199, 2004.

[19] R. Dawkins. *The Selfish Gene*. Oxford University Press, Oxford, 1976.

[20] K. A. De Jong. *Analysis of the Behavior of a Class of Genetic Adaptive Systems*. PhD thesis, University of Michigan, Ann Arbor, Michigan, 1975.

[21] K. Deb. *Multi-objective optimization using evolutionary algorithms*. John Wiley & Sons, Chichester, UK, 2001.

[22] K. Deb, A. Pratap, S. Agarwal, and T. Meyarivan. A fast and elitist multiobjective genetic algorithm: NSGA-II. *Evolutionary Computation, IEEE Transactions on*, 6(2):182–197, April 2002.

[23] A. Dervieux, B. Van Leer, J. Périaux, and A. Rizzi, editors. *Numerical Simulation of Compressible Euler Flows*. Braunschweig/Wiesbaden, Germany, 1989.

[24] J.-A. Désidéri and A. Janka. Multilevel shape parameterization for aerodynamic optimization – application to drag and noise reduction of transonic/supersonic business jet. In *European Congress on Computational Methods in Applied Sciences and Engineering, ECCOMAS*, Jyväskylä, Finland, 24-28 July, 2004, 2004.

[25] J.-A. Désidéri, J. Périaux, and Z. L. Tang. Multi-objective design strategies using deterministic optimization with different parametrizations in aerodynamics. In P. Neittaanmäki, T. Rossi, S. Korotov, E. Oñate, J. Périaux, and D. Knörzer, editors, *European Congress on Computational Methods in Applied Sciences and Engineering ECCOMAS 2004*, 2004.

[26] M. Dorigo, A. Colorni, and V. Maniezzo. The ant system: Optimization by a colony of cooperating agents. *IEEE Transactions on Systems, Man, and Cybernetics Part B: Cybernetics*, 26(1):29–41, 1996.

[27] L. Dumas. CFD-based optimization for automotive aerodynamics. In D. Thévenin and G. Janiga, editors, *Optimization and Computational Fluid Dynamics*, pages 191–216. Springer–Verlag, Berlin Heidelberg, 2008.

[28] E. Elsen, P. LeGresley, and E. Darve. Large calculation of the flow over a hypersonic vehicle using a GPU. *Journal of Computational Physics*, 227(24):10148–10161, 2008.

[29] B. Epstein, A. Jameson, S. Peigin, D. Roman, N. Harrison, and J. Vassberg. Comparative study of three-dimensional wing drag minimization by different optimization techniques. *Journal of Aircraft*, 46(2):526–541, March–April 2009.

[30] L.J. Eshelman and J.D. Schaffer. Real-coded genetic algorithms and interval-schemata. In L.D. Whitley, editor, *Foundations of Genetic Algorithms 2*, pages 187–202. Morgan Kaufmann, San Mateo, 1993.

[31] C. Farhat and F.-X. Roux. A method of finite element tearing and interconnecting and its parallel solution algorithm. *International Journal for Numerical Methods in Engineering*, 32(6):1205–1227, 1991.

[32] M. Fatica. Accelerating Linpack with CUDA on heterogeneous clusters. In *GPGPU-2 Proceedings of 2nd Workshop on General Purpose Processing on Graphics Processing Units*, 2009.

[33] L. J. Fogel, A. J. Owens, and M. J. Walsh. *Artificial Intelligence through Simulated Evolution*. John Wiley, New York, 1966.

[34] C. M. Fonseca and P. J. Fleming. Genetic algorithms for multiobjective optimization: Formulation, discussion and generalization. In S. Forrest, editor, *In Genetic Algorithms: Proceedings of the Fifth International Conference*, pages 416–423, San Mateo, CA, 1993. Morgan Kauffman.

[35] C. M. Fonseca and P. J. Fleming. An overview of evolutionary algorithms in multiobjective optimization. *Evolutionary Computation*, 3:1–16, 1995.

[36] A. Gainaru, E. Slusanschi, and S. Trausan-Matu. Mapping data mining algorithms on a GPU architecture: A study. In M. Kryszkiewicz, H. Rybinski, A. Skowron, and Z. Ras, editors, *Foundations of Intelligent Systems*, volume

6804 of *Lecture Notes in Computer Science*, pages 102–112. Springer–Verlag, Berlin Heidelberg, 2011. 10.1007/978-3-642-21916-0_12.

[37] M. Garland, S. Le Grand, J. Nickolls, J. Anderson, J. Hardwick, S. Morton, E. Phillips, Y. Zhang, and V. Volkov. Parallel computing experiences with CUDA. *IEEE Micro*, 28(4):13–27, 2008.

[38] C. A. Georgopoulou and K. C. Giannakoglou. Multiobjective metamodel–assisted memetic algorithms. In C.-K. Goh, Y.-S. Ong, and K. C. Tan, editors, *Multi-Objective Memetic Algorithms*, pages 153–181. Springer–Verlag, Berlin Heidelberg, 2009.

[39] K. C. Giannakoglou. Design of optimal aerodynamic shapes using stochastic optimization methods and computational intelligence. *Progress in Aerospace Sciences*, 38(1):43–76, 2001.

[40] F. W. Glover and M. Laguna. *Tabu Search*. Springer, June 1998.

[41] D. Göddeke, S.H.M. Buijssen, H. Wobker, and S. Turek. GPU acceleration of an unmodified parallel finite element Navier–Stokes solver. In W. Smari and J. P. McIntire, editors, *High Performance Computing & Simulation 2009*, pages 12–21, June 2009.

[42] D. Göddeke and R. Strzodka. Cyclic reduction tridiagonal solvers on GPUs applied to mixed precision multigrid. *IEEE Transactions on Parallel and Distributed Systems*, 22:22–32, January 2011.

[43] D. Göddeke, R. Strzodka, J. Mohd-Yusof, P. McCormick, S.H.M. Buijssen, M. Grajewski, and S. Turek. Exploring weak scalability for FEM calculations on a GPU-enhanced cluster. *Parallel Computing*, 33(10–11):685–699, September 2007.

[44] D. Göddeke, R. Strzodka, and S. Turek. Performance and accuracy of hardware-oriented native-, emulated- and mixed-precision solvers in FEM simulations. *International Journal of Parallel, Emergent and Distributed Systems*, 22(4):221–256, January 2007.

[45] C.-K. Goh, Y.-S. Ong, and K. C. Tan, editors. *Multi-Objective Memetic Algorithms*. Springer–Verlag, Berlin Heidelberg, 2009.

[46] D. E. Goldberg. *Genetic Algorithms in Search, Optimization and Machine Learning*. Addison–Wesley, Reading, Massachusetts, January 1989.

[47] A.O. Griewank. Generalized descent for global optimization. *Journal of Optimization Theory and Applications*, 34(1):11–39, 1981.

[48] R. T. Haftka, Z. Gürdal, and M. P. Kamat. *Elements of structural optimization*. Kluwer Academic Publishers, 1990.

[49] J. P. Hämäläinen, K. Miettinen, P. Tarvainen, and J. Toivanen. Interactive solution approach to a multiobjective optimization problem in a paper machine headbox design. *Journal of Optimization Theory and Applications*, 112(2):265–281, 2003.

[50] M. Harris. Fast fluid dynamics simulation on the GPU. In *GPU Gems*, pages 637–665. NVIDIA, 2004.

[51] W. E. Hart, N. Krasnogor, and J. E. Smith. Memetic evolutionary algorithms. In W. E. Hart, N. Krasnogor, and J. E. Smith, editors, *Recent Advances in Memetic Algorithms*, pages 3–27. Springer Verlag, 2004.

[52] J. Haslinger and R. A. E. Mäkinen. *Introduction to shape optimization: theory, approximation, and computation*. SIAM, Philadelphia, Pennsylvania, 2003.

[53] S. B. Hazra, V. Schulz, J. Brezillon, and N. R. Gauger. Aerodynamic shape optimization using simultaneous pseudo-timestepping. *Journal of Computational Physics*, 204(1):46–64, March 2005.

[54] M. R. Hestenes and E. Stiefel. Methods of conjugate gradients for solving linear systems. *Journal of Research of the National Bureau of Standards*, 49(6), December 1952.

[55] C. Hirsch. *Numerical Computation of Internal and External Flows*, volume 1–2. J. Wiley & Sons, New York, 1988.

[56] J. H. Holland. *Adaptation in Natural and Artificial Systems*. The University of Michigan Press, Ann Arbor, 1975.

[57] D. A. Jacobsen, J. C. Thibault, and I. Senocak. An MPI–CUDA implementation for massively parallel incompressible flow computations on multi-GPU clusters. In *48th AIAA Aerospace Sciences Meeting and Exhibit*, January 2010.

[58] A. Jameson. Iterative solution of transonic flows over airfoils and wings, including flows at Mach 1. *Communications on Pure and Applied Mathematics*, 27:283–309, 1974.

[59] A. Jameson and M. Fatica. Using computational fluid dynamics for aerodynamics. Technical report, Stanford University, Stanford, California, 2006.

[60] M. T. Jensen. Guiding single-objective optimization using multi-objective methods. In S. Cagnoni, C. Johnson, J. Cardalda, E. Marchiori, D. Corne, J.-A. Meyer, J. Gottlieb, M. Middendorf, A. Guillot, G. Raidl, and E. Hart, editors, *Applications of Evolutionary Computing*, volume 2611 of *Lecture Notes in Computer Science*, pages 91–98. Springer, Berlin Heidelberg, 2003.

[61] J. M. Johnson and Y. Rahmat-Samii. Genetic algorithms in engineering electromagnetics. *IEEE Antennas and Propagation Magazine*, 39(4):7–21, August 1997.

[62] I. C. Kampolis and K. C. Giannakoglou. A multilevel approach to single- and multiobjective aerodynamic optimization. *Computer Methods in Applied Mechanics and Engineering*, 197(33–40):2963–2975, June 2008.

[63] J. Kennedy and R. Eberhart. Particle swarm optimization. In *Proceedings of the IEEE International Conference on Neural Networks 1995*, volume 4, pages 1942–1948, Perth, WA, Australia, August 1995. IEEE Micro Service, Piscataway, NJ.

[64] S. Kirkpatrick, C. D. Gelatt, and M. P. Vecchi. Optimization by simulated annealing. *Science*, 220(4598):671–680, 1983.

[65] J. Knowles, R. Watson, and D. Corne. Reducing local optima in single-objective problems by multi-objectivization. In E. Zitzler, L. Thiele, K. Deb, C. Coello Coello, and D. Corne, editors, *Evolutionary Multi-Criterion Optimization*, pages 269–283. Springer, Berlin Heidelberg, 2001.

[66] J. D. Knowles and D. W. Corne. Approximating the nondominated front using the Pareto archived evolution strategy. *Evolutionary Computation*, 8(2):149–172, 2000.

[67] N. Kroll, N. R. Gauger, J. Brezillon, R. Dwight, A. Fazzolari, D. Vollmer, K. Becker, H. Barnewitz, V. Schulz, and S. Hazra. Flow simulation and shape optimization for aircraft design. *Journal of Computational and Applied Mathematics*, 203:397–411, 2007.

[68] I. Kroo. Aeronautical applications of evolutionary design. VKI Lecture Series on Optimization Methods & Tools for Multicriteria/Multidisciplinary Design, November 2004.

[69] G. Kuruvila, S. Ta'asan, and M.D. Salas. Airfoil design and optimization by the one-shot method. Technical Report 95–0478, AIAA, January 1995.

[70] D.S. Lee, L. F. Gonzalez, J. Périaux, and G. Bugeda. Design optimization using advanced artificial intelligent system coupled to hybrid-game strategies. In M. Negnevitsky, editor, *Proceedings of the 3rd International Workshop on Artificial Intelligence in Science and Technology (AISAT 2009), Hobart, Tasmania, Australia, November 23–24, 2009*. University of Tasmania, 2009.

[71] D.S. Lee, L.F. Gonzalez, J. Périaux, and K. Srinivas. Efficient hybrid-game strategies coupled to evolutionary algorithms for robust multidisciplinary design optimization in aerospace engineering. *IEEE Transactions on Evolutionary Computation*, 15(2):133–150, 2011.

[72] D.S. Lee, L.F. Gonzalez, K. Srinivas, and J. Périaux. Robust evolutionary algorithms for UAV/UCAV aerodynamic and RCS design optimisation. *Computers & Fluids*, 37(5):547–564, 2008.

[73] D.S. Lee, J. Périaux, K. Srinivas, L.F. Gonzalez, N. Qin, and E. Oñate. Shock control bump optimization on natural laminar aerofoil. In *Computational Fluid Dynamics 2010*, pages 253–260, Berlin Heidelberg, 2011. Springer–Verlag.

[74] V. W. Lee, C. Kim, J. Chhugani, M. Deisher, D. Kim, A. D. Nguyen, N. Satish, M. Smelyanskiy, S. Chennupaty, P. Hammarlund, R. Singhal, and P. Dubey. Debunking the 100x GPU vs. CPU myth: an evaluation of throughput computing on CPU and GPU. *SIGARCH Comput. Archit. News*, 38(3):451–460, June 2010.

[75] J. Leskinen and J. Périaux. A new distributed optimization approach for solving CFD design problems using Nash games coalition and evolutionary algorithms. In Randolph Bank, Michael Holst, Olof Wildund, and Jinchao Xu, editors, *Domain Decomposition Methods in Science and Engineering XX*, Lecture Notes in Computational Science and Engineering, to appear. Proceedings of the 20th International Conference on Domain Decomposition Methods held at La Jolla / San Diego, California, February 7–11, 2011.

[76] J. Leskinen, H. Wang, and J. Périaux. Increasing parallelism of evolutionary algorithms by Nash games in design inverse flow problems. *Engineering Computations*, to appear.

[77] P. Loridan and J. Morgan. A theoretical approximation scheme for Stackelberg problems. *Journal of Optimization Theory and Applications*, 61(1):95–110, 1989.

[78] R. A. E. Mäkinen, P. Neittaanmäki, J. Périaux, M. Sefrioui, and J. Toivanen. Parallel genetic solution for multiobjective MDO. In P. Schiano, A. Ecer, J. Périaux, and N. Satofuka, editors, *Parallel Computational Fluid Dynamics: Algorithms and Results Using Advanced Computers*, pages 352–359. Elsevier Science, 1997.

[79] J. Mandel. Balancing domain decomposition. *Communications in Numerical Methods in Engineering*, 9(3):233, 1993.

[80] E. Mezura-Montes, J. Velázquez-Reyes, and C. A. Coello Coello. A comparative study of differential evolution variants for global optimization. In *GECCO '06: Proceedings of the 8th annual conference on Genetic and evolutionary computation*, pages 485–492, New York, NY, USA, 2006. ACM Press.

[81] K. Miettinen. *Nonlinear Multiobjective Optimization*. Kluwer Academic Publishers, Boston, 1999.

[82] B. Mohammadi. Fluid dynamics computation with NSC2KE. An user-guide release 1.0. Technical Report 0164, INRIA, May 1994.

[83] P. Moscato. On evolution, search, optimization, genetic algorithms and martial arts: Towards memetic algorithms. Technical Report Caltech

Concurrent Computation Program 826, California Institute of Technology, Pasadena, California, 1989.

[84] H. Mühlenbein. Evolution in time and space – the parallel genetic algorithm. In G. J. E. Rawlins, editor, *Foundations of Genetic Algorithms*, pages 316–337, San Francisco, CA, 1991. Morgan Kaufmann.

[85] H. Mühlenbein, M. Schomisch, and J. Born. The parallel genetic algorithm as function optimizer. In *Proceedings of the 4th International Conference on Genetic Algorithms*, page 271, 1991.

[86] J. Nash. Non-cooperative games. *The Annals of Mathematics*, 2(54):286–295, 1951.

[87] F. Neri. *Fitness diversity adaptation in memetic algorithms*. PhD thesis, University of Jyväskylä, Finland, 2007.

[88] F. Neri and V. Tirronen. Recent advances in differential evolution: a survey and experimental analysis. *Artificial Intelligence Review*, 33(1–2):61–106, 2010.

[89] J. Nickolls and W.J. Dally. The GPU computing era. *IEEE Micro*, 30(2):56–69, March–April 2010.

[90] M. Nowostawski and R. Poli. Parallel genetic algorithm taxonomy. In L. Jain, editor, *International Conference on Knowledge-Based Intelligent Electronic Systems*, pages 88–92. IEEE Press, 1999.

[91] NVIDIA Corporation. CUDA. URL: <http://www.nvidia.com/cuda>.

[92] NVIDIA Corporation. Tuning CUDA applications for Fermi, 2010.

[93] NVIDIA Corporation. NVIDIA CUDA C programming guide version 4.1, 2011.

[94] V. Pareto. *Cours D'Economie Politique*, volume I–II. F. Rouge, Lausanne, 1896.

[95] J. Périaux, H. Q. Chen, B. Mantel, M. Sefrioui, and H. T. Sui. Combining game theory and genetic algorithms with application to DDM-nozzle optimization problems. *Finite Elements in Analysis and Design*, 37(5):417–429, May 2001.

[96] O. Pironneau. *Finite Element Methods for Fluids*. J. Wiley & Sons, Chichester, 1989.

[97] D. Quagliarella and A. Vicini. Viscous angle and multicomponent airfoil design with genetic algorithms. *Finite Elements in Analysis and Design*, 37:365–380, 2001.

[98] A. Quarteroni and G. Rozza. Optimal control and shape optimization of aorto-coronaric bypass anastomoses. *Mathematical Models and Methods in Applied Sciences*, 13(12):1801–1824, 2003.

[99] A. Rapoport and A. M. Chammah. *Prisoner's Dilemma*. University of Michigan Press, 1965.

[100] I. Rechenberg. *Evolutionsstrategie: Optimierung technischer Systeme nach Prinzipien der biologischen Evolution*. Frommann–Holzboog Verlag, Stuttgart, 1973.

[101] J. Robinson and Y. Rahmat-Samii. Particle swarm optimization in electromagnetics. *IEEE Transactions on Antennas and Propagation*, 52(2):397–407, February 2004.

[102] S. Rostrup and H. De Sterck. Parallel hyperbolic PDE simulation on clusters: Cell versus GPU. *Computer Physics Communications*, 181:2164–2179, 2010.

[103] J. D. Schaffer. *Some experiments in machine learning using vector evaluated genetic algorithms*. PhD thesis, Vanderbilt University, Nashville, TN (USA), 1985.

[104] H. A. Schwarz. Über einen grenzübergang durch alternierendes verfahren. *Vierteljahrsschrift der Naturforschenden Gesellschaft in Zürich*, 15:272–286, 1870.

[105] H.-P. Schwefel. *Numerische Optimierung von Computer-Modellen mittles der Evolutionsstrategie*, volume 26 of *Interdiscplinary Systems Research*. Birkhäuser, Basel, 1977.

[106] M. Sefrioui and J. Périaux. A hierarchical genetic algorithm using multiple models for optimization. In M. Schoenauer, K. Deb, G. Rudolph, X. Yao, E. Lutton, J. J. Merelo, and H.-P. Schwefel, editors, *Parallel Problem Solving from Nature PPSN VI*, number 1917 in Lecture Notes in Computer Science, pages 879–888. Springer, 2000.

[107] M. Sefrioui and J. Périaux. Nash genetic algorithms: Examples and applications. In *Proceedings of the 2000 Congress on Evolutionary Computation: CEC00 (June–September 2000)*, pages 509–516, La Jolla Marriott Hotel, La Jolla, California, USA, 2000. IEEE.

[108] K. Sindhya. *Hybrid evolutionary multi-objective optimization with enhanced convergence and diversity*. PhD thesis, University of Jyväskylä, Finland, 2011.

[109] B. Smith, P. Bjørstad, and W. Gropp. *Domain Decomposition: Parallel Multilevel Methods for Elliptic Partial Differential Equations*. Cambridge University Press, June 1996.

[110] N. Srinivas and K. Deb. Multiobjective optimization using nondominated sorting in genetic algorithms. *Evolutionary Computation*, 2(3):221–248, Fall 1994.

[111] H. Von Stackelberg. *The Theory of the Market Economy*. Oxford University Press, Oxford, England, 1952.

[112] R. Steuer. *Multiple criteria optimization: Theory, computation and application*. John Wiley & Sons, Inc., New York, 1986.

[113] J. E. Stone, D. J. Hardy, I. S. Ufimtsev, and K. Schulten. GPU-accelerated molecular modeling coming of age. *Journal of Molecular Graphics and Modelling*, 29(2):116–125, September 2010.

[114] S.S. Stone, J.P. Haldar, S.C. Tsao, W. m.W. Hwu, B.P. Sutton, and Z.-P. Liang. Accelerating advanced MRI reconstructions on GPUs. *Journal of Parallel and Distributed Computing*, 68(10):1307–1318, 2008.

[115] R. Storn and K. Price. Differential Evolution – a simple and efficient adaptive scheme for global optimization over continuous spaces. Technical Report TR-95-012, International Computer Science Institute, Berkeley, CA, 1995.

[116] R. Storn and K. Price. Differential evolution – a simple and efficient heuristic for global optimization over continuous spaces. *Journal of Global Optimization*, 11(4):341–359, December 1997.

[117] J. I. Toivanen, R. A. E. Mäkinen, J. Rahola, S. Järvenpää, and P. Ylä-Oijala. Gradient-based shape optimisation of ultra-wideband antennas parameterised using splines. *IET Microwaves, Antennas and Propagation*, 4:1406–1414, 2010.

[118] TOP500.org. TOP500 supercomputer sites. URL: <http://www.top500.org>.

[119] A. Vicini and D. Quagliarella. Airfoil and wing design through hybrid optimization strategies. *AIAA Journal*, 37(5):634–645, 1999.

[120] V. Volkov and J. W. Demmel. Benchmarking GPUs to tune dense linear algebra. In *SC '08: Proceedings of the 2008 ACM/IEEE Conference on Supercomputing*, pages 1–11, Piscataway, NJ, USA, November 2008. IEEE Press.

[121] W. A. Wall, M. A. Frenzel, and C. Cyron. Isogeometric structural shape optimization. *Computer Methods in Applied Mechanics and Engineering*, 197(33–40):2976–2988, June 2008.

[122] G. G. Wang and S. Shan. Review of metamodeling techniques in support of engineering design optimization. *Journal of Mechanical Design*, 129(4):370–380, 2007.

[123] H. Wang, H.Q. Chen, and J. Périaux. A study of gridless method with dynamic clouds of points for solving unsteady CFD problems in aerodynamics. *International Journal for Numerical Methods in Fluids*, 64(1):98–118, 2010.

[124] M. Y. Wang, X. M. Wang, and D. M. Guo. A level set method for structural topology optimization. *Computer Methods in Applied Mechanics and Engineering*, 192(1–2):227–246, January 2003.

[125] D.H. Wolpert and W.G. Macready. No free lunch theorems for optimization. *Evolutionary Computation, IEEE Transactions on*, 1(1):67–82, April 1997.

[126] E. Zitler and L. Thiele. Multiobjective evolutionary algorithms: A comparative case study and the strength Pareto approach. *IEEE Transactions on Evolutionary Computation*, 3(4):257–271, 1999.