

The Numerical Objects Report Series

Report 1997:16

Numerical Solution of PDEs on Parallel Computers Utilizing Sequential Simulators

Are Magnus Bruaset Xing Cai
Hans Petter Langtangen
Aslak Tveito

September 28, 1997

**NUMERICAL
NO
OBJECTS**

This document is classified as Open. All information herein is the property of Numerical Objects AS and should be treated in accordance with the stated classification level. For documents that are not publicly available, explicit permission for redistribution must be collected in writing.

Numerical Objects Report 1997:16
Title <i>Numerical Solution of PDEs on Parallel Computers Utilizing Sequential Simulators</i>
Written by <i>Are Magnus Bruaset Xing Cai Hans Petter Langtangen Aslak Tveito</i>
Approved by <i>Are Magnus Bruaset September 28, 1997</i>

Numerical Objects, Diffpack and Siscat and other names of Numerical Objects products referenced herein are trademarks or registered trademarks of Numerical Objects AS, P.O. Box 124, Blindern, N-0314 Oslo, Norway.

Other product and company names mentioned herein may be the trademarks of their respective owners.

Contact information

Phone: +47 22 06 73 00
Fax: +47 22 06 73 50
Email: info@nobjects.com
Web: <http://www.nobjects.com>

**Copyright © Numerical Object AS, Oslo, Norway
September 28, 1997**

Contents

- 1 Introduction** **1**
- 2 Domain decomposition and parallelization** **2**
- 3 A simulator-parallel model in Diffpack** **3**
- 4 A generic programming framework** **5**
- 5 Some numerical experiments** **7**
 - 5.1 The Poisson equation 8
 - 5.2 The linear elasticity problem 9
 - 5.3 The two-phase porous media flow problem 10

This report should be referenced as shown in the following `BIBTEX` entry:

```
@techreport{NO1997:16,  
  author = "Are Magnus Bruaset and Xing Cai and Hans Petter Langtangen and Aslak Tveit",  
  title = "Numerical Solution of PDEs on Parallel Computers Utilizing Sequential Simulation",  
  institution = "Numerical Objects AS, Oslo, Norway",  
  type = "The Numerical Objects Report Series",  
  number = "\#{}1997:16",  
  year = "September 28, 1997",  
}
```

Numerical Solution of PDEs on Parallel Computers Utilizing Sequential Simulators

Are Magnus Bruaset Xing Cai
Hans Petter Langtangen Aslak Tveito

Abstract

We propose a strategy, based on domain decomposition methods, for parallelizing existing sequential simulators for solving partial differential equations. Using an object-oriented programming framework, high-level parallelizations can be done in an efficient and systematic way. Concrete case studies, including numerical experiments, are provided to further illustrate this parallelization strategy.

Chapter 1

Introduction

The purpose of this paper is to address the following problem: How can existing sequential simulators be utilized in developing parallel simulators for the numerical solution of partial differential equations (PDEs)? We discuss this issue in the light of object-oriented (OO) programming techniques, which can substantially increase the efficiency of implementing parallel PDE software. Our parallelization approach is to use domain decomposition (DD) as an overall numerical strategy. More precisely, DD is applied at the level of subdomain simulators, instead of at the level of linear algebra. This gives rise to a simulator-parallel programming model that allows easy migration of sequential simulators to multiprocessor platforms. We propose a generic programming framework in which the simulator-parallel model can be realized in a flexible and efficient way. The computational efficiency of the resulting parallel simulator depends strongly on the efficiency of the original sequential simulators, and can be enhanced by the numerical efficiency of the underlying DD structure. This parallelization strategy enables a high-level parallelization of existing OO codes. We show that the strategy is flexible and efficient and we illustrate the parallelization process by developing some concrete parallel simulators in Diffpack [DP].

Chapter 2

Domain decomposition and parallelization

Roughly speaking, DD algorithms (see e.g. [SBG96]) search the solution of the original large problem by iteratively solving many smaller problems over subdomains. DD methods are e.g. very efficient numerical techniques for solving large linear systems, even on sequential computers. Such methods are particularly attractive for parallel computing if the subproblems can be solved concurrently.

In this paper we concentrate on a particular DD method, called the (overlapping) *additive Schwarz method* (see [SBG96]). In this method the subdomains form an overlapping covering of the original domain. The method can be formulated as an iterative process, where in each iteration we solve updated boundary value problems over the subdomains. The work on each subdomain in each iteration consists mainly of solving the PDE(s) restricted to a subdomain using values from its neighboring subdomains, computed in the previous iteration, as Dirichlet boundary conditions. The subproblems can be solved in parallel because neighboring subproblems are only coupled through previously computed values in the overlapping region between the non-physical inner boundaries. The convergence of the above iterative process depends on the amount of overlapping. It can be shown for many elliptic PDE problems that the additive Schwarz method has an optimal convergence behavior, for a fixed level of overlapping, when an additional coarse grid correction is applied in each iteration (see e.g. [CM94] for the details).

Chapter 3

A simulator-parallel model in Diffpack

We hereby propose a *simulator-parallel* model for parallelizing existing sequential PDE simulators. The programming model uses DD at the level of subdomain simulators and assigns processors of a parallel computer each with a sequential simulator, which is readily extended from existing sequential simulator(s). The parallel computation related global administration and communication can be easily realized in an OO framework. We have tested this parallelization approach within Diffpack [DP], which is an OO environment for scientific computing, with particular emphasis on numerical solution of PDEs.

The C++ Diffpack libraries contain user-friendly objects for I/O, GUIs, arrays, linear systems and solvers, grids, scalar and vector fields, visualization and grid generation interfaces etc. The flexible and modular design of Diffpack allows easy incorporation of new numerical methods into its framework, whose content grows constantly. In Diffpack, a finite element (FE) based PDE solver is typically realized as a C++ class having a grid, FE fields for the unknowns, the integrands in the weak formulation of the PDE, a linear system toolbox and some standard functions for prescribing essential boundary conditions, together with some interface functions for data input and output. The above parallelization approach offers flexibility in the sense that different types of grid, linear system solvers, preconditioners, convergence monitors etc. are allowed for different subproblems. A new parallel simulator can thus be derived from reliable sequential simulators in a flexible and efficient way.

Diffpack was originally designed without paying particular attention to parallel computing. A large number of flexible, efficient, extensible, reliable, and optimized sequential PDE solver classes has been developed during re-

cent years. The parallelization strategy presented in this paper thus extends Diffpack by offering means of adapting these sequential solvers for concurrent computers, hopefully without significant loss of computational efficiency compared to a special-purpose application code which is implemented particularly for parallel computing. Normally, such special-purpose parallel codes employ *distributed* arrays, grids and so on. That is, the conceptual model contains abstractions representing *global* quantities. In our simulator-parallel model proposed above, we only work with local PDE problems. This avoids the need for data distribution in the traditional sense. We are only concerned with a standard, sequential PDE solver on each processor and some glue for communicating boundary values for the local problems, in addition to an overall numerical iteration. OO programming is a key ingredient that makes this migration to parallel computers fast and reliable. The advantages of such an approach are obvious and significant: (1) optimized and reliable existing sequential solvers can be re-used for each subproblem, (2) message passing statements are kept to a minimum and can be hidden from the application programmer in generic objects, (3) the extra glue for communication and iteration is just a short code at the same abstraction level as the theoretical description of the parallel DD algorithm. We believe that the suggested parallel extension of Diffpack will make it much easier for application developers to utilize parallel computing environments. Nevertheless, the fundamental question is how efficient our general high-level approach is. This will be addressed in the numerical experiments.

Chapter 4

A generic programming framework

To increase flexibility and portability, the simulator-parallel model is realized in a generic programming framework consisting of three main parts (see Figure 4.1); 1. `SubdomainSimulator`; 2. `Communicator`; 3. `Administrator`. Furthermore, each of the three parts of the programming framework is implemented as a C++ class hierarchy, where different subclasses specialize in different types of PDEs and specific numerical methods. In this way a programmer can quickly adapt his existing sequential simulator(s) into the framework. The subsequent discussion requires some knowledge of OO programming techniques.

The base class `SubdomainSimulator` is a generic interface class offering an abstract representation of a sequential simulator. First, the class contains *virtual* functions that are to be overridden in a derived class to make the connection between `SubdomainSimulator` and an existing sequential simulator. Second, the class contains functions for accessing the subdomain local data needed by the communication between neighboring subdomains. We have introduced subclasses of `SubdomainSimulator` to generalize different specific simulators, e.g. `SubdomainFEMSolver` for simulators solving a scalar/vector elliptic PDE discretized by FE methods and `SubdomainFDMSolver` for simulators using finite difference discretizations.

With the data access functions in `SubdomainSimulator` at disposal, the base class `Communicator` is designed to generalize the communication between neighboring subdomains. The primary task of the class is to determine the communication pattern by finding on which processors lay the neighboring subdomain simulators and which part of the local data should be sent to each neighbor etc. A hierarchy of different communicators are also built to handle different situations. One example is `CommunicatorFEMSP` which spe-

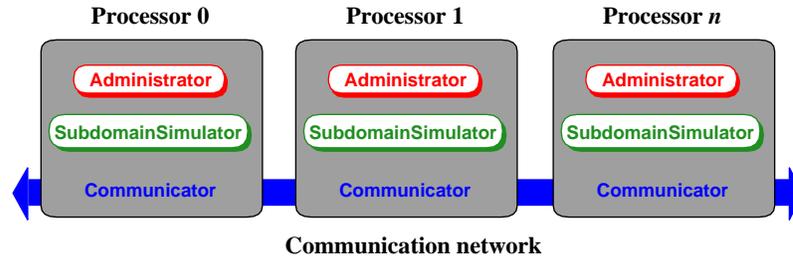


Figure 4.1: A generic framework for the simulator-parallel programming model.

cializes in communication between subdomain simulators using the FE discretization. More importantly, the concrete message passing model, which is MPI in the current implementation, is hidden from the user such that a new message passing model may be easily inserted without changing the other parts of the framework.

Finally, `Administrator` performs the numerics underlying the iterative process of DD. Coordinating with each other, an object of `Administrator` exists on each subdomain. Typically, `Administrator` has under its control a `SubdomainSimulator` and a `Communicator`, so that member functions of `SubdomainSimulator` are invoked for computation and member functions of `Communicator` are invoked for communication. A hierarchy of administrator classes are built to realize different solution processes for different model problems, among those is class `PdeFemAdmSP` for cases of a scalar/vector elliptic PDE discretized by FE methods.

Chapter 5

Some numerical experiments

In this section, we apply the proposed parallelization strategy to develop parallel simulators for solving the Poisson equation, the linear elasticity problem and the two-phase porous media flow problem, respectively. The implementation of the parallel simulators consists essentially of extending the existing Diffpack sequential simulators to fit in the programming framework mentioned above. Given the fact that we apply the FE discretization for elliptic PDEs, the situation for the first two test problems is straightforward, because only a single elliptic PDE is involved in each case, even though it is a scalar equation in one case and a vector equation in the other. The classes `CommunicatorFEMSP` and `PdeFemAdmSP` can be used directly. So the only necessary implementation is to derive a new simulator class as a subclass of both the existing sequential simulator and class `SubdomainFEMSolver`. Using a subclass for gluing the sequential solver and the parallel computing environment avoids *any modifications* of the original sequential solver class. The parallelization is done within an hour. The situation for the third test problem is slightly more demanding, because of the involved system of PDEs consisting of an elliptic PDE and a hyperbolic PDE. The two PDEs are solved with different numerical methods which means additional work for implementing a suitable administrator, as a subclass of `PdeFemAdmSP`. However, programming in the proposed framework enables a quick implementation which is done in a couple of days.

In the following, we give for each test problem the mathematical model and CPU consumption measurements associated with different numbers of processors in use. The measurements are obtained on a SRI Cray Origin 2000 parallel computer with R10000 processors. The resulting parallel simulators demonstrate not only nice scalability, but also in some simulations the intrinsic numerical efficiency of the underlying DD algorithm.

P	M	CPU	speedup	subdomain grid
1	4	44.41	—	257×257
4	4	11.37	3.91	257×257
6	6	7.92	5.61	257×181
8	8	5.84	7.60	257×135
12	12	3.92	11.33	257×91
16	16	3.06	14.51	257×69

Table 5.1: CPU consumptions (in seconds) of the iterative DD solution process for different M . The fixed total number of unknowns is 481×481 .

5.1 The Poisson equation

Our first test problem is the 2D Poisson equation on the unit square, with a right hand side and Dirichlet boundary conditions such that $u(x, y) = -xe^y$ is the solution of the problem.

We denote the number of subdomains by M and introduce an underlying uniform 481×481 global grid covering Ω . (The grid is only used as the basis for partitioning Ω for different choices of M .) With the number of processors P equal to M , we study the CPU consumption of the DD solution process with different choices of P . Suitable coarse grid corrections, together with a fixed level of overlapping, are used to achieve the convergence up to a prescribed accuracy under the same number of DD iterations, independently of P . The largest coarse grid problem is considerably smaller than any subproblem, so the CPU time spent on the coarse grid correction is negligible. To obtain a reference CPU consumption associated with a single processor, we make a sequential implementation of the same iterative DD process and measure the CPU consumption with $M = 4$. In this test problem we are able to use a fast direct solver for the subproblems. The important property is that its computational work is linearly proportional to the number of unknowns. From the numerical experiments, we observe that the communication overhead is very small in comparison with the solution of the subproblems. These two factors together attribute to the nice speedup in Table 5.1.

Even though DD methods achieve convergence independently of the problem size, it might be more efficient to use a standard Krylov subspace method for the current test problem when only a small number of processors are available. So the purpose of the current test problem is mainly to verify that our implementation of the generic programming framework has kept the communication overhead at a negligible level. In the following case of solving a

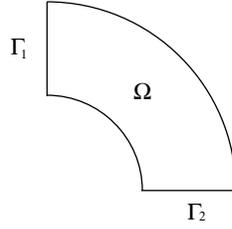


Figure 5.1: The solution domain for the linear elasticity problem.

linear elasticity problem, we will see that the parallel DD approach shows its numerical efficiency even for a discrete problem of a medium size.

5.2 The linear elasticity problem

The 2D deformation of an elastic continuum subject to a plain strain can be modelled by the following partial differential equation

$$-\mu\Delta\mathbf{U} - (\mu + \lambda)\nabla\nabla \cdot \mathbf{U} = \mathbf{f},$$

where the displacement field $\mathbf{U} = (u_1, u_2)$ is the primary unknown and \mathbf{f} is a given vector function. Here μ and λ are constants. The 2D domain is a the quarter of a hollow disk, see Figure 5.1. On the boundary the stress vector is prescribed, except on Γ_1 , where $u_1 = 0$ and on Γ_2 , where $u_2 = 0$.

We use a structured 241×241 curvilinear global grid as the basis for partitioning Ω into M overlapping subdomain grids. Again we have $P = M$. In this test problem we apply an inexact solver for the subproblems. More precisely, a conjugate gradient method preconditioned with incomplete LU-factorization (ILU) is used to solve the subproblems. Convergence of the subproblems is considered reached when the residual of the local equation system, in the discrete L_2 -norm, is reduced by a factor of 10^2 . For the global convergence of the DD method we require that the global residual, in the discrete L_2 -norm, is reduced by a factor of 10^4 . In Table 5.2, we list the CPU consumptions and the number of iterations I used in connection with different M . The fixed number of degrees of freedom is 116,162. To demonstrate the superior efficiency of the parallel DD approach, we also list the CPU consumption and number of iterations used by a sequential conjugate gradient method, preconditioned with ILU, for the same global convergence requirement and problem size. Note that the super linearity of the speedup is due to the fact that the preconditioned conjugate method works more efficiently for smaller subproblems.

Table 5.2: CPU consumptions and number of iterations used by the parallel DD approach. The measurements for $M = 1$ are associated with a sequential conjugate gradient method preconditioned with ILU.

M	CPU	speedup	I	subdomain grid
1	110.34	–	204	241×241
2	40.97	2.69	7	129×241
4	23.20	4.76	11	129×129
6	12.43	8.88	8	93×129
8	7.95	13.88	7	69×129
12	4.95	22.29	7	47×129
16	3.64	30.31	7	35×129

5.3 The two-phase porous media flow problem

We consider a simple model of two-phase (oil and water) porous media flow in oil reservoir simulation,

$$s_t + \vec{v} \cdot \nabla(f(s)) = 0 \quad \text{in } \Omega \times (0, T], \quad (5.1)$$

$$-\nabla \cdot (\lambda(s) \nabla p) = q \quad \text{in } \Omega \times (0, T]. \quad (5.2)$$

In the above system of PDEs, s and p are the primary unknowns and $\vec{v} = -\lambda(s) \nabla p$. We carry out simulations for the time interval $0 < t \leq 0.4$ seconds. At each discrete time level, the two PDEs are solved in sequence. The hyperbolic Equation (5.1), referred to as the saturation equation, is solved by an explicit finite difference scheme, whereas the elliptic Equation (5.2), referred to as the pressure equation, is solved by a FE method. The 2D spatial domain is the unit square with impermeable boundaries. An injection well (modelled by a delta function) is located at the origin and a production well at (1,1). Initially, $s = 0$ except at the injection well, where $s = 1$.

A uniform 241×241 global grid is used to cover Ω . For all the discrete time levels, the convergence requirement for the DD method when solving (5.2) is that the absolute value of the discrete L_2 -norm of the global residual becomes smaller than 10^{-8} . We have used a conjugate gradient method preconditioned with ILU as the inexact subdomain solver, where local convergence is considered reached when the local residual in the discrete L_2 -norm is reduced by a factor of 10^3 .

Table 5.3: Simulations of two-phase flow by a parallel DD approach; Columns of total CPU consumptions are listed for 1. The whole simulation, 2. The pressure equation, 3. The saturation equation, accumulated from 744 discrete time levels.

M	total CPU	subdomain grid	CPU pre. eq.	CPU sat. eq.
2	12409.01	129×241	11807.60	241.41
4	5343.50	129×129	5202.79	140.71
6	3201.89	129×91	3101.38	100.09
8	2827.77	129×69	2770.05	77.72
12	1881.33	91×69	1826.82	54.51
16	1388.75	69×69	1346.36	42.39

Acknowledgments

This work has received support from The Research Council of Norway (Programme for Supercomputing) through a grant of computing time. The authors also thank Dr. Klas Samuelsson for numerous discussions on various subjects.

Bibliography

- [CM94] T.F. Chan and T.P. Mathew. Domain decomposition algorithms. *Acta Numerica*, pages 61–143, 1994.
- [DP] Diffpack Home Page. <http://www.nobjects.com/Diffpack>.
- [SBG96] B.F. Smith, P.E. Bjørstad and W.D. Gropp. *Domain Decomposition, Parallel Multilevel Methods for Elliptic Partial Differential Equations*. Cambridge University Press, 1996.