CPU-measurements of some numerical PDE-applications (version 1.0)

Are Magnus Bruaset[*] Xing Cai[†] Hans Petter Langtangen[‡]
Klas Samuelsson[†] Aslak Tveito[§] Gerhard Zumbusch[*]

September 10, 1996

Contents

1 Introduction **2**

2 Model problems **2**
2.1 The elliptic boundary value problem 3
2.2 The parabolic problem 4
2.3 The nonlinear water wave problem 5
2.4 The species transport problem 6
2.5 Richard's equation in 1D (RE1) 7

3 Numerical methods **7**
3.1 Discretization of the problems 7
3.1.1 The HE2 problem 8
3.1.2 The nonlinear wave problem 8
3.1.3 The AD3 problem 8
3.1.4 The RE1 problem 9
3.2 Solution of the linear system of equations 9
3.3 Diffpack implementation and notations 10

4 Total CPU-time for the simulators **11**

5 Solving the linear system **15**
5.1 Summary 15
5.2 Efficiency of the linear algebra in Diffpack 19

6 Constructing the linear system **20**
6.1 Uniform grids 20
6.2 The mapped Laplace equation in the WA3 problem 20
6.3 Parabolic problem 21

7 CPU-measurements for PLTMG **23**
7.1 Numerical results 23

[*]SINTEF Applied Mathematics, P.O. Box 124 Blindern, N-0314 Oslo, Norway.
[†]Department of Informatics, University of Oslo, P.O. Box 1080 Blindern, N-0316 Oslo, Norway.
[‡]Mechanics Division, Dept. of Mathematics, University of Oslo, P.O. Box 1053 Blindern, N-0316 Oslo, Norway. Also at SINTEF Applied Mathematics.
[§]Department of Informatics, University of Oslo, P.O. Box 1080 Blindern, N-0316 Oslo, Norway. Also at SINTEF Applied Mathematics.

Identifier	Problem description
PE2	2D Poisson equation
PE3	3D Poisson equation
EB2	2D elliptic problem with variable coefficients
EB3	3D elliptic problem with variable coefficients
DP2	2D pressure equation with discontinuous coefficients
HE2	2D heat conduction equation
WA2	2D nonlinear water wave equation
WA3	3D nonlinear water wave equation
AD3	3D linear advection-diffusion equation
RE1	1D Richard's equation

Table 0.1: Identifiers for the test problems.

Solution method	Identifier	Diffpack name		Reference(s)
		`basic method`	`matrix type`	
Gauss elim.; banded matrix	*BG*	`GaussElim`	`MatBand`	[15]
Gauss elim.; sparse matrix	*SG*	`GaussElim`	`MatSparse`	[12], [14]
Jacobi iterations	*J*	`Jacobi`	`MatSparse`	[18], [25], [26]
Gauss-Seidel iterations	*GS*	`SOR`	`MatSparse`	[18], [25], [26]
Conjugate Gradient	*CG*	`ConjGrad`	`MatSparse`	[13], [19]
PCG + RILU prec. ($\omega = 0.97$)	*RPCG*	`ConjGrad`	`MatSparse`	[3], [16]
PCG + Fast Fourier prec.	*FPCG*	`ConjGrad`	`MatSparse`	[24]
Nested Multi-grid cycles	*NMG*	`DDSolver`	`MatSparse`	[17]

Table 0.2: Identifiers for all the solution methods included in the report and Diffpack names used in connection with `MenuSystem`.

1 Introduction

The purpose of the report is to gain some information about the CPU consumptions in a series of applications based on solving partial differential equations. Our aim is to be able to indicate suitable performance models for a number of methods and problems. For almost all of the applications involved, the simulation programs have been developed using Diffpack [11], which is a generic C++ library based on object-oriented programming techniques.

We first present the mathematical formulations of a series of model problems. Then we describe briefly the numerical methods to be used. Thereafter we list for each application the total CPU consumptions in the simulation. We also analyze the CPU data in detail by examining the two major parts of each simulation: The time for constructing the linear system of equations and the time for solving it. Later we discuss some implementation issues that are of importance for the efficiency of simulation programs.

2 Model problems

In this section we present the mathematical formulations of a collection of model problems. For each model problem, one or several test problems are specified.

2.1 The elliptic boundary value problem

We consider the second order boundary value problems of the form

$$\begin{aligned} -\nabla \cdot (\boldsymbol{K}\nabla u) &= f \quad \text{in } \Omega, \\ u &= u_D \quad \text{on } \Gamma_1, \\ \frac{\partial u}{\partial n} &= u_N \quad \text{on } \Gamma_2, \end{aligned} \tag{2.1}$$

where $\Gamma_1 \cup \Gamma_2 = \partial\Omega$ and $\Gamma_1 \cap \Gamma_2 = \emptyset$. Usually we have $\boldsymbol{K} = K\boldsymbol{I}$ where $\boldsymbol{I}$ denotes the identity (unity tensor) and $K > 0$ is a scalar function defined in Ω.

We begin with the simplest case, namely the Poisson equation with homogeneous Dirichlet boundary conditions,

$$\begin{aligned} -\nabla^2 u &= f \quad \text{in } \Omega, \\ u &= 0 \quad \text{on } \partial\Omega. \end{aligned}$$

The 2D Poisson equation (PE2)**:** For this 2D application the solution domain Ω is the unit square, i.e. $\Omega = [0,1]^2$. The source term is of the form

$$f(x,y) = e^{\sin(2\pi xy)}.$$

The 3D Poisson equation (PE3)**:** The solution domain Ω, in the 3D application, is the unit cube, i.e. $\Omega = [0,1]^3$. The source term is of the form

$$f(x,y,z) = e^{\sin(2\pi xyz)}.$$

The 2D elliptic problem with variable coefficients (EB2)**:** The solution domain Ω and the source term $f(x,y)$ are the same as in the PE2 problem. Homogeneous Dirichlet boundary conditions $u_D = 0$ are given on the entire boundary. The scalar function $K(x,y)$ is given by

$$K(x,y) = 1 + xy + (xy)^2.$$

The 3D elliptic problem with variable coefficients (EB3)**:** The boundary condition, the solution domain Ω and the source term $f(x,y,z)$ are the same as in the PE3 problem. The scalar function $K(x,y,z)$ has the following expression:

$$K(x,y,z) = 1 + xyz + (xyz)^2.$$

The 2D pressure equation with discontinuous coefficient (DP2)**:** The original two-dimensional solution domain Ω (see Figure 2.1) has curved boundaries. In addition the coefficient K of problem (2.1) contains discontinuities. More precisely, we have

$$K(x,y) = \begin{cases} \delta_i & \text{for } (x,y) \in \Omega_i, \ i = 1,2,3, \\ 1.0 & \text{for } (x,y) \in \Omega - \Omega_1 - \Omega_2 - \Omega_3. \end{cases}$$

Here $\delta_i = 10^{-i}$, $i = 1,2,3$. The boundary condition valid on the whole boundary is homogeneous Neumann, i.e. we have $\partial u/\partial n = 0$ on $\partial\Omega$. The source term f is of the form

$$f(x,y) = \begin{cases} 1 & \text{for } (x,y) \in [0.1625, 0.2] \times [0.2375, 0.275], \\ -1 & \text{for } (x,y) \in [0.8, 0.8375]^2, \\ 0 & \text{elsewhere,} \end{cases}$$

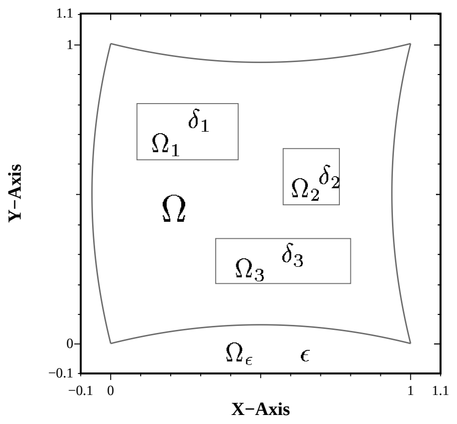

Figure 2.1: A non-rectangular solution domain after domain imbedding.

which satisfies the condition

$$\int_\Omega f dx = 0.$$

By a domain imbedding technique (see [1, 22]), we introduce a regularization parameter ϵ such that we solve

$$-\nabla \cdot (\boldsymbol{K}_\epsilon \nabla u_\epsilon) \quad = \quad f$$

in a larger and regular domain $\Omega_\epsilon = [-0.1, 1.1]^2$ where

$$K_\epsilon(x, y) \quad = \quad \begin{cases} K(x, y) & \text{for } (x, y) \in \Omega, \\ \epsilon & \text{for } (x, y) \in \Omega_\epsilon - \Omega. \end{cases}$$

For this application we have chosen $\epsilon = 10^{-9}$. In order to get a unique solution, we introduce the additional requirement $\int_{\Omega_\epsilon} u_\epsilon = 0$.

2.2 The parabolic problem

We consider a parabolic initial-boundary value problem,

$$\begin{aligned} \frac{\partial u}{\partial t} &= \nabla \cdot (K \nabla u) + f \quad \text{in } \Omega \times (0, T], \\ u &= u_D \quad \text{on } \Gamma \times (0, T], \\ \frac{\partial u}{\partial n} &= u_N \quad \text{on } \partial\Omega \backslash \Gamma \times (0, T], \\ u(\boldsymbol{x}, 0) &= u^0(\boldsymbol{x}) \quad \text{in } \Omega. \end{aligned} \qquad (2.2)$$

The 2D heat conduction equation (He2): The system (2.2) is to be solved in the spatial domain $\Omega = [0, 1]^2$ and in the time interval from 0 to $T = 1$. Using $K = 1$, the governing equation takes a simpler form:

$$\frac{\partial u}{\partial t} \quad = \quad \nabla^2 u + f,$$

and we use

$$\begin{aligned} f(x, y, t) \quad &= \quad e^t(\tilde{x}\tilde{y} + 2\tilde{x} + 2\tilde{y}), \\ \tilde{x} = x(1 - x), \qquad & \tilde{y} = y(1 - y), \end{aligned}$$

and

$$u^0(x,y) \;=\; xy(1-x)(1-y).$$

In addition, we assume that Dirichlet boundary conditions, $u = 0$, apply on the entire boundary.

2.3 The nonlinear water wave problem

Fully nonlinear free surface waves can be modelled by standard potential theory. We introduce the velocity potential $\varphi(\bar{x},\bar{y},\bar{z},t)$ and the free surface elevation $\eta(\bar{x},\bar{y},t)$ as the primary unknowns. The mathematical model consists of a coupled system of partial differential equations. Under the standard assumption that φ is divergence free and irrotational, the most important system takes the following form,

$$\begin{aligned}
\nabla^2\varphi &= 0 \quad \text{in the water volume,} && (2.3)\\
\eta_t + \varphi_{\bar{x}}\eta_{\bar{x}} + \varphi_{\bar{y}}\eta_{\bar{y}} - \varphi_{\bar{z}} &= 0 \quad \text{on the free surface,} && (2.4)\\
\varphi_t + \frac{1}{2}(\varphi_{\bar{x}}^2 + \varphi_{\bar{y}}^2 + \varphi_{\bar{z}}^2) + g\eta &= 0 \quad \text{on the free surface.} && (2.5)
\end{aligned}$$

Here (2.3) is the Laplace equation and equations (2.4) and (2.5) are referred to as the kinematic boundary condition and the dynamic boundary condition, respectively. For detailed mathematical description we refer to [7] and the references therein.

The water volume of interest can be written on the form

$$\overline{\Omega}(t) = \{(\bar{x},\bar{y},\bar{z}) \,|\, (\bar{x},\bar{y}) \in \Omega_{\bar{x}\bar{y}}, -H \le \bar{z} \le \eta(\bar{x},\bar{y},t)\}, \tag{2.6}$$

where $\partial\varphi/\partial n = 0$ on $\partial\overline{\Omega}$ except on the free surface. The time dependency of $\overline{\Omega}$ means that the Laplace equation (2.3) has to be solved in a dynamic computational domain. However, by introducing a time dependent transformation

$$x = \bar{x}, \quad y = \bar{y}, \quad z = \left(\frac{\bar{z}+H}{\eta+H} - 1\right)H, \tag{2.7}$$

we can instead solve a new boundary value problem on the form of (2.1) in a fixed computational domain at each time step. The time dependent coefficient matrix $\boldsymbol{K}$ of the new problem reads

$$\boldsymbol{K}(x,y,z,t) = \frac{1}{H}\begin{bmatrix} \eta+H & 0 & -(z+H)\eta_x \\ 0 & \eta+H & -(z+H)\eta_y \\ -(z+H)\eta_x & -(z+H)\eta_y & \dfrac{H^2+(z+H)^2(\eta_x^2+\eta_y^2)}{\eta+H} \end{bmatrix}. \tag{2.8}$$

Note that $\boldsymbol{K}$ is symmetric, positive definite. It is well-defined provided that the condition $|\eta| < H$ is satisfied. Because the $\bar{x}$- and $\bar{y}$-coordinates are the same as the x- and y-coordinates after the transformation, we will drop notations $\bar{x}$,$\bar{y}$ and use x, y instead. Thus the new system takes the following form:

$$\begin{aligned}
\nabla\cdot(\boldsymbol{K}\nabla\varphi) &= 0 \quad \text{in } \Omega = \Omega_{xy}\times[-H,0],\\
\eta_t + \varphi_x\eta_x + \varphi_y\eta_y - \varphi_z\frac{H}{\eta+H} &= 0 \quad \text{on the free surface,}\\
\varphi_t + \frac{1}{2}\left(\varphi_x^2 + \varphi_y^2 + \left(\frac{\varphi_z H}{\eta+H}\right)^2\right) + g\eta &= 0 \quad \text{on the free surface.}
\end{aligned}$$

The 2D nonlinear water wave equation (WA2): In this 2D application the wave motion is to be simulated in the $(\bar{x}, \bar{z})$ spatial coordinates. We solve the 2D wave system in the time interval $0 < t \leq T$, $T = 8$ in the computational domain

$$(x, z) \in [0, L] \times [-H, 0], \quad L = 160,\ H = 70,$$

which is bounded by the free water surface on the top and solid walls ($\partial\varphi/\partial n = 0$) on the rest part of the boundary. The initial conditions on the free surface are

$$\begin{aligned} \eta(x, 0) &= \eta^0(x), \\ \varphi(x, \eta^0(x), 0) = \varphi_z(x, \eta^0(x), 0) &= 0, \end{aligned}$$

where the initial form of the wave is given by

$$\eta^0(x) = \frac{729}{16}\left[\left(\frac{x}{L}\right)^2\left(\frac{x-L}{L}\right)^4 - \frac{1}{105}\right].$$

The 3D nonlinear water wave equation (WA3): The 3D wave system is to be solved in the time interval $0 \leq t \leq T$, $T = 4$ in the computational domain

$$(x, y, z) \in [0, L_1] \times [0, L_2] \times [-H, 0], \quad L_1 = L_2 = 80,\ H = 50$$

which is bounded by the free water surface on the top and solid walls on the rest part of the boundary. The initial conditions on the free surface are

$$\begin{aligned} \eta(x, y, 0) &= \eta^0(x, y), \\ \varphi(x, y, \eta^0(x, y), 0) = \varphi_z(x, y, \eta^0(x, y), 0) &= 0, \end{aligned}$$

where the initial form of the wave can be expressed by

$$\eta^0(x, y) = \left(-0.9\cos\left(\frac{\pi x}{L_1}\right) + \cos\left(\frac{2\pi x}{L_1}\right)\right)\left(1 - 0.9\cos\left(\frac{\pi y}{L_2}\right) + \cos\left(\frac{2\pi y}{L_2}\right)\right).$$

2.4 The species transport problem

Consider the following advection-diffusion-reaction equation with initial and boundary conditions:

$$\frac{\partial C}{\partial t} = \nabla \cdot (\mathbf{D}\nabla C) - \mathbf{v} \cdot \nabla C - kC^2 \quad \text{in } \Omega \times (0, T], \tag{2.9}$$

$$C = C_0 \quad \text{on } \Gamma_1 \times (0, T], \tag{2.10}$$

$$\frac{\partial C}{\partial n} = 0 \quad \text{on } \Gamma_2 \times (0, T], \tag{2.11}$$

$$C(\mathbf{x}, 0) = 0 \quad \text{in } \Omega. \tag{2.12}$$

Here $C(\mathbf{x}, t)$ is a solute concentration in a flow field with velocity $\mathbf{v}$, $\mathbf{D}$ is the hydrodynamic diffusion tensor, Ω is the domain of interest, and Γ_1 and Γ_2 denote the partition of the boundary: $\partial\Omega = \Gamma_1 \cup \Gamma_2$, with $\Gamma_1 \cap \Gamma_2 = \emptyset$.

The 3D linear advection-diffusion equation (AD3): With $k = 0$ equation (2.9) transforms into a linear equation. In this application we simply prescribe a constant velocity $\mathbf{v} = (1, 0, 0)^T$. The anisotropic $\mathbf{D}$ tensor is taken to be constant, more precisely,

$$\mathbf{D} = \begin{bmatrix} 0.01 & 0 & 0 \\ 0 & 0.001 & 0 \\ 0 & 0 & 0.001 \end{bmatrix}.$$

Moreover, $T = 1$ and Ω is a hypercube $(x, y, z) \in [0, 1] \times [0, 0.2] \times [0, 0.1]$ and Γ_1 is the plane $x = 0$ where $C_0 = 100$.

2.5 Richard's equation in 1D (Re1)

We consider a special case of the general multiphase flow and transport equations: The case of aqueous movement in a porous media system initially containing both a gas phase and an aqueous phase. For the limiting case of negligible changes in gas-phase pressure, Richard's equation (RE) [10] is applied as a mathematical model. RE is a single equation description of aqueous flow in a two-fluid porous media. It preserves many of the features of the general multiphase equations: severe nonlinearity, relatively complex constitutive relationships, and sharp-front solutions in space and time for certain sets of auxiliary conditions.

The compressible form of the one-dimensional RE considered here is

$$\begin{aligned} \frac{\partial \theta}{\partial t} + S_s S \frac{\partial \psi}{\partial t} &= \frac{\partial}{\partial z}\left[K\left(\frac{\partial \psi}{\partial z}+1\right)\right] \quad \text{in } \Omega \times (0,T], \\ \psi &= 0 \quad \text{for } z=0,\ 0 \le t \le T, \\ \psi &= \psi_L \quad \text{for } z=L,\ 0 \le t \le T, \\ \psi(z,0) &= -z \quad \text{in } \Omega, \end{aligned} \tag{2.13}$$

where $\Omega = [0, L]$ is the spatial domain of interest, θ is the volume fraction of the aqueous phase, ψ is the aqueous-phase pressure head, S is the aqueous-phase saturation, S_s is the specific storativity of the porous media resulting from compressibility of the aqueous phase, and z is the vertical coordinate. The unsaturated hydraulic conductivity K is a property of the porous media and a function of S, which in turn depends upon ψ. The quantities θ, S, and K can be related to the pressure ψ through constitutive relations.

As a specific application we choose $L = 10$, $T = 0.002$ and $\psi_L = 0.1$. The constitutive relations used to close the equations are

$$S_e = \frac{\theta - \theta_r}{\theta_s - \theta_r} = \begin{cases} (1+|\alpha\psi|^{n_v})^{-m_v}, & \text{for } \psi < 0, \\ 1, & \text{for } \psi \ge 0, \end{cases} \tag{2.14}$$

and

$$K = K_s S_e^{1/2}[1-(1-Se^{1/m_v})^{m_v}]^{1/2}, \tag{2.15}$$

where $m_v = 1 - 1/n_v$. The quantities α, n_v, and K_s are physical parameters of a porous medium.

3 Numerical methods

In this section we first briefly describe the numerical discretization of the model problems mentioned above. Thereafter we concentrate on the solution of the linear systems of equations involved in the applications. It is known that the choice of the solution method is important for the efficiency of simulations. Therefore different solution methods, essentially those falling into the category of iterative methods, will be considered. Finally we also discuss some implementation issues of Diffpack.

3.1 Discretization of the problems

The finite element discretization (see [2]) is applied for all the model problems. This involves primarily numerical calculation of the integrals of the weak formulation on the element level. The contributions are then added in the assembly process to construct a linear system of equations. For the Pe2, Pe3, Eb2 and Eb3 problems the discretization results in

$$\mathbf{Ax} = \mathbf{b}, \tag{3.1}$$

where $\mathbf{A}$ is a sparse, symmetric and positive definite matrix. The vector $\mathbf{x}$ contains the unknown values at the grid points. For the DP2 problem $\mathbf{A}$ is in fact only defined up to a constant because of the boundary condition. However, the additional requirement $\sum x_i = 0$ is introduced in order to get a unique solution. Uniform grids have been applied. Linear triangular elements are used in the two-dimensional applications, while trilinear elements are used in the three-dimensional applications.

3.1.1 The HE2 problem

For the HE2 problem we apply the finite element method for the spatial discretization and the fully implicit Euler scheme in the temporal discretization. At each time level, the computational task is nevertheless to solve a linear system of equations

$$\mathbf{A}\mathbf{u}^p = \mathbf{b}(\mathbf{u}^{p-1}). \tag{3.2}$$

Here $\mathbf{u}$ is the vector of nodal values of the approximate solution u. The superscript p denotes the time level. In the HE2 problem, $\mathbf{A}$ is independent of $\mathbf{u}$ and t and thus needs only to be computed once. We refer to section 6 for the computational details. Linear triangular elements in 2D are used in the finite element discretization.

3.1.2 The nonlinear wave problem

The free surface boundary conditions are discretized by the standard centered finite difference scheme. The values of η and $\varphi|_{\bar{z}=\eta}$, which will be used in the solution of the governing Laplace equation, are updated for each time level in a leapfrog manner. However, most of the computational effort is spent on the solution of the mapped Laplace equation, which is discretized by the finite element method. This means that a linear system of equations on the form (3.1) needs to be solved at each time level. The (banded) Gauss elimination or preconditioned conjugate gradient (PCG) method have been chosen as the solution method. In the PCG method the solution from the previous time level is used as the start vector, and the iteration is stopped when the L^2-norm of the residual is less than 10^{-8}. For the WA2 and WA3 problems, the new solution domain for the mapped Laplace equation is a rectangle or a hypercube, respectively. This enables a uniform partition, such that every finite element is of the same size. To be more specific, we use linear elements in 2D and trilinear elements in 3D. Note, however, that due to the mapping, a variable coefficient (2.8) is introduced and thus the element matrices differ.

3.1.3 The AD3 problem

For the AD3 application, we use the finite element method with trilinear elements of the uniform size. The discretization in time is based on backward (fully implicit) Euler scheme. The resulting discrete equations take the form

$$\mathbf{A}\,\mathbf{c}^p = \mathbf{b}(\mathbf{c}^{p-1})$$

on each time level. Here, $\mathbf{c}$ is the vector of nodal values of C. The superscript p denotes the time level. The coefficient matrix $\mathbf{A}$ is independent of $\mathbf{c}$ and t such that it needs not be recomputed at every time level. Thus the CPU-time will virtually be spent on the solution of the linear system .We apply the Bi-Conjugate Gradient (BiCGstab) method with Jacobi (diagonal) preconditioning. The result from the previous time level is used as the start vector. The iteration is stopped when the L^2-norm of the preconditioned residual divided by the L^2-norm of $\mathbf{b}$ is less than a prescribed tolerance 10^{-10}.

3.1.4 The Re1 problem

The formulation described in (2.13) is termed a mixed form Richard's equation (MFRE), since both θ and ψ appear as dependent variables. In our context, we expand the θ-term around the iteration level, m, in terms of ψ, followed by solution using Picard iteration [9]. By the truncated Taylor series approximation, we have

$$\theta_k^{p+1,m+1} = \theta_k^{p+1,m} + (\psi_k^{p+1,m+1} - \psi_k^{p+1,m}) \left.\frac{\partial \theta_k}{\partial \psi}\right|^{p+1,m} \quad \text{for } k = 1, ..., N, \tag{3.3}$$

where N is the number of nodes in the system.

The initial-boundary value problem is solved using the finite element method with linear elements and a backward Euler scheme in time. This yields a set of discrete nonlinear algebraic equations that are solved by Picard iteration (also called the method of successive substitution here). The mass matrix terms are lumped, and the iteration is terminated when the L^2-norm of the difference of the solution from two successive iterations, divided by the norm of the most recent solution, is less than a prescribed tolerance 10^{-4}. The coefficient matrices are formed at each iteration using three-point Gauss quadrature. Banded Gauss elimination is used to solve the tridiagonal linear system at each stage of the nonlinear iteration. The solution for the current time level is used as the initial guess for the nonlinear iteration at the next time level.

3.2 Solution of the linear system of equations

Linear systems of equations arise in the solution process of all the applications. However, different methods result in different computational behaviours. Table 3.1 contains a heuristic comparison of several "standard" methods, in respect of the computational cost, when applied to the elliptic model problem.

Method	2D	3D
Banded Gauss Elim.	$\mathcal{O}(N^2)$	$\mathcal{O}(N^{7/3})$
Nested Dissection	$\mathcal{O}(N^{3/2})$	$\mathcal{O}(N^2)$
Jacobi	$\mathcal{O}(N^2)$	$\mathcal{O}(N^2)$
Gauss-Seidel	$\mathcal{O}(N^2)$	$\mathcal{O}(N^2)$
Conjugate Gradient	$\mathcal{O}(N^{3/2})$	$\mathcal{O}(N^{4/3})$
CG + MILU	$\mathcal{O}(N^{5/4})$	$\mathcal{O}(N^{7/6})$
CG + FFT	$\mathcal{O}(N \log N)$	$\mathcal{O}(N \log N)$
Multigrid	$\mathcal{O}(N)$	$\mathcal{O}(N)$

Table 3.1: Comparison of different methods in respect of computational cost; N denotes the number of unknowns. These bounds apply to regular grids.

Generally, the solution methods can be divided into two categories: the direct methods and the iterative methods. However, the direct methods, with the banded Gauss (*BG*) elimination as a classical example, are rarely suitable for our applications. As an alternative we may apply iterative solution methods. For recent surveys of these iterative methods we refer to [5] for detailed explanations. In these methods, we start with an initial guess $\mathbf{x}^0$ and generate a sequence of approximations $\{\mathbf{x}^k\}$ which converge to the solution $\mathbf{x}$. Traditional iterative methods based on splittings of the coefficient matrix $\mathbf{A}$ can be represented by the Jacobi (*J*) and the Gauss-Seidel (*GS*) methods, while for a symmetric and positive definite $\mathbf{A}$ the conjugate gradient (*CG*) method (as one of the Krylov methods) is more appropriate. Krylov methods can be preconditioned to speed up the convergence. One particularly simple scheme is known as "incomplete LU-factorization" (ILU) [3, 16]. On hypercubes direct fast Poisson solver (FFT) are available. Used as more robust preconditioners (or iterative methods themselves) new methods such as domain decomposition [23] and multigrid [17] are under implementation in Diffpack.

The stopping criterion is another important part of the iterative methods. In other words, we continue the iterations until a prescribed tolerance of e.g. the residual is reached. There are several choices such as

- *Sc0*(ϵ): The relative stopping criterion for the deviation from the reference solution $\frac{||\boldsymbol{x}-\boldsymbol{x}_k||}{||\boldsymbol{x}||} < \epsilon$.
- *Sc1*(ϵ): The relative stopping criterion for the residual $\frac{||\boldsymbol{b}-\boldsymbol{A}\boldsymbol{x}_k||}{||\boldsymbol{b}-\boldsymbol{A}\boldsymbol{x}_0||} < \epsilon$.
- *Sc2*(ϵ): The absolute stopping criterion for the residua relating to the quasi L^2-norm
$$||\boldsymbol{b}-\boldsymbol{A}\boldsymbol{x}_k||_{L^2} < \epsilon, \quad \text{where } ||\boldsymbol{g}||_{L^2} \equiv \sqrt{\frac{1}{N}\sum_{i=1}^{N} g_i^2}.$$
- *Sc3*(ϵ): The absolute stopping criterion for the residual relating to the l^2-norm
$$||\boldsymbol{b}-\boldsymbol{A}\boldsymbol{x}_k||_{l^2} < \epsilon, \quad \text{where } ||\boldsymbol{g}||_{l^2} \equiv \sqrt{\sum_{i=1}^{N} g_i^2}.$$

Ideally, the chosen stopping criterion and tolerance level should be related to the size of the discretization error which only depends on grid and ansatz, in other words, the error caused by an incomplete solution of the system of equations should be negligible compared to the discretization error. In general the stopping criterion would therefore be problem dependent. For simplicity we have in the following experiments used a fixed tolerance level well below the discretization error.

We have observed from experiments that for most of our applications, the above stopping criteria behave similarly. Table 3.2 demonstrates the results concerning the number of iterations of the CG method when applied in the EB2 application. It should be noted here that *Sc0*(ϵ) is of little practical interest since it requires the solution we are seeking. The problem with *Sc3*(ϵ) is that it does not mimic a norm for continuous functions. Thus we regard *Sc1*(ϵ) and *Sc2*(ϵ) as the most appropriate alternatives. We apply *Sc1*(ϵ) for stationary problems: PE2, PE3, EB2, EB3, DP2; and *Sc2*(ϵ) for dynamic problems: HE2, WA2, WA3. The reason for not applying *Sc1*(ϵ) for dynamic problems is that it will grow stricter as time increases and better start vectors are available.

3.3 Diffpack implementation and notations

In Diffpack the management of the solution of the linear equation system can be controlled by the class `LinEqAdm` which is an interface with access to different choices of solution methods, stopping criteria, preconditioners etc.. On the other hand, the class `FEM` offers the standard finite element algorithms. Therefore a simulator class containing an object of `LinEqAdm` can be derived from `FEM`. The user can thus redefine the corresponding inherited member functions from `FEM` while including extra member functions suited for the specific problem. It is worth mentioning that for the extensive usage of the Diffpack classes `MenuSystem` can help to simplify the choice of algorithms and parameters.

Method	CG + fast Fourier transform				CG + RILU ($\omega = 0.9$)			
Criterion	$Sc0(\epsilon)$	$Sc1(\epsilon)$	$Sc2(\epsilon)$	$Sc3(\epsilon)$	$Sc0(\epsilon)$	$Sc1(\epsilon)$	$Sc2(\epsilon)$	$Sc3(\epsilon)$
9×9	10	12	10	11	7	8	6	7
17×17	12	14	11	13	10	12	9	11
33×33	12	15	11	13	14	16	11	14
65×65	12	16	10	13	23	27	18	23
129×129	12	16	9	13	42	49	29	42
257×257	12	16	8	12	81	95	52	79
513×513	12	16	7	12	159	188	97	146

Table 3.2: An example demonstrating the effect of different stopping criteria on the number of iterations for the EB2 application; $\epsilon = 10^{-8}$. The reference solution $\boldsymbol{x}$ is gained by using the *FPCG* method with the $Sc1(10^{-16})$ stopping criterion.

Criterion identifier	Diffpack name	Comments
$Sc0(\epsilon)$	`CMRelRefSolution`	
$Sc1(\epsilon)$	`CMRelResidual(CMRelTrueResidual)`	
$Sc2(\epsilon)$	`CMAbsResidual(CMAbsTrueResidual)`	`norm_tp=L2`
$Sc3(\epsilon)$	`CMAbsResidual(CMAbsTrueResidual)`	`norm_tp=l2`

Table 3.3: Identifiers of different stopping criteria and their Diffpack names.

4 Total CPU-time for the simulators

In this section we list for every application the total CPU consumptions in different simulations. Based on the data we also try to establish a performance model for the involved solution methods for almost all the applications. To be more specific, we use the number of the grid points in one space direction n as the major input parameter. For a particular solution method applied to a particular application we choose different n for different simulations. Thereafter we establish the performance model $C \cdot n^m$, where the exponent m is known by theory, by the appropriate fitting constant C. An example is depicted in Figure 4.1.

Machine identifier	`sgi1`	`sgi2`
Model	sgi-R4400(250MHz)	sgi-R8000(75MHz)
Memory	320M bytes	512M bytes
Flops per second	100M*	300M
Operating system	IRIX 5.3	IRIX 6.1
Compilation flags	`CC -mips2 -O`	`CC -mips4 -O`

Table 4.1: Specifications of different machines used to execute simulation programs. *) The number of flops per second displayed for `sgi1` applies actually for the sgi-R4400(200MHz) model.

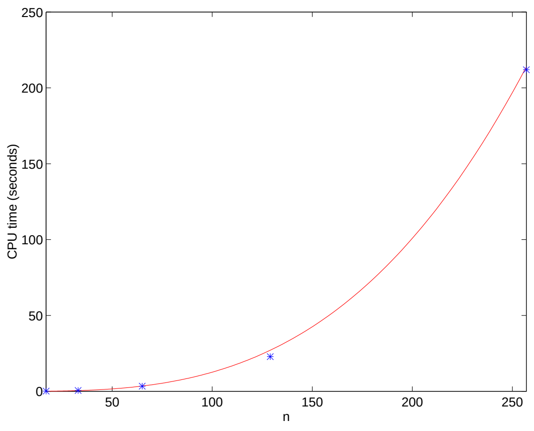

Figure 4.1: An example of establishing the performance model; the *CG* method is applied to the PE2 problem, n has discrete values 17, 33, 65, 129 and 257. The CPU measurements are marked by '$*$' while the curve represents the estimated performance model: $1.26 \times 10^{-5} \cdot n^3$.

Identifier	CPU-model	Method	Source
PE2	$6.0 \times 10^{-5} \cdot n^2 \log n$	*FPCG*	Table 4.3
PE3	$3.4 \times 10^{-4} \cdot n^3 \log n$	*FPCG*	Table 4.4
EB2	$4.7 \times 10^{-4} \cdot n^2$	*NMG*	Table 4.5
EB3	$1.3 \times 10^{-3} \cdot n^3$	*NMG*	Table 4.6
DP2	$1.6 \times 10^{-4} \cdot n^2 \log n$	*FPCG*	Table 4.7
HE2	$7.0 \times 10^{-6} \cdot n^{7/2}$	*RPCG*	Table 4.8
WA2	$2.1 \times 10^{-5} \cdot n^3 \log n$	*FPCG*	Table 4.9
WA3	$8.0 \times 10^{-5} \cdot n^4 \log n$	*FPCG*	Table 4.10

Table 4.2: Summary of the total CPU consumption for the test problems; the CPU models are associated with the best solution method applied in the simulations.The *NMG* method is applied only to the EB2 and EB3 problems.

n	17	33	65	129	257	513	CPU
N	17^2	33^2	65^2	129^2	257^2	513^2	model
BG	0.03	0.23	2.72	36.81			$1.3 \times 10^{-7} \cdot n^4$
J	1.04	14.98	243.72	4239.72			$1.5 \times 10^{-5} \cdot n^4$
GS	0.56	7.74	132.47	2063.86			$7.6 \times 10^{-6} \cdot n^4$
CG	0.11	0.54	3.37	22.84	211.98		$1.3 \times 10^{-5} \cdot n^3$
RPCG	0.10	0.41	1.99	10.66	70.18	492.17	$6.0 \times 10^{-5} \cdot n^{5/2}$
FPCG	0.09	0.31	1.32	5.22	22.03	96.70	$6.0 \times 10^{-5} \cdot n^2 \log n$

Table 4.3: Total CPU-time (in seconds) measured for the PE2 problem run on `sgi1`. The stopping criterion *Sc1*(10^{-8}) is used by the iterative methods.

n N	9 9^3	17 17^3	33 33^3	65 65^3	CPU model
BG	1.03	71.87	8981.96		$2.1 \times 10^{-7} \cdot n^7$
J	2.30	56.49	2199.73		$1.7 \times 10^{-6} \cdot n^6$
GS	1.26	30.98	1122.81		$8.6 \times 10^{-7} \cdot n^6$
CG	0.67	6.56	749.44	1089.44	$6.1 \times 10^{-5} \cdot n^4$
RPCG	0.67	5.97	57.52	645.33	$3.1 \times 10^{-4} \cdot n^{7/2}$
FPCG	0.55	4.69	40.20	430.79	$3.4 \times 10^{-4} \cdot n^3 \log n$

Table 4.4: Total CPU-time (in seconds) measured for the PE3 problem run on `sgi2`. The stopping criterion *Sc1*(10^{-8}) is used by the iterative methods.

n N	17 17^2	33 33^2	65 65^2	129 129^2	257 257^2	513 513^2	CPU model
BG	0.04	0.24	2.79	37.29			$1.4 \times 10^{-7} \cdot n^4$
J	1.02	14.82	251.04	4192.30			$1.5 \times 10^{-5} \cdot n^4$
GS	0.57	8.00	136.89	2034.49			$7.5 \times 10^{-6} \cdot n^4$
CG	0.12	0.64	4.27	32.09	308.07		$1.5 \times 10^{-5} \cdot n^3$
RPCG	0.11	0.43	1.93	11.02	70.02	541.36	$6.6 \times 10^{-5} \cdot n^{5/2}$
FPCG	0.11	0.43	1.83	7.66	32.56	182.63	$1.1 \times 10^{-4} \cdot n^2 \log n$
NMG	0.16	0.51	1.80	7.32	31.63	127.36	$4.7 \times 10^{-4} \cdot n^2$

Table 4.5: Total CPU-time (in seconds) measured for the EB2 problem run on `sgi1`. The stopping criterion *Sc1*(10^{-8}) is used by the iterative methods.

n N	9 9^3	17 17^3	33 33^3	65 65^3	CPU model
BG	1.10	72.71	8999.13		$2.0 \times 10^{-7} \cdot n^7$
J	2.40	56.66	2168.64		$1.7 \times 10^{-6} \cdot n^6$
GS	1.53	31.94	1121.55		$8.7 \times 10^{-7} \cdot n^6$
CG	0.75	7.62	94.81	2039.58	$1.1 \times 10^{-4} \cdot n^4$
RPCG	0.77	6.63	64.16	1307.90	$3.3 \times 10^{-4} \cdot n^{7/2}$
FPCG	0.73	6.20	55.69	1170.64	$5.0 \times 10^{-4} \cdot n^3 \log n$
NMG	0.87	5.81	48.73	509.41	$1.3 \times 10^{-3} \cdot n^3$

Table 4.6: Total CPU-time (in seconds) measured for the EB3 problem run on `sgi2`. The stopping criterion *Sc1*(10^{-8}) is used by the iterative methods.

n N	33 33^2	65 65^2	129 129^2	257 257^2	513 513^2	CPU model
J	24.75	372.27	5930.47			$2.1 \times 10^{-5} \cdot n^4$
GS	12.64	191.81	2957.34			$1.2 \times 10^{-5} \cdot n^4$
CG	1.86	13.85	112.39	2165.23		$5.2 \times 10^{-5} \cdot n^3$
RPCG	0.52	2.28	11.40	71.12	493.05	$6.7 \times 10^{-5} \cdot n^{5/2}$
FPCG	0.75	3.15	13.15	55.09	258.15	$1.6 \times 10^{-4} \cdot n^2 \log n$

Table 4.7: Total CPU-time (in seconds) measured for the DP2 problem run on `sgi1`. The stopping criterion *Sc1*(10^{-6}) is used by the iterative methods.

n	17	33	65	129	257	CPU
N	17^2	33^2	65^2	129^2	257^2	model
Δt	2^{-4}	2^{-5}	2^{-6}	2^{-7}	2^{-8}	
# steps	16	32	64	128	256	
BG	0.25	2.06	25.73	317.22		$1.2 \times 10^{-6} \cdot n^4$
J	6.81	115.78	1537.66	24346.28		$8.8 \times 10^{-5} \cdot n^4$
GS	3.69	54.98	809.90	12281.75		$4.5 \times 10^{-5} \cdot n^4$
CG	0.71	6.70	63.81	896.47		$3.2 \times 10^{-6} \cdot n^4$
RPCG	0.48	3.72	26.53	238.43	1799.91	$7.0 \times 10^{-6} \cdot n^{7/2}$

Table 4.8: Total CPU-time (in seconds) measured for the HE2 problem run on `sgi1`. The stopping criterion is $Sc2(10^{-10})$ is used by the iterative methods.

n	17	33	65	129	257	CPU
N	17^2	33^2	65^2	129^2	257^2	model
Δt	2^{-1}	2^{-2}	2^{-3}	2^{-4}	2^{-5}	
# steps	16	32	64	128	256	
BG	0.36	6.02	165.84	4675.75		$1.4 \times 10^{-7} \cdot n^5$
CG	1.40	18.03	261.64	4817.26		$1.7 \times 10^{-5} \cdot n^4$
RPCG	0.66	5.17	53.73	765.73	11637.98	$4.0 \times 10^{-5} \cdot n^{7/2}$
FPCG	0.54	3.70	27.77	223.05	1974.41	$2.1 \times 10^{-5} \cdot n^3 \log n$

Table 4.9: Total CPU-time (in seconds) measured for the WA2 problem run on `sgi1`. The stopping criterion $Sc2(10^{-8})$ is used by the iterative methods.

n	9	17	33	65	CPU
N	9^3	17^3	33^3	65^3	model
Δt	2^{-1}	2^{-2}	2^{-3}	2^{-4}	
# steps	8	16	32	64	
BG	8.09	1147.40			$1.6 \times 10^{-7} \cdot n^8$
J	93.36	4483.06	291206.15		$2.2 \times 10^{-4} \cdot n^6$
GS	48.52	2257.88	148243.08		$1.1 \times 10^{-4} \cdot n^6$
CG	5.31	108.85	3205.19		$8.2 \times 10^{-5} \cdot n^5$
RPCG	3.76	58.34	1109.35	21495.91	$1.5 \times 10^{-4} \cdot n^{9/2}$
FPCG	2.22	28.13	407.96	5954.33	$8.0 \times 10^{-5} \cdot n^4 \log n$

Table 4.10: Total CPU-time (in seconds) measured for the WA3 problem run on `sgi2`. The stopping criterion $Sc2(10^{-8})$ is used by the iterative methods.

Grid	$101 \times 21 \times 11$	$201 \times 21 \times 11$	$501 \times 11 \times 11$
Δt	10^{-2}	10^{-2}	10^{-2}
# steps	100	100	100
CPU	862.0	1644.8	2116.0

Table 4.11: Total CPU-time (in seconds) measured for the AD3 problem run on `sgi1`.

Grid points	801	1601	3201
Δt	10^{-3}	10^{-3}	10^{-3}
# steps	20	20	20
CPU	18.4	56.9	182.3

Table 4.12: Total CPU-time (in seconds) measured for the RE1 problem run on `sgi1`.

5 Solving the linear system

5.1 Summary

Solution of the linear systems of equations is the core part of each simulation. As discussed above, the choice of the solution method is important for the computational behaviour in each case. Therefore we investigate the CPU times spent on the solution of the linear systems further in this section.

Identifier	CPU-model	Method	Source
PE2	$3.2 \times 10^{-6} \cdot n^2 \log n$	*FPCG*	Table 5.2
PE3	$1.1 \times 10^{-5} \cdot n^3 \log n$	*FPCG*	Table 5.4
EB2	$9.2 \times 10^{-5} \cdot n^2$	*NMG*	Table 5.3
EB3	$2.1 \times 10^{-4} \cdot n^3$	*NMG*	Table 5.5
DP2	$7.8 \times 10^{-5} \cdot n^2 \log n$	*FPCG*	Table 5.6
HE2	$5.1 \times 10^{-6} \cdot n^{5/2}$	*RPCG*	Table 5.7
WA2	$1.1 \times 10^{-5} \cdot n^2 \log n$	*FPCG*	Table 5.8
WA3	$4.4 \times 10^{-5} \cdot n^3 \log n$	*FPCG*	Table 5.9

Table 5.1: Summary of the the solution of the linear systems for the test problems; the CPU models are associated with the best solution method applied in the simulations. The *NMG* method is applied only to the EB2 and EB3 problems.

n	17		33		65		129		257		513		CPU
N	17^2		33^2		65^2		129^2		257^2		513^2		model
	CPU	# it.	CPU	# it.	CPU	# it.	CPU	# it.	CPU	# it.	CPU	# it.	
BG	0.01	~	0.15	~	2.38	~	35.33	~					$1.3 \times 10^{-7} \cdot n^4$
J	0.97	945	14.69	3794	242.53	15187	4234.85	60751					$1.5 \times 10^{-5} \cdot n^4$
GS	0.49	474	7.45	1898	131.25	7594	2058.87	30376					$7.4 \times 10^{-6} \cdot n^4$
CG	0.03	45	0.22	94	2.11	192	17.73	389	190.62	791			$8.0 \times 10^{-6} \cdot n^3$
RPCG	0.02	12	0.09	16	0.69	27	5.40	49	48.10	96	393.53	189	$4.4 \times 10^{-5} \cdot n^{5/2}$
*FPCG**	0.00	1	0.01	1	0.07	1	0.27	1	1.20	1	5.22	1	$3.2 \times 10^{-6} \cdot n^2 \log n$

Table 5.2: Solution of the linear system in the PE2 problem run on `sgi1`; CPU-time (in seconds) and number of iterations. The stopping criterion *Sc1*(10^{-8}) is used by the iterative methods. *) For the Poisson equation the preconditioner is an exact solver and consequently we get convergence in one iteration.

n	17		33		65		129		257		513		CPU
N	17^2		33^2		65^2		129^2		257^2		513^2		model
	CPU	# it.	CPU	# it.	CPU	# it.	CPU	# it.	CPU	# it.	CPU	# it.	
BG	0.02	~	0.14	~	2.37	~	35.34	~					$1.3 \times 10^{-7} \cdot n^4$
SG	0.01	~	0.10	~	0.80	~	7.79	~	93.56	~			$3.6 \times 10^{-6} \cdot n^3 \log n$
J	0.94	935	14.52	3755	249.80	15030	4187.15	60122					$1.5 \times 10^{-5} \cdot n^4$
GS	0.49	468	7.70	1877	135.64	7512	2029.36	30054					$7.4 \times 10^{-6} \cdot n^4$
CG	0.04	51	0.32	116	2.98	254	26.64	542	285.57	1141			$1.2 \times 10^{-5} \cdot n^3$
RPCG	0.02	12	0.09	16	0.58	26	5.47	49	47.14	95	395.09	188	$4.4 \times 10^{-5} \cdot n^{5/2}$
FPCG	0.03	14	0.11	15	0.52	16	2.23	16	10.02	16	44.33	16	$2.7 \times 10^{-5} \cdot n^2 \log n$
NMG	0.07	6	0.16	6	0.39	5	1.53	5	7.11	5	24.04	4	$9.2 \times 10^{-5} \cdot n^2$

Table 5.3: Solution of the linear system in the EB2 problem run on `sgi1`; CPU-time (in seconds) and number of iterations. The stopping criterion *Sc1*(10^{-8}) is used by the iterative methods. The result of the *NMG* method is obtained by using the *Sc3*(10^{-8}) stopping criterion.

n N	9 9^3		17 17^3		33 33^3		65 65^3		CPU model
	CPU	# it.	CPU	# it.	CPU	# it.	CPU	# it.	
BG	0.93	$\sim$	71.04	$\sim$	8972.41	$\sim$			$2.1 \times 10^{-7} \cdot n^7$
J	1.76	231	51.96	940	2160.61	3770			$1.6 \times 10^{-6} \cdot n^6$
GS	0.92	117	26.35	472	1071.48	1888			$8.1 \times 10^{-7} \cdot n^6$
CG	0.14	27	2.04	57	40.32	118	617.51	241	$3.8 \times 10^{-5} \cdot n^4$
RPCG	0.13	8	1.43	12	18.61	18	224.91	31	$1.1 \times 10^{-4} \cdot n^{7/2}$
*FPCG**	0.02	1	0.16	1	1.37	1	12.88	1	$1.1 \times 10^{-5} \cdot n^3 \log n$

Table 5.4: Solution of the linear system in the PE3 problem run on `sgi2`; CPU-time (in seconds) and number of iterations. The stopping criterion *Sc1*(10^{-8}) is used by the iterative methods.

n N	9 9^3		17 17^3		33 33^3		65 65^3		CPU model
	CPU	# it.	CPU	# it.	CPU	# it.	CPU	# it.	
BG	0.94	$\sim$	71.33	$\sim$	8982.24	$\sim$			$2.0 \times 10^{-7} \cdot n^7$
J	1.77	230	51.68	936	2113.73	3753			$1.6 \times 10^{-6} \cdot n^6$
GS	0.91	117	26.65	469	1067.04	1878			$8.1 \times 10^{-7} \cdot n^6$
CG	0.14	28	2.43	68	52.27	154	937.56	340	$5.3 \times 10^{-5} \cdot n^4$
RPCG	0.15	8	1.47	12	21.45	18	228.15	31	$1.1 \times 10^{-4} \cdot n^{7/2}$
FPCG	0.12	11	1.07	13	12.34	14	90.04	15	$7.9 \times 10^{-5} \cdot n^3 \log n$
NMG	0.30	5	0.97	5	7.66	5	59.41	5	$2.1 \times 10^{-4} \cdot n^3$

Table 5.5: Solution of the linear system in the EB3 problem run on `sgi2`; CPU-time (in seconds) and number of iterations. The stopping criterion *Sc1*(10^{-8}) is used by the iterative methods. The result of the *NMG* method is obtained by using the *Sc3*(10^{-8}) stopping criterion.

n N	33 33^2		65 65^2		129 129^2		257 257^2		513 513^2		CPU model
	CPU	# it.	CPU	# it.	CPU	# it.	CPU	# it.	CPU	# it.	
J	24.37	5007	370.74	19619	5924.15	77666					$2.0 \times 10^{-5} \cdot n^4$
GS	12.27	2497	190.25	9817	2950.74	38847					$1.1 \times 10^{-5} \cdot n^4$
CG	1.49	505	12.29	1091	106.03	2248	3138.55	9048			$4.4 \times 10^{-5} \cdot n^3$
RPCG	0.14	21	0.74	30	5.00	51	44.03	95	366.68	186	$6.2 \times 10^{-5} \cdot n^{5/2}$
FPCG	0.36	32	1.56	34	6.52	34	27.62	35	128.34	36	$7.8 \times 10^{-5} \cdot n^2 \log n$

Table 5.6: Solution of the linear system in the DP2 problem run on `sgi1`; CPU-time (in seconds) and number of iterations. The stopping criterion *Sc1*(10^{-6}) is used by the iterative methods.

n N	17 17^2		33 33^2		65 65^2		129 129^2		257 257^2		CPU model
	CPU	# it.	CPU	# it.	CPU	# it.	CPU	# it.	CPU	# it.	
BG	0.00	$\sim$	0.02	$\sim$	0.16	$\sim$	1.24	$\sim$			$5.8 \times 10^{-7} \cdot n^3$
J	0.41	358.94	3.56	805.81	23.81	1546.22	189.41	2531.72			$8.7 \times 10^{-5} \cdot n^3$
GS	0.21	180.87	1.66	404.37	12.45	774.83	95.15	1267.51			$4.5 \times 10^{-5} \cdot n^3$
CG	0.03	22.13	0.15	33.84	0.79	45.06	6.18	67.85			$2.9 \times 10^{-6} \cdot n^3$
RPCG	0.01	7.00	0.06	7.94	0.21	7.20	0.97	6.46	3.92	5.23	$5.1 \times 10^{-6} \cdot n^{5/2}$

Table 5.7: Solution of the linear system in the He2 problem run on `sgi1`; averaged CPU-time (in seconds) and number of iterations at each time level. The stopping criterion *Sc2*(10^{-10}) is used by the iterative methods.

n N	17 17^2		33 33^2		65 65^2		129 129^2		257 257^2		CPU model
	CPU	# it.	CPU	# it.	CPU	# it.	CPU	# it.	CPU	# it.	
BG	0.01	$\sim$	0.15	$\sim$	2.38	$\sim$	35.38	$\sim$			$1.3 \times 10^{-7} \cdot n^4$
CG	0.07	109.81	0.51	203.37	3.87	384.39	36.78	729.51			$1.4 \times 10^{-5} \cdot n^3$
RPCG	0.03	14.56	0.10	18.75	0.62	27.69	5.11	45.48	42.02	84.09	$2.7 \times 10^{-5} \cdot n^{5/2}$
FPCG	0.02	7.69	0.06	6.97	0.22	6.69	0.91	6.07	3.97	5.87	$1.1 \times 10^{-5} \cdot n^2 \log n$

Table 5.8: Solution of the linear system in the Wa2 problem run on `sgi1`; averaged CPU-time (in seconds) and number of iterations at each time level. The stopping criterion *Sc2*(10^{-8}) is used by the iterative methods.

n N	9 9^3		17 17^3		33 33^3		65 65^3		CPU model
	CPU	# it.	CPU	# it.	CPU	# it.	CPU	# it.	
BG	0.94	$\sim$	71.01	$\sim$					$1.8 \times 10^{-7} \cdot n^7$
J	11.47	1518.23	279.04	5016.25	9092.16	16057.34			$2.3 \times 10^{-4} \cdot n^5$
GS	5.82	761.00	139.92	2500.50	4624.35	8055.25			$1.2 \times 10^{-4} \cdot n^5$
CG	0.46	95.37	5.58	158.12	91.90	269.50			$7.7 \times 10^{-5} \cdot n^4$
RPCG	0.28	15.25	2.50	18.81	26.85	23.16	277.16	33.77	$1.3 \times 10^{-4} \cdot n^{7/2}$
FPCG	0.09	6.37	0.64	6.50	5.22	6.25	36.81	5.39	$4.4 \times 10^{-5} \cdot n^3 \log n$

Table 5.9: Solution of the linear system in the Wa3 problem run on `sgi2`; averaged CPU-time (in seconds) and number of iterations at each time level. The stopping criterion *Sc2*(10^{-8}) is used by the iterative methods.

5.2 Efficiency of the linear algebra in Diffpack

The linear algebra tools (see [6]) in Diffpack have a carefully designed structure. By utilizing the object-oriented programming (OOP) techniques, Diffpack offers the users with a rich collection of solution methods, preconditioners and stopping criteria. The users are able to make flexible combinations at run time and the process of parameter fit is therefore greatly simplified. The generality is, of course, obtained at some cost of the computational efficiency. However, we will demonstrate through the following example that the linear algebra tools in Diffpack maintain a relatively high performance level.

As a competitor code, we study a specially coded C program `CGmilu` for the solution of 3D elliptic equations on the unit cube, i.e. the PE3 and EB3 problems. Here, `CGmilu` uses a very efficient finite difference method for the discretization which means that the CPU time spent on the construction of the linear system in `CGmilu` is almost negligible. We have thus found it fair to compare the CPU time spent on the solution of the linear system by the Diffpack simulator with the total CPU time consumed by the `CGmilu` program. Assuming homogeneous Dirichlet boundary conditions on the entire boundary, `CGmilu` considers only the values on the inner grid points as unknowns and therefore reduces the size of the resulting linear system. For the solution method, `CGmilu` restricts to the (preconditioned) CG method where a specially coded matrix-vector product routine guarantees the extraordinary efficiency of the program. We also mention that the Diffpack simulator and `CGmilu` use the same stopping criterion SC1(10^{-8}), so both programs converge under same number of CG iterations.

Solving the PE3 problem with *CG* (no prec.)					
System size	$9\times9\times9$	$17\times17\times17$	$33\times33\times33$	$65\times65\times65$	CPU model
Diffpack	0.04	0.60	11.14	235.06	$1.3\times10^{-5}\cdot n^4$
System size	$7\times7\times7$	$15\times15\times15$	$31\times31\times31$	$63\times63\times63$	CPU model
`CGmilu`	0.02	0.43	7.44	164.33	$8.5\times10^{-6}\cdot n^4$
Solving the PE3 problem with *CG*+MILU prec.					
System size	$9\times9\times9$	$17\times17\times17$	$33\times33\times33$	$65\times65\times65$	CPU model
Diffpack	0.04	0.46	6.40	97.58	$4.4\times10^{-5}\cdot n^{3.5}$
System size	$7\times7\times7$	$15\times15\times15$	$31\times31\times31$	$63\times63\times63$	CPU model
`CGmilu`	0.02	0.28	3.49	58.57	$2.1\times10^{-5}\cdot n^{3.5}$
Solving the EB3 problem with *CG* (no prec.)					
System size	$9\times9\times9$	$17\times17\times17$	$33\times33\times33$	$65\times65\times65$	CPU model
Diffpack	0.04	0.71	13.58	328.12	$1.8\times10^{-5}\cdot n^4$
System size	$7\times7\times7$	$15\times15\times15$	$31\times31\times31$	$63\times63\times63$	CPU model
`CGmilu`	0.02	0.52	9.65	231.29	$1.0\times10^{-5}\cdot n^4$
Solving the EB3 problem with *CG*+MILU prec.					
System size	$9\times9\times9$	$17\times17\times17$	$33\times33\times33$	$65\times65\times65$	CPU model
Diffpack	0.04	0.46	6.35	95.18	$4.3\times10^{-5}\cdot n^{3.5}$
System size	$7\times7\times7$	$15\times15\times15$	$31\times31\times31$	$63\times63\times63$	CPU model
`CGmilu`	0.02	0.27	3.48	57.51	$2.1\times10^{-5}\cdot n^{3.5}$

Table 5.10: Comparison between the Diffpack simulator and a specially coded C program `CGmilu` for the PE3 and EB3 problems. The CPU time (in seconds) are measured on `sgi2`.

6 Constructing the linear system

The linear system arising from the finite element discretization is computed in an element assembly process [21], i.e. looping through each element,

1. Calculate the element matrix and vector;
2. Enforce the essential boundary conditions;
3. Assemble the element contribution to the global linear system.

Steps 1 and 3 are the CPU-dominating parts of the whole assembly process. The CPU-time consumed is proportional to the number of elements, therefore the CPU-time for the whole assembly is proportional to the number of unknowns (grid points). However, the matrix storage strategy does influence the actual CPU consumption in step 3. Due to uniform grids $\mathbf{A}$ has a banded-structure where the bandwidth is typically n in 2D and n^2 in 3D, respectively. Each entry of $\mathbf{A}$ is thus easily accessed if a banded matrix storage strategy is used. But for each row of $\mathbf{A}$ there are only a small number (independent of n) of nonzeros. This means that a band matrix may waste too much storage on zero entries. To save the storage and the computational cost of matrix-vector products, a compressed sparse row storage strategy which only stores nonzero entries is preferable. In this case, the determination of the structure of $\mathbf{A}$ (location of the nonzeros) requires more work and a relatively complicated indexing procedure is needed to access each entry of $\mathbf{A}$. Therefore the efficiency of the assembly process will suffer. In the Diffpack context, for a matrix of type `MatSparse` more CPU-time is spent on constructing the linear system than for a matrix of type `MatBand`.

Grid	17×17	33×33	65×65	129×129
`MatBand`	0.02	0.06	0.26	1.17
`MatSparse`	0.07	0.27	1.16	4.60

Table 6.1: The CPU-time (in seconds) spent on the `makeSystem` function in the PE2 problem; different matrix storage strategies result in different CPU consumptions.

It is not difficult to see, from further analysis of the CPU consumptions, that the CPU-time spent on the construction of the linear system is often the dominating part of the entire computation. However, under certain circumstances, it is possible to improve the efficiency of the linear system construction process by implementing a suitable `makeSystem` function which runs more efficiently than the default `FEM::makeSystem`. We will explain this in detail by looking at the following examples.

6.1 Uniform grids

Uniform grids imply that every element is of the same shape. Accordingly, evaluations of basis functions and their derivatives will give the same result for a specific numerical integration point in any element. This means that we should avoid such unnecessary re-evaluations. For this purpose the command-line option `-bfeopt ON` can be very helpful. In addition, the property of constant K or f should be utilized such that recalculations of the element matrix/vector on every element are avoided.

6.2 The mapped Laplace equation in the WA3 problem

The special transformation (2.7) from the physical solution domain $\overline{\Omega}(t)$ (2.6) to a stationary computational domain Ω produces a new elliptic boundary value problem. The definition of $\boldsymbol{K}$ (2.8) indicates a relation between the element matrices for elements which only differ by z-coordinates. More precisely, we have in the WA3 problem

$$\begin{aligned}A_{i,j}^{I,J,K} = \int\int\int \frac{1}{H}\Bigg[&\frac{\partial N_i}{\partial x}\left((\eta+H)\frac{\partial N_j}{\partial x} - (z+H)\eta_x\frac{\partial N_j}{\partial z}\right)\\ &+\frac{\partial N_i}{\partial y}\left((\eta+H)\frac{\partial N_j}{\partial y} - (z+H)\eta_y\frac{\partial N_j}{\partial z}\right)\\ &-(z+H)\eta_x\frac{\partial N_i}{\partial z}\frac{\partial N_j}{\partial x} - (z+H)\eta_y\frac{\partial N_i}{\partial z}\frac{\partial N_j}{\partial y}\\ &+\frac{\left[H^2+(z+H)^2(\eta_x^2+\eta_y^2)\right]}{\eta+H}\frac{\partial N_i}{\partial z}\frac{\partial N_j}{\partial z}\Bigg]\,dxdydz\end{aligned}$$

for each matrix $\boldsymbol{A}_{i,j}$ on the element level.

Suppose we in this case give a tripled index to each element matrix and the element size is Δz in the z-direction. Then element matrices with same I, J indices will have such a resemblance that the difference between $A_{i,j}^{I,J,K}$ and $A_{i,j}^{I,J,1}$ consists of two parts which only vary with respect to $(K-1)\Delta z$; one linearly varying part and one quadratically varying part. More precisely,

$$A_{i,j}^{I,J,K} = A_{i,j}^{I,J,1} + (K-1)D_{i,j}^{I,J} + (K-1)^2R_{i,j}^{I,J}$$

where

$$\begin{aligned}D_{i,j}^{I,J} = -\Delta z\int\int\int\frac{1}{H}\Bigg[&\eta_x\left(\frac{\partial N_i}{\partial x}\frac{\partial N_j}{\partial z}+\frac{\partial N_i}{\partial z}\frac{\partial N_j}{\partial x}\right)\\ &+\eta_y\left(\frac{\partial N_i}{\partial y}\frac{\partial N_j}{\partial z}+\frac{\partial N_i}{\partial z}\frac{\partial N_j}{\partial y}\right) - 2\frac{(z+H)(\eta_x^2+\eta_y^2)}{\eta+H}\frac{\partial N_i}{\partial z}\frac{\partial N_j}{\partial z}\Bigg]\,dxdydz\end{aligned}$$

and

$$R_{i,j}^{I,J} = \Delta z^2\int\int\int\frac{1}{H}\frac{(\eta_x^2+\eta_y^2)}{\eta+H}\frac{\partial N_i}{\partial z}\frac{\partial N_j}{\partial z}\,dxdydz.$$

This clearly can simplify the construction of the linear system since a 3D assembly process is replaced by a 2D assembly process. In this way, major efficiency can be gained just by utilizing the properties of the integrand.

`makeSystem`	standard	special
CPU	41.44	1.55

Table 6.2: An example demonstrating the different behaviours of the standard `FEM::makeSystem` and a specially coded `makeSystem` function in the WA3 application; the grid size is $17 \times 17 \times 21$.

6.3 Parabolic problem

Let us consider a general parabolic problem on the form

$$\begin{aligned}\frac{\partial u}{\partial t} &= \mathcal{L}(u) + f \quad \text{in } \Omega\times[0,T],\\ u &= u_D \quad \text{on } \partial\Omega\times[0,T],\\ u(\mathbf{x},0) &= u^0(\mathbf{x}) \quad \text{in } \Omega,\end{aligned}$$

where $\mathcal{L}$ is a linear elliptic operator.

As usual we apply the finite element method for the spatial discretization and the θ-rule for the temporal discretization. This results in a linear equation system of equations on the form (3.2) that needs to be solved at each time level. However, the time dependency of $\mathcal{L}$, f and the essential boundary conditions u_D will determine the computational cost of both the assembly and the solution processes. To clarify this, it is beneficial to rewrite (3.2) in a different manner. Define

$$\mathbf{A} = \mathbf{M} + \theta\Delta t\mathbf{K} + \mathbf{A}_{\text{mod}}, \quad \mathbf{A}_{\text{rhs}} = \mathbf{M} + (\theta - 1)\Delta t\mathbf{K}, \quad \mathbf{b} = \mathbf{A}_{\text{rhs}}\mathbf{u}^{n-1} + \mathbf{c} + \mathbf{b}_{\text{mod}},$$

where $\mathbf{M}$ is the consistent mass matrix and $\mathbf{K}$ is the stiffness matrix relating to $\mathcal{L}$. Here, $\mathbf{c}$ contains contributions from the f term, while both $\mathbf{K}$ and $\mathbf{c}$ are constructed by the standard integral process without regards to the essential boundary conditions. $\mathbf{A}_{\text{mod}}$ and $\mathbf{b}_{\text{mod}}$ are modifications of the system matrix and the right hand side vector respectively due to the essential boundary condition u_D.

Remark: Let I denote the set of degrees of freedom for which there are essential boundary conditions. For a vector $\mathbf{u}_D$ containing the value of the essential boundary condition for $i \in I$ (and arbitrary values for $i \notin I$), we have

$$\tilde{b}_i = \mathbf{A}^{(i)}_{\text{mod}}\mathbf{u}_D, \quad \text{for } i \notin I, \tag{6.1}$$

where $\mathbf{b}_{\text{mod}} = (\tilde{b}_1, \tilde{b}_2, \ldots, \tilde{b}_N)^T$ and $\mathbf{A}^{(i)}_{\text{mod}}$ denotes the ith-row of $\mathbf{A}_{\text{mod}}$.

The mass matrix $\mathbf{M}$ depends only on the grid and can thus be calculated once and for all.

- For the simplest situation where $\mathcal{L}$, f and u_D are all independent of time, it is obvious that $\mathbf{A}$ remains the same for every time level. Moreover, only a matrix-vector product ($\mathbf{M}\mathbf{u}^{n-1}$) is needed to generate the right hand side vector $\mathbf{b}$ (see [20]).

- However, for the situation where only f is time dependent, $\mathbf{b}_{\text{mod}}$ will still be the same for all time levels. This allows us to skip the recalculation of the element matrix (and only recalculate the vector at the element level). In this case $\mathbf{A}$ needs only to be calculated once and we obtain at the same time $\mathbf{b}_{\text{mod}}$.

- When $\mathcal{L}$ is time dependent and f is not time dependent, it is still possible to achieve a slightly better performance by skipping the recalculation of $\mathbf{c}$.

- For a special case where both $\mathcal{L}$ and f are time independent while u_D is time dependent, only two matrix-vector products are needed to construct the linear system at all the time levels except the first one. This can be achieved by utilizing the relation (6.1) between $\mathbf{b}_{\text{mod}}$ and $\mathbf{A}_{\text{mod}}$.

Assume that we create a common simulator class for a parabolic problem. Based on the discussion above, we find it convenient to introduce three flags concerning the time-dependency of the linear operator $\mathcal{L}$, the source term f and the essential boundary condition u_D, respectively. We should redefine the `makeSystem` function accordingly aiming at improved efficiency.

Remark: In Table 6.3, the relations between different CPU consumptions for different situations of time dependency are rather of importance. For an application carried out on a non-uniform grid, the actual CPU-time will surely differ from the numbers listed in Table 6.3. However, the user will notice similar relations between different situations of time dependency, provided that the implementation idea concerning the construction of the linear system at each time level is applied.

time-dependency			CPU on `makeSystem`	
$\mathcal{L}$	f	u_D	first call	other calls
NO	NO	NO	5.49	0.05
NO	NO	YES	5.45	0.09
NO	YES	NO	5.36	0.67
NO	YES	YES	5.45	0.71
YES	NO		5.15	1.69
YES	YES		5.16	2.06

Table 6.3: Analysis of the specially implemented `makeSystem` function in a heat conduction simulator class; the CPU-times (in seconds) are measured on `sgi1` for applications on a uniform 129×129 grid.

7 CPU-measurements for PLTMG

PLTMG [4] is a program for solving boundary value problems in two-dimensional space. The equation is given in the form

$$-\nabla \bar{a}(x, y, u, \nabla u, \lambda) + f(x, y, u, \nabla u, \lambda) = 0 \quad \text{in } \Omega,$$

with boundary conditions

$$\begin{aligned} u &= g_1(x, y, \lambda) \quad \text{on } \Gamma_1, \\ \bar{a} \cdot n &= g_2(x, y, u, \lambda) \quad \text{on } \Gamma_2, \end{aligned}$$

where $\Gamma_1 \cup \Gamma_2 = \partial\Omega$ and $\Gamma_1 \cap \Gamma_2 = \emptyset$. Here $\bar{a}$ is the vector $(a_1, a_2)^t$ and λ is a continuation parameter (not used here).

PLTMG is a public domain software and is available at the file archive system Netlib. We have here used version 7.1 with double precision. PLTMG features mesh generation, adaptive refinement, error estimation and solution of the piecewise linear finite element equations with a hierarchical basis multigrid method.

We will here use PLTMG for solving three of the model problems solved by Diffpack in earlier sections: the 2D Poisson equation (PE2), the 2D elliptic boundary value problem (EB2), and the 2D pressure equation with discontinuous coefficient (DP2). The problems (PE2) and (EB2) will be solved with uniform meshes to ease comparison with the Diffpack computations. The problem DP2 will be solved with both uniform and adaptive refinement. For elliptic problems of (generalized) Poisson type, adaptivity may be important if the coefficients are strongly varying or the if the domain has reentrant corners. This is not the case for PE2 and EB2.

7.1 Numerical results

The triangulations used for PE2 and EB2 are constructed by dividing the unit square into 9 triangles of equal size. This initial mesh is then uniformly refined, the first entry in Table 7.1 with $N = 9^2$ corresponds to 2 refinements.

The fictitious domain method applied by the experiments with Diffpack is not appropriate for computations with PLTMG. It is better to directly approximate the non-rectangular outer domain with edges. Furthermore, it is advantageous to place edges along the internal edges where the coefficient $K(x, y)$ is discontinuous, cf. Figures 2.1 and 7.1.

Implementation of the boundary conditions are straightforward for PE2 and EB2. For DP2, with Neumann conditions on all boundaries, uniqueness of the solution is obtained by using Dirichlet boundary conditions on one edge of length 0.001 in the lower left corner, cf. Figure 7.2.

Remark: The refinement criterion used for DP2 is not the standard one in PLTMG, instead the minimization of the error of the quantity $||K(x,y)\nabla u||_{L_2(\Omega)}$ is the goal for the adaptivity. (This is implemented by a change in the subroutine `energy.f` of PLTMG.)

Parameters for solution of PE2 and EB2:

```
 s ir=3, f=1, l1=2, i=1
```

Explanations of the PLTMG parameters: `s`, the solver; `ir=3`, refine uniformly 3 times; `f=1`, solving first time; `l1=2`, use level 2 as coarsest level in multigrid solver; `i=1` symmetric matrix. For a detailed explanation of the parameters see [4].

n N	9 9^2	17 17^2	33 33^2	65 65^2	129 129^2	257 257^2	CPU model
PE2	0.11	0.44	1.73	6.97	31.45	143.63	$1.02 \times 10^{-4} \cdot n^{2.13}$
EB2	0.11	0.43	1.67	5.73	30.27	140.42	$1.02 \times 10^{-4} \cdot n^{2.12}$
N_{UNI}	245	903	3461	13545	53585		
DP2UNI	0.11	1.11	4.89	20.99	100.77		$6.17 \times 10^{-4} \cdot N^{1.10}$
N_{ADAP}	245	1005	4043	16189	53587		
DP2ADAP	0.11	1.85	8.59	28.00	109.43		$1.80 \times 10^{-3} \cdot N^{1.01}$

Table 7.1: Total CPU-time (in seconds) measured for the PLTMG simulations on `sgi1`; The meshes are uniformly or adaptively refined.

The number of multigrid iterations for PE2, EB2 and DP2UNI are 6-8, where about half is required to reach a residual corresponding to the stopping criterion *Sc1*(10^{-8}) used in the Diffpack experiments. In experiment DP2UNI the mesh in Figure 7.2 is used, the first mesh has 245 degrees of freedom, each uniform refinement increases the unknowns approximately by a factor of 4. In experiment DP2ADAP the mesh is adaptively refined where the meshes contains approximately the same number of degrees of freedom as in the uniformly refined case. Only two multigrid iterations are here performed by PLTMG.

Remarks: From Table 7.1 it is seen that the time consumption is essentially the same for all the model problems. PLTMG does not use that the problem is linear. The convergence rate of the multigrid iterations depends on the conditioning of the problem. Approximately 50% of the CPU-time is used for assembling the system of equations.

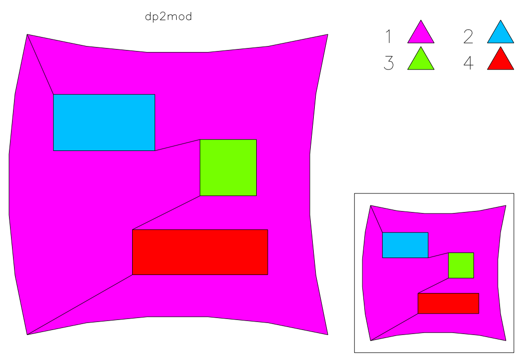

Figure 7.1: Skeleton for input of DP2 problem in PLTMG.

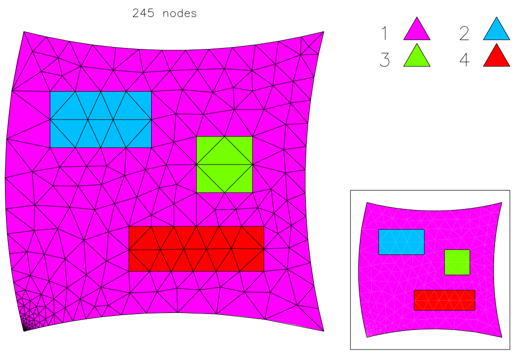

Figure 7.2: Mesh for the Dp2 problem in PLTMG generated from skeleton with mesh size parameter h=0.08.

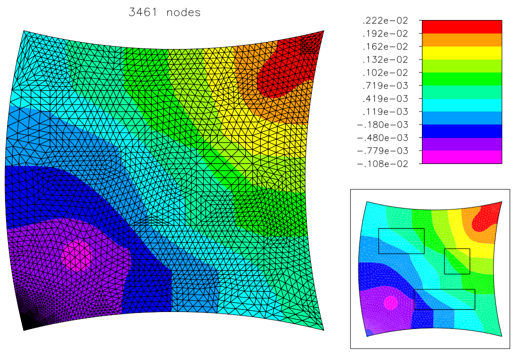

Figure 7.3: Solution for the Dp2 problem with PLTMG. The mesh is uniformly refined two times and has 3461 degrees of freedom.

The results show that for the Dp2 problem there is not much extra cost in increased CPU-time for using the adaptivity compared with uniformly refined meshes with the same number of degrees of freedom. But note that the uniformly refined meshes are solved to a higher accuracy. This accuracy is higher than that is necessary.

References

[1] G. B. Astrakhantsev, *Methods of fictitious domains for a second-order elliptic equation with natural conditions*, U.S.S.R. Comput. Maht. and Math. Phys., 18 (1978), pp. 114-121.

[2] O. Axelsson, V. A. Barker, *Finite Element Solution of Boundary Value Problems, Theory and Computation*, Academic Press, Orlando, FL, 1984.

[3] O. Axelsson, G. Lindskog, *On the eigenvalue distribution of a class of preconditioning methods*, Numer. Math., 48:479–498, 1986.

[4] R. E. Bank, *PLTMG: A Software Package for Solving Elliptic Partial Differential Equations, Users' Guide 7.0*, SIAM, Philadelphia, 1994.

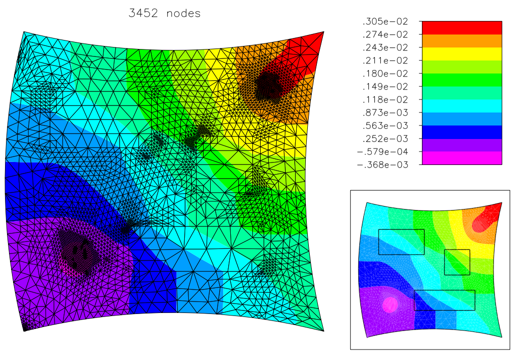

Figure 7.4: Solution for the Dp2 problem with PLTMG. The mesh is adaptively refined two times and contains 3452 degrees of freedom.

[5] A. M. Bruaset, *A Survey of Preconditioned Iterative Methods*, Pitman Research Notes In Mathematics Series, Longman Scientific & Technical, 1995.

[6] A. M. Bruaset and H. P. Langtangen, *Object-oriented design of preconditioned iterative methods*, Report no. STF33 A94036, Diffpack report, 1994.

[7] X. Cai, H. P. Langtangen, B. F. Nielsen and A. Tveito, *A finite element method for fully nonlinear water waves*, Preprint at the Department of Informatics, University of Oslo, 1996.

[8] X. Cai, B. F. Nielsen and A. Tveito, *An analysis of a preconditioner for the discretized pressure equation arising in reservoir simulation*, Preprint 1995-4 at The Department of Informatics, University of Oslo.

[9] M. A. Celia and E. T. Bouloutas. A general mass-conservative numerical solution for the unsaturated flow equation. *Water Resources Research*, 26(7):1483–1496, 1990.

[10] G. de Marsily. Contaminant immobilization and containments: hydraulics—a case study. In *Subsurface Restoration Conference—Third International Conference on Water Quality Research*, pages 32–33, National Center for Ground Water Research, Houston, TX, 1992.

[11] Diffpack Home Page, *http://www.oslo.sintef.no/diffpack*.

[12] I. S. Duff, A. M. Erisman and J. K. Reid, *Direct Methods for Sparse Matrices*, Oxford Science Publications, 1989.

[13] V. Faber and T. Manteuffel, *Necessary and sufficient conditions for the existence of a conjugate gradient method*, SIAM J. Numer. Anal., 21:352–362, 1984.

[14] A. George and J. W. Liu, *Computer Solution of Large Sparse Positive Definite Systems*, Prentice Hall, 1981.

[15] G. H. Golub and C. F. van Loan, *Matrix Computations*, Johns Hopkins University Press, 1989.

[16] I. Gustafsson, *A class of first order factorization methods*, BIT, 12 (1978), pp. 142–156.

[17] W. Hackbusch, *Multi-Grid Methods and Applications*, Springer-Verlag, Heidelberg, Berlin, 1985.

[18] L. A. Hageman and D. M. Young, *Applied Iterative Methods*, Academic Press, 1981.

[19] M. R. Hestenes and E. Stiefel, *Method of conjugate gradients for solving linear systems*, J. Res. Nat. Bur. Stand., 49:409–436, 1952.

[20] H. P. Langtangen, *Improving the efficiency of Diffpack simulators for PDEs with time independent coefficients*, Diffpack report, 1995.

[21] H. P. Langtangen, *Details of Finite Element Programming in Diffpack*, Report no. STF33 A94050, Diffpack report, 1994.

[22] B. F. Nielsen and A. Tveito, *On the approximation of the solution of the pressure equation by changing the domain*, to appear in SIAM J. Appl. Math.

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

[24] , *The methods of cyclic reduction, Fourier analysis and the FACR algorithm for the discrete solution of Poisson's equation on a rectangle*, SIAM Review, 19:490–501, 1977.

[25] R. S. Varga, *Matrix Iterative Analysis*, Prentice Hall, 1962.

[26] D. M. Young, *Iterative Solution of Large Linear Systems*, Academic Press, 1971.