CPU-measurements of some numerical PDE simulations in Diffpack

Are Magnus Bruaset* Xing Cai† Hans Petter Langtangen‡
Glenn Terje Lines† Klas Samuelsson† Wen Shen†
Aslak Tveito§ Gerhard Zumbusch¶

June 18, 1997

Contents

1 Introduction **3**

2 Model problems **3**
2.1 The elliptic boundary value problem 3
2.2 The parabolic problem 5
2.3 The nonlinear water wave problem 5
2.4 The two-phase flow problem in 2D (Tf2) 7
2.5 The species transport problem 7
2.6 Richard's equation in 1D (Re1) 8

3 Numerical methods **8**
3.1 The elliptic model problem 8
3.2 The Hc2 problem 9
3.3 The nonlinear water wave problem 9
3.4 The Tf2 problem 9
3.5 The Ad3 problem 10
3.6 The Re1 problem 10
3.7 Solution of $\mathbf{Ax} = \mathbf{b}$ 10
3.8 Diffpack implementation and notations 12

4 Total CPU-time consumptions **12**

5 Solution of the linear systems of equations **18**
5.1 Summary 18
5.2 Measurements 18
5.3 Efficiency of the linear algebra in Diffpack 22

*Numerical Objects AS, 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.
¶Institute for Applied Mathematics, Department for Scientific Computing and Numerical Simulation, University of Bonn, D-53113 Bonn, Germany.

6 Construction of the linear systems of equations **22**
6.1 The process 23
6.2 The mapped Laplace equation in the WA3 problem 23
6.3 Parabolic problems 24

7 Concluding remarks **25**

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
HC2	2D heat conduction equation
WA2	2D nonlinear water wave equation
WA3	3D nonlinear water wave equation
TF2	2D two-phase flow problem
AD3	3D linear advection-diffusion equation
RE1	1D Richard's equation

Table 0.1: Identifiers for the test problems included in the report.

Identifier	Solution method	Reference(s)
BG	Gauss elim. on banded matrix	[16]
SG	Gauss elim. on sparse matrix	[13], [15]
J	Jacobi iterations	[25], [26]
GS	Gauss-Seidel iterations	[25], [26]
CG	Conjugate gradient method	[14], [19]
MPCG	CG + MILU prec.	[3], [17]
RPCG	CG + RILU prec. ($\omega = 0.9$)	[3], [17]
FPCG	CG + fast Fourier trans. prec.	[24]
NMG	Nested multigrid cycles	[18]

Table 0.2: Identifiers for the solution methods for linear systems.

1 Introduction

The primary purpose of this report is to collect some information on the CPU-time consumptions in a series of typical numerical simulations for solving partial differential equations (PDEs). We would like to establish, through analyzing the CPU-measurements, the performance model for a number of numerical methods when applied in different model problems. All the simulators, i.e. the software programs carrying out the simulations, have been developed using Diffpack [12] which is a generic C++ library based on object-oriented programming techniques. Therefore, the established performance models may offer a rough prediction of real CPU-time consumptions by actual Diffpack simulators for practical problems. Additionally, the report can also be regarded as an investigation of the computational efficiency of the current Diffpack implementations. Last but not least, we wish to point out to Diffpack programmers some implementation issues that may affect the performance of the simulators.

The report is organized as follows. 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 every simulation the total CPU-time consumption. We also analyze the CPU-measurements 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. Finally we discuss some implementation issues that are of importance for the efficiency of the Diffpack simulators.

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.

The simplest case of the above model problem is 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 test problem 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 test problem, 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

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

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:

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

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

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

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

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

which satisfies the condition

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

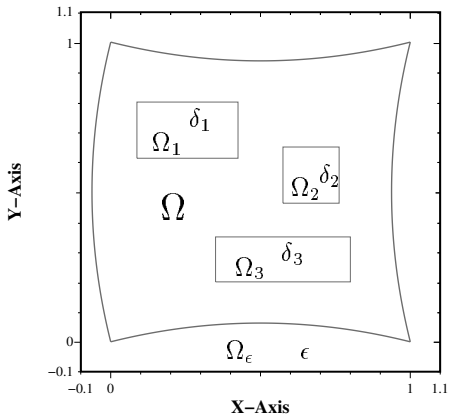

Figure 2.1: A non-rectangular solution domain after domain imbedding.

By a domain imbedding technique (see [1, 21]), 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 test problem 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

The following parabolic initial-boundary value problem is of interest,

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

The 2D heat conduction equation (Hc2): The model problem (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:

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

where

$$f(x,y,t) = e^t(\tilde{x}\tilde{y} + 2\tilde{x} + 2\tilde{y}), \quad \tilde{x} = x(1-x), \quad \tilde{y} = y(1-y).$$

The initial condition is

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

and 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,

$$\nabla^2 \varphi = 0 \quad \text{in the water volume,} \tag{2.3}$$

$$\eta_t + \varphi_{\bar{x}}\eta_{\bar{x}} + \varphi_{\bar{y}}\eta_{\bar{y}} - \varphi_{\bar{z}} = 0 \quad \text{on the free surface,} \tag{2.4}$$

$$\varphi_t + \frac{1}{2}(\varphi_{\bar{x}}^2 + \varphi_{\bar{y}}^2 + \varphi_{\bar{z}}^2) + g\eta = 0 \quad \text{on the free surface.} \tag{2.5}$$

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

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

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 test problem 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 $0 < t \le T$, $T = 8$ with the computational domain being

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

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 free surface 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 time interval of interest is chosen to be $0 \le t \le T$, $T = 4$. For this 3D test problem we run the simulations 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 ($\partial\varphi/\partial n = 0$) 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 free surface is 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 two-phase flow problem in 2D (TF2)

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

$$\begin{aligned} s_t + \vec{v} \cdot \nabla(f(s)) &= 0 \quad \text{in } \Omega \times (0,T] && (2.9)\\ -\nabla \cdot (\lambda(s)\nabla p) &= q \quad \text{in } \Omega \times (0,T] && (2.10)\\ \vec{v} &= -\lambda(s)\nabla p. && (2.11) \end{aligned}$$

In the above system of PDEs, s and p are the primary unknowns which represent the saturation of water and the pressure distribution, respectively. Equation (2.9) is referred to as the saturation equation which is a conservation law, and equation (2.10) is referred to as the pressure equation which is a second order elliptic boundary value problem. Here $\vec{v}$ is the total velocity field, λ is the mobility and f is the flux function. The source term q in (2.10) represents the injection and production wells. The boundary condition is $\vec{v} \cdot \vec{n} = 0$, which indicates no outflow on the boundary.

As a test problem, we consider the model with

$$f(s) = \frac{s^2}{s^2 + a(1-s)^2}, \qquad \lambda(s) = s^2 + a(1-s)^2,$$

where $a = 0.5$. Note that this flux function is of the Bucklett-Levrett type and is nonstrictly hyperbolic. The domain Ω is chosen to be the unit square, and one injection well is located at $(0,0)$, while one production well is at $(1,1)$. The contributions of the wells to the source term are treated as delta functions with proper strength, with positive value for injection well and negative value for production well. The test problem is to be simulated in the time interval $[0,T]$ with $T = 0.1$. The initial condition for s is s^0, where $s^0 = 1$ at the injection well and $s^0 = 0$ elsewhere.

2.5 The species transport problem

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

$$\begin{aligned} \frac{\partial C}{\partial t} &= \nabla \cdot (\mathbf{D}\nabla C) - \mathbf{v} \cdot \nabla C - kC^2 \quad \text{in } \Omega \times (0,T], && (2.12)\\ C &= C_0 \quad \text{on } \Gamma_1 \times (0,T], && (2.13)\\ \frac{\partial C}{\partial n} &= 0 \quad \text{on } \Gamma_2 \times (0,T], && (2.14)\\ C(\mathbf{x},0) &= 0 \quad \text{in } \Omega. && (2.15) \end{aligned}$$

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 (2.12) transforms into a linear equation. In this test problem 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 $C_0 = 100$.

2.6 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) [11] 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

$$\begin{aligned} \frac{\partial \theta}{\partial t} + S_s S \frac{\partial \psi}{\partial t} &= \frac{\partial}{\partial z}\left[K\left(\frac{\partial \psi}{\partial z} + 1\right)\right] \quad \text{in } \Omega \times (0,T], \\ \psi &= 0 \quad \text{for } z = 0,\ 0 \le t \le T, \\ \psi &= \psi_L \quad \text{for } z = L,\ 0 \le t \le T, \\ \psi(z,0) &= -z \quad \text{in } \Omega, \end{aligned} \tag{2.16}$$

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 test problem we choose $L = 10$, $T = 0.002$ and $\psi_L = 0.1$. The constitutive relations used to close the equations are

$$S_e = \frac{\theta - \theta_r}{\theta_s - \theta_r} = \begin{cases} (1 + |\alpha\psi|^{n_v})^{-m_v}, & \text{for } \psi < 0, \\ 1, & \text{for } \psi \ge 0, \end{cases} \tag{2.17}$$

and

$$K = K_s S_e^{1/2}[1 - (1 - Se^{1/m_v})^{m_v}]^{1/2}, \tag{2.18}$$

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 simulations. Finally we discuss some implementation issues of Diffpack.

3.1 The elliptic model problem

The finite element discretization (see [2]) plays a central role in the numerical solution of all the test problems derived from the second order elliptic model problem. 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 test problems PE2, PE3, EB2 and EB3, the FEM discretization results in a linear system of equations

$$\mathbf{A}\mathbf{x} = \mathbf{b}, \tag{3.1}$$

where $\mathbf{A}$ is a sparse, symmetric and positive definite matrix. The vector $\mathbf{x}$ contains the unknown values at the grid points. For test problem DP2, however, $\mathbf{A}$ is only defined up to a constant because of the boundary condition. Therefore, the additional requirement $\sum x_i = 0$ is introduced in order to get a unique solution. Different solution methods for linear systems will be discussed later in subsection 3.7.

Due to the simple shapes of the solution domains, uniform grids have been applied. Note that test problem DP2 is to be solved in the imbedded domain of rectangular shape. Linear triangular elements are used in the two-dimensional test problems, while trilinear elements are used in the three-dimensional test problems.

3.2 The HC2 problem

For the HC2 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 main computational task reduces to the solution of the following 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 HC2 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 details of constructing $\mathbf{A}$. Linear 2D triangular elements of the uniform size are used in the finite element discretization.

3.3 The nonlinear water wave problem

The fixed computational domain for the mapped Laplace equation is a rectangle for WA2 or a hypercube for WA3, 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. Due to the mapping, a variable coefficient (2.8) is introduced instead of the original constant coefficient. The free surface boundary conditions are thus discretized by the standard centered finite difference scheme. The values of η and $\varphi|_{\tilde{z}=\eta}$, 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.

3.4 The TF2 problem

The TF2 problem is solved by the Implicit-Pressure-Explicit-Saturation (IMPES) algorithm. This is a sequential algorithm where the saturation and pressure equations are solved separately on discrete time levels. The algorithm reads:

At time level t^n, given s^n, do the following iteration:

I: Solve the pressure equation to get p^n, using s^n in the mobility;

II: Compute the velocity $\vec{v}^n$ based on p^n;

III: Solve the saturation equation explicitly to get s^{n+1} on the next time level, using $\vec{v}^n$.

At step I, the pressure equation is solved by the standard finite element method (with linear elements) for the elliptic problems, as mentioned above in subsection 3.1. The saturation equation at step III is solved explicitly by an unsplit second order

Godunov scheme (cf. [4, 10, 22]). The time stepping process is adaptive where the CFL condition for the Godunov scheme is always ensured. This results in a nonuniform temporal discretization.

3.5 The AD3 problem

For the AD3 problem, we use the finite element method with trilinear elements of the uniform size. The discretization in time is based on the 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 quasi L^2-norm of the preconditioned residual divided by the quasi L^2-norm of $\mathbf{b}$ is less than a prescribed tolerance 10^{-10}.

3.6 The RE1 problem

The formulation described in (2.16) 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 quasi L^2-norm of the difference between the solutions from two successive iterations, divided by the norm of the most recent solution, is less than the 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.

3.7 Solution of $\mathbf{Ax} = \mathbf{b}$

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

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 in large simulations. Alternatively we may apply iterative solution methods. For a recent survey of these iterative

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 solution methods in respect of computational cost; N denotes the total number of unknowns. These bounds apply to linear systems arising from discretization on regular grids.

methods we refer to [5]. 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 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, 17]. On hypercubes direct fast Poisson solvers based on fast Fourier transform (FFT) are available as preconditioners.

Recently, new iterative methods such as domain decomposition [23] and multigrid [18] have emerged as more robust preconditioners or efficient iterative methods themselves. We like to mention that prototypical implementations for such methods are ready in Diffpack. However, optimization of the codes for e.g. *NMG* is yet premature, in comparison with the other methods included in the report. The reader should thus take into consideration the fact that the CPU measurements associated with the *NMG* method may not fully represent its numerical efficiency. Table 0.2 has summarized all the solution methods to be used in the simulations.

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 monitoring the deviation from the reference solution $\dfrac{\|\boldsymbol{x} - \boldsymbol{x}_k\|}{\|\boldsymbol{x}\|} < \epsilon$.
- *Sc1*(ϵ): The relative stopping criterion monitoring the residual $\dfrac{\|\boldsymbol{b} - \boldsymbol{A}\boldsymbol{x}_k\|}{\|\boldsymbol{b} - \boldsymbol{A}\boldsymbol{x}_0\|} < \epsilon$.
- *Sc2*(ϵ): The absolute stopping criterion monitoring the residual 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 monitoring the residual relating to the l^2-norm

$$\|\boldsymbol{b} - \boldsymbol{A}\boldsymbol{x}_k\|_{l^2} < \epsilon, \quad \text{where } \|\boldsymbol{g}\|_{l^2} \equiv \sqrt{\sum_{i=1}^{N} g_i^2}.$$

In general, the chosen stopping criterion and tolerance level should be related to the size of the discretization error,because we want the error caused by an incomplete solution of the system of equations to be negligible in comparison with the discretization error. Ideally, the stopping criterion should thus be grid dependent. However for simplicity, we have used a fixed tolerance level well below the discretization error in the simulations.

Moreover, we have observed that for most of our test problems, the above stopping criteria behave similarly. Table 3.2 demonstrates the results concerning the number of iterations of the preconditioned CG method when applied in test problem Eb2. It should be noted here that $Sc0(\epsilon)$ is of little practical interest since it requires the solution we are seeking. The problem with $Sc3(\epsilon)$ is that it does not mimic a norm for continuous functions. Thus we regard $Sc1(\epsilon)$ and $Sc2(\epsilon)$ as the most appropriate alternatives. We apply $Sc1(\epsilon)$ for stationary problems: Pe2, Pe3, Eb2, Eb3, Dp2; and $Sc2(\epsilon)$ for dynamic problems: Hc2, Wa2, Wa3 and Tf2. The reason for not applying $Sc1(\epsilon)$ for dynamic problems is that it will grow stricter as time increases and better start vectors are available.

Method	CG + fast Fourier trans. prec.				CG + RILU ($\omega = 0.9$) prec.			
Criterion	$Sc0(\epsilon)$	$Sc1(\epsilon)$	$Sc2(\epsilon)$	$Sc3(\epsilon)$	$Sc0(\epsilon)$	$Sc1(\epsilon)$	$Sc2(\epsilon)$	$Sc3(\epsilon)$
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 test problem Eb2 where $\epsilon = 10^{-8}$. The reference solution $\boldsymbol{x}$ needed in criterion $Sc0$ is gained by applying the $FPCG$ method with the $Sc1(10^{-16})$ stopping criterion.

3.8 Diffpack implementation and notations

In Diffpack, the solution of linear systems of equations is administrated by the interface class `LinEqAdm` which has 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 `MenuSystem` is usually used to enable different choices of solution methods and problem dependent parameters.

4 Total CPU-time consumptions

The total CPU-time consumption by any numerical simulation is a machine dependent amount which also depends on many other parameters, the most important of those are the size of the discretized problem and the choice of the numerical method for solving linear systems. Through analyzing measurements of the total CPU-time consumptions by different simulations, we intend to establish simplified performance models associated with different methods, as a function of grid size, for specific test problems on particular machine platforms. Hopefully, these performance models will

Identifier	Diffpack name		Comments
	`basic method`	`matrix type`	
BG	`GaussElim`	`MatBand`	
SG	`GaussElim`	`MatSparse`	
J	`Jacobi`	`MatSparse`	
GS	`SOR`	`MatSparse`	`relaxation parameter=1`
CG	`ConjGrad`	`MatSparse`	
MPCG	`ConjGrad`	`MatSparse`	`RILU relaxation parameter=1`
RPCG	`ConjGrad`	`MatSparse`	`RILU relaxation parameter=.9`
FPCG	`ConjGrad`	`MatSparse`	
NMG	`DDIter`	`MatSparse`	

Table 3.3: Identifiers for the solution methods and their Diffpack names used in connection with `MenuSystem`.

Identifier	Diffpack name	Comments
$Sc0(\epsilon)$	`CMRelRefSolution`	
$Sc1(\epsilon)$	`CMRelResidual(CMRelTrueResidual)`	
$Sc2(\epsilon)$	`CMAbsResidual(CMAbsTrueResidual)`	`norm type=L2`
$Sc3(\epsilon)$	`CMAbsResidual(CMAbsTrueResidual)`	`norm type=l2`

Table 3.4: Identifiers of different stopping criteria and their Diffpack names.

offer a rough prediction of real CPU-time consumptions by actual Diffpack simulators for practical problems.

As stated above, we want to establish simplified CPU performance models for numerical methods based on measurements of the total CPU-time consumptions. To be more specific, we use the number of grid points in one space direction n as the independent parameter. Then for a particular numerical method of interest, we run simulations for a specific test problem with different choices of n. By assuming that the simplified performance model takes the form

$$t \quad = \quad C \cdot n^m,$$

we can estimate the values of the exponent m and the constant C by a least square method. An example is depicted in Figure 4.1.

Machine identifier	`SGI`	`HP`
Model	SGI Power Challenge L	HP K260
CPU	sgi-R8000(75MHz)	hp-pa8000(180MHz)
Memory	512M bytes	2G bytes
Performance	300M Flops	19.4 SPECfp95
Operating system	IRIX 6.1	HP-UX 10.20
Compilation flags	`CC -mips4 -O`	`CC -O`

Table 4.1: Specifications of different machines used to run simulations.

Table 4.1 contains the technical specifications of the platforms on which the simulations have been run. Double precision is used in all the simulations. In particular, Diffpack libraries of version 2.6 have been used for the compilation of the simulators involved in the current report. Before listing for every test problem the total CPU-time consumptions by different simulations, we also present in Table 4.2 the best performance model for each test problem. The reader should note that the performance

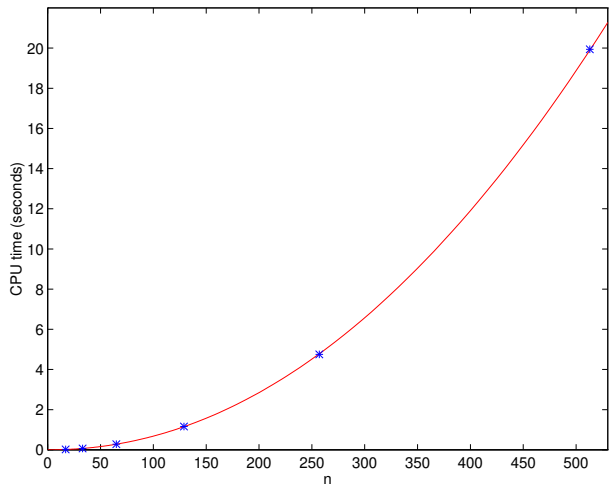

Figure 4.1: An example of establishing the performance model; the *FPCG* method is applied to the PE2 problem, n has discrete values 17, 33, 65, 129, 257 and 513. The CPU-time measurements are marked by '$*$' while the curve represents the estimated performance model: $5.11 \times 10^{-5} \cdot n^{2.06}$.

models are subject to changes since Diffpack undergoes continual modification. It should also be noted that the efficient *FPCG* method only applies to uniform grids. For more general domains and grids, methods like e.g. *NMG* will normally perform most efficiently.

Problem	CPU-model	Method	Source
PE2	$5.11 \times 10^{-5} \cdot n^{2.06}$	*FPCG*	Table 4.3
PE3	$1.12 \times 10^{-4} \cdot n^{3.09}$	*FPCG*	Table 4.4
EB2	$1.27 \times 10^{-4} \cdot n^{2.05}$	*NMG*	Table 4.5
EB3	$1.38 \times 10^{-4} \cdot n^{3.15}$	*FPCG*	Table 4.6
DP2	$5.86 \times 10^{-5} \cdot n^{2.22}$	*FPCG*	Table 4.7
HC2	$3.21 \times 10^{-5} \cdot n^{3.10}$	*MPCG*	Table 4.8
WA2	$3.19 \times 10^{-5} \cdot n^{3.10}$	*FPCG*	Table 4.9
WA3	$1.11 \times 10^{-4} \cdot n^{3.99}$	*FPCG*	Table 4.10
TF2	$1.76 \times 10^{-4} \cdot n^{3.07}$	*FPCG*	Table 4.11

Table 4.2: Summary of the best performance model for each test problem based on measurements of the total CPU-time consumptions.

n / N	17 / 17^2	33 / 33^2	65 / 65^2	129 / 129^2	257 / 257^2	513 / 513^2	CPU model
BG	0.02	0.09	0.77	9.17			$6.04 \times 10^{-7} \cdot n^{3.39}$
J	0.33	4.62	99.21	1907.32			$9.26 \times 10^{-7} \cdot n^{4.42}$
GS	0.17	2.32	43.55	906.35			$5.16 \times 10^{-7} \cdot n^{4.38}$
CG	0.03	0.12	0.95	7.76	74.61		$1.63 \times 10^{-6} \cdot n^{3.17}$
$MPCG$	0.03	0.10	0.55	3.28	20.56	119.84	$1.05 \times 10^{-5} \cdot n^{2.61}$
$FPCG$	0.03	0.08	0.27	1.15	4.72	19.59	$5.11 \times 10^{-5} \cdot n^{2.06}$

Table 4.3: Total CPU-time consumptions (in seconds) of the PE2 simulations measured on HP. 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	0.23	16.61	3846.97		
J	0.24	9.05	353.43		$1.08 \times 10^{-6} \cdot n^{5.61}$
GS	0.16	4.80	172.35		$1.18 \times 10^{-6} \cdot n^{5.37}$
CG	0.09	0.84	10.58	147.72	$1.51 \times 10^{-5} \cdot n^{3.85}$
$MPCG$	0.10	0.82	8.49	90.13	$4.02 \times 10^{-5} \cdot n^{3.50}$
$FPCG$	0.10	0.69	5.63	43.25	$1.12 \times 10^{-4} \cdot n^{3.09}$

Table 4.4: Total CPU-time consumptions (in seconds) of the PE3 simulations measured on HP. 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	0.02	0.09	0.80	9.39			$5.74 \times 10^{-7} \cdot n^{3.41}$
J	0.33	4.50	87.60	1808.28			$9.39 \times 10^{-7} \cdot n^{4.40}$
GS	0.17	2.32	48.01	894.78			$5.53 \times 10^{-7} \cdot n^{4.37}$
CG	0.03	0.14	1.08	9.55	102.64		$1.03 \times 10^{-6} \cdot n^{3.31}$
$MPCG$	0.03	0.11	0.62	3.50	20.70	119.47	$1.05 \times 10^{-5} \cdot n^{2.61}$
$FPCG$	0.04	0.11	0.53	2.09	9.43	41.36	$7.00 \times 10^{-5} \cdot n^{2.13}$
NMG	0.06	0.18	0.66	2.78	11.80	45.57	$1.27 \times 10^{-4} \cdot n^{2.05}$

Table 4.5: Total CPU-time consumptions (in seconds) of the EB2 simulations measured on HP. 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	0.23	16.60	3946.26		
J	0.25	9.08	338.49		$1.29 \times 10^{-6} \cdot n^{5.55}$
GS	0.17	4.89	171.47		$1.40 \times 10^{-6} \cdot n^{5.32}$
CG	0.10	1.00	13.96	198.47	$1.41 \times 10^{-5} \cdot n^{3.94}$
$MPCG$	0.11	0.93	9.52	96.37	$5.19 \times 10^{-5} \cdot n^{3.46}$
$FPCG$	0.13	1.02	8.52	69.55	$1.38 \times 10^{-4} \cdot n^{3.15}$
NMG	0.16	1.25	10.66	87.44	$1.61 \times 10^{-4} \cdot n^{3.17}$

Table 4.6: Total CPU-time consumptions (in seconds) of the EB3 simulations measured on HP. The stopping criterion $Sc1(10^{-8})$ is used by the iterative methods.

n	33	65	129	257	513	CPU
N	33^2	65^2	129^2	257^2	513^2	model
J	8.32	192.08	3952.63			$1.16 \times 10^{-6} \cdot n^{4.52}$
GS	4.27	107.58	2135.76			$5.35 \times 10^{-7} \cdot n^{4.56}$
CG	0.45	4.91	57.63	1184.82		$2.61 \times 10^{-7} \cdot n^{3.99}$
$RPCG$	0.10	0.56	4.89	37.75	292.42	$1.93 \times 10^{-6} \cdot n^{3.02}$
$FPCG$	0.16	0.60	2.96	13.42	52.29	$5.86 \times 10^{-5} \cdot n^{2.22}$

Table 4.7: Total CPU-time consumptions (in seconds) of the DP2 simulations measured on HP. 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	0.14	1.14	11.53	125.60		$6.54 \times 10^{-6} \cdot n^{3.45}$
J	2.71	42.49	738.51	16671.10		$9.10 \times 10^{-6} \cdot n^{4.38}$
GS	1.55	23.74	375.92	9065.49		$5.30 \times 10^{-6} \cdot n^{4.36}$
CG	0.29	2.58	28.32	509.61		$3.08 \times 10^{-6} \cdot n^{3.88}$
$MPCG$	0.23	1.61	13.52	116.66	926.31	$3.21 \times 10^{-5} \cdot n^{3.10}$

Table 4.8: Total CPU-time consumptions (in seconds) of the HC2 simulations measured on HP. The stopping criterion $Sc2(10^{-10})$ 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	0.13	1.86	43.85	1149.77		$1.28 \times 10^{-7} \cdot n^{4.71}$
CG	0.55	6.07	124.73	2450.55		$1.72 \times 10^{-6} \cdot n^{4.40}$
$MPCG$	0.29	2.29	25.43	376.11	4063.00	$5.98 \times 10^{-6} \cdot n^{3.67}$
$FPCG$	0.26	1.63	13.22	105.61	954.85	$3.19 \times 10^{-5} \cdot n^{3.10}$

Table 4.9: Total CPU-time consumptions (in seconds) of the WA2 simulations measured on HP. 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	1.60	257.97			$3.78 \times 10^{-8} \cdot n^{7.99}$
CG	1.33	46.52	1345.77		$2.66 \times 10^{-5} \cdot n^{5.07}$
$MPCG$	1.05	24.67	487.08	9845.04	$7.96 \times 10^{-5} \cdot n^{4.47}$
$FPCG$	0.61	9.01	128.30	1901.05	$1.11 \times 10^{-4} \cdot n^{3.99}$

Table 4.10: Total CPU-time consumptions (in seconds) of the WA3 simulations measured on HP. The stopping criterion $Sc2(10^{-8})$ 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
# steps	36	70	142	289	588	
RPCG	1.39	9.13	95.29	1290.04	15209.59	$2.67 \times 10^{-5} \cdot n^{3.63}$
FPCG	1.40	9.20	66.19	530.19	4527.20	$1.76 \times 10^{-4} \cdot n^{3.07}$
NMG	3.47	15.24	98.75	813.07	6593.82	$2.86 \times 10^{-4} \cdot n^{3.06}$

Table 4.11: Total CPU-time consumptions (in seconds) of the TF2 simulations measured on HP. The stopping criterion $Sc2(10^{-10})$ 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 4.12: Total CPU-time consumptions (in seconds) of the AD3 simulations measured on HP.

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

Table 4.13: Total CPU-time consumptions (in seconds) of the PE2 simulations measured on HP.

5 Solution of the linear systems of equations

The preceding section presents the information on total CPU-time consumptions by different simulations. We proceed our analysis further by examining how the CPU-times have been spent.

Roughly, the CPU-time consumption of a Diffpack simulation may be divided into three parts:

1. Preparation – parameter input, grid generation etc;
2. Discretization – construction of the linear systems of equations;
3. Solution of the linear system of equations.

For the test problems included in this report, Part 1 normally takes under 5% of the total CPU-time consumption and is therefore negligible. Between the two remaining parts, Part 3 is considered the core part of each simulation because it usually spends more CPU-time. This section thus examines the CPU-time consumptions on the solution of the linear systems further. Performance models are established accordingly.

5.1 Summary

As discussed in subsection 3.7, the choice of the solution method is important for the computational behavior in each simulation. Table 5.1 thus summarizes the method with best performance for each test problem.

Problem	CPU-model	Method	Source
Pe2	$2.82 \times 10^{-6} \cdot n^{2.21}$	*FPCG*	Table 5.2
Pe3	$9.17 \times 10^{-6} \cdot n^{3.04}$	*FPCG*	Table 5.4
Eb2	$5.00 \times 10^{-5} \cdot n^{2.03}$	*NMG*	Table 5.3
Eb3	$2.64 \times 10^{-5} \cdot n^{3.28}$	*FPCG*	Table 5.5
Dp2	$4.76 \times 10^{-5} \cdot n^{2.21}$	*FPCG*	Table 5.6
Hc2	$2.07 \times 10^{-5} \cdot n^{2.06}$	*MPCG*	Table 5.7
Wa2	$1.34 \times 10^{-5} \cdot n^{2.17}$	*FPCG*	Table 5.8
Wa3	$7.15 \times 10^{-5} \cdot n^{2.93}$	*FPCG*	Table 5.9
Tf2	$5.49 \times 10^{-5} \cdot n^{2.05}$	*FPCG*	Table 5.10

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.

5.2 Measurements

The following are measurements of the (averaged) CPU-time consumptions on solving the linear system of equations in every simulation. For iterative methods, the (averaged) number of iterations needed is also listed. Theoretical estimates on the computational cost for different methods (Table 3.1) apply to test problems Pe2, Pe3, Eb2 and Eb3, which are derived from the elliptic model problem. The performance models based on the actual measurements agree quite nicely with the theoretical estimates.

n / N	17 / 17^2		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.	CPU	# it.	
BG	0.01	$\sim$	0.04	$\sim$	0.60	$\sim$	8.45	$\sim$					$4.43 \times 10^{-8} \cdot n^{3.93}$
J	0.32	945	4.56	3794	98.95	15187	1906.29	60751					$8.86 \times 10^{-7} \cdot n^{4.43}$
GS	0.16	474	2.26	1898	43.30	7594	905.33	30376					$4.73 \times 10^{-7} \cdot n^{4.40}$
CG	0.01	45	0.06	94	0.71	192	6.74	389	70.33	791			$6.09 \times 10^{-7} \cdot n^{3.34}$
MPCG	0.01	16	0.04	24	0.30	36	2.26	55	16.44	83	103.06	125	$3.35 \times 10^{-6} \cdot n^{2.77}$
*FPCG**	0.01	1	0.01	1	0.03	1	0.12	1	0.58	1	2.86	1	$2.82 \times 10^{-6} \cdot n^{2.21}$

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

n / N	17 / 17^2		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.	CPU	# it.	
BG	0.00	$\sim$	0.04	$\sim$	0.59	$\sim$	8.48	$\sim$					$4.36 \times 10^{-8} \cdot n^{3.93}$
SG	0.01	$\sim$	0.10	$\sim$	0.80	$\sim$	7.79	$\sim$	93.56	$\sim$			$\cdot n^3 \log n$
J	0.31	935	4.43	3755	87.33	15030	1807.17	60122					$8.91 \times 10^{-7} \cdot n^{4.41}$
GS	0.15	468	2.25	1877	47.75	7512	893.68	30054					$4.99 \times 10^{-7} \cdot n^{4.39}$
CG	0.01	51	0.07	116	0.81	254	8.83	541	98.03	1142			$3.83 \times 10^{-7} \cdot n^{3.49}$
MPCG	0.00	15	0.04	24	0.34	36	2.39	54	16.20	81	101.26	123	$4.54 \times 10^{-6} \cdot n^{2.71}$
FPCG	0.02	14	0.05	15	0.25	16	0.97	16	4.94	16	23.13	16	$1.39 \times 10^{-5} \cdot n^{2.30}$
NMG	0.02	7	0.06	7	0.18	6	0.90	6	4.15	6	14.74	5	$5.00 \times 10^{-5} \cdot n^{2.03}$

Table 5.3: Solution of the linear system in the EB2 simulations; CPU-time (in seconds) measured on HP 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	0.18	$\sim$	16.23	$\sim$	3842.83	$\sim$			
J	0.16	231	8.45	940	348.51	3770			$3.90 \times 10^{-7} \cdot n^{5.91}$
GS	0.08	117	4.19	472	167.35	1888			$2.10 \times 10^{-7} \cdot n^{5.88}$
CG	0.01	27	0.24	57	5.66	118	108.02	241	$6.26 \times 10^{-7} \cdot n^{4.55}$
MPCG	0.02	12	0.21	20	3.54	30	50.37	48	$2.05 \times 10^{-6} \cdot n^{4.09}$
*FPCG**	0.00	1	0.05	1	0.37	1	2.93	1	$9.17 \times 10^{-6} \cdot n^{3.04}$

Table 5.4: Solution of the linear system in the PE3 simulations; CPU-time (in seconds) measured on HP 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	0.17	$\sim$	16.04	$\sim$	3939.41	$\sim$			
J	0.16	230	8.36	936	332.65	3753			$4.24 \times 10^{-7} \cdot n^{5.88}$
GS	0.08	117	4.17	469	165.63	1878			$2.14 \times 10^{-7} \cdot n^{5.87}$
CG	0.01	28	0.29	68	8.14	154	151.59	341	$5.71 \times 10^{-7} \cdot n^{4.67}$
MPCG	0.01	12	0.21	19	3.65	30	49.54	47	$2.15 \times 10^{-6} \cdot n^{4.07}$
FPCG	0.03	11	0.28	13	2.62	14	22.76	15	$2.64 \times 10^{-5} \cdot n^{3.28}$
NMG	0.03	8	0.27	8	2.86	8	25.27	8	$1.93 \times 10^{-5} \cdot n^{3.38}$

Table 5.5: Solution of the linear system in the EB3 simulations; CPU-time (in seconds) measured on HP and number of iterations. 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
	CPU	# it.	CPU	# it.	CPU	# it.	CPU	# it.	CPU	# it.	
J	8.27	5019	191.89	19579	3951.87	77668					$1.14 \times 10^{-5} \cdot n^{4.53}$
GS	4.22	2503	107.39	9797	2134.94	38847					$5.14 \times 10^{-7} \cdot n^{4.57}$
CG	0.40	500	4.73	1085	56.87	2245	1181.65	9046			$2.26 \times 10^{-7} \cdot n^{4.02}$
RPCG	0.05	20	0.37	33	4.13	62	34.56	120	297.38	242	$1.50 \times 10^{-6} \cdot n^{3.05}$
FPCG	0.11	29	0.42	26	2.19	29	10.22	29	46.30	30	$4.76 \times 10^{-5} \cdot n^{2.21}$

Table 5.6: Solution of the linear system in the DP2 simulations; CPU-time (in seconds) measured on HP and number of iterations. 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
	CPU	# it.	CPU	# it.	CPU	# it.	CPU	# it.	CPU	# it.	
BG	0.00	$\sim$	0.01	$\sim$	0.06	$\sim$	0.40	$\sim$			$7.67 \times 10^{-7} \cdot n^{2.71}$
J	0.16	358.94	1.30	805.81	11.43	1546.22	129.81	2531.72			$9.31 \times 10^{-6} \cdot n^{3.38}$
GS	0.09	180.88	0.71	404.38	5.76	774.83	70.40	1267.51			$5.04 \times 10^{-6} \cdot n^{3.37}$
CG	0.01	22.12	0.05	33.84	0.34	45.06	3.56	67.85			$8.27 \times 10^{-7} \cdot n^{3.13}$
$MPCG$	0.01	7.25	0.02	8.09	0.11	7.88	0.49	6.57	1.87	4.89	$2.07 \times 10^{-5} \cdot n^{2.06}$

Table 5.7: Solution of the linear system in the Hc2 simulations; averaged CPU-time (in seconds) measured on HPand number of iterations at each time level. The stopping criterion $Sc2(10^{-10})$ 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
	CPU	# it.	CPU	# it.	CPU	# it.	CPU	# it.	CPU	# it.	
BG	0.00	$\sim$	0.04	$\sim$	0.60	$\sim$	8.55	$\sim$			$4.29 \times 10^{-8} \cdot n^{3.94}$
CG	0.03	109.81	0.17	203.41	1.87	384.39	18.82	729.51			$9.92 \times 10^{-7} \cdot n^{3.45}$
$MPCG$	0.01	14.75	0.05	19.88	0.31	26.59	2.60	34.62	14.52	43.95	$2.81 \times 10^{-6} \cdot n^{2.80}$
$FPCG$	0.01	7.69	0.03	6.97	0.12	6.69	0.49	6.07	2.38	5.87	$1.34 \times 10^{-5} \cdot n^{2.17}$

Table 5.8: Solution of the linear system in the Wa2 simulations; averaged CPU-time (in seconds) measured on HP and number of iterations at each time level. 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
	CPU	# it.	CPU	# it.	CPU	# it.	CPU	# it.	
BG	0.17	$\sim$	15.83	$\sim$					$2.68 \times 10^{-8} \cdot n^{7.13}$
CG	0.13	95.37	2.64	158.12	40.12	269.50			$2.36 \times 10^{-5} \cdot n^{4.10}$
$MPCG$	0.09	16.00	1.27	22.38	13.28	30.94	138.93	42.23	$6.31 \times 10^{-5} \cdot n^{3.50}$
$FPCG$	0.04	6.38	0.29	6.50	2.07	6.25	14.85	5.39	$7.15 \times 10^{-5} \cdot n^{2.93}$

Table 5.9: Solution of the linear system in the Wa3 simulations; averaged CPU-time consumption (in seconds) measured on HP and number of iterations at each time level. The stopping criterion $Sc2(10^{-8})$ 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
	CPU	# it.	CPU	# it.	CPU	# it.	CPU	# it.	CPU	# it.	
$RPCG$	0.01	14.41	0.05	19.90	0.37	31.59	3.30	51.99	21.25	81.28	$1.87 \times 10^{-6} \cdot n^{2.94}$
$FPCG$	0.01	9.46	0.04	10.61	0.18	10.94	0.71	10.31	3.17	9.50	$2.67 \times 10^{-5} \cdot n^{2.10}$
NMG	0.06	7.92	0.10	7.52	0.28	6.75	1.21	5.89	4.78	5.26	$5.49 \times 10^{-5} \cdot n^{2.05}$

Table 5.10: Solution of the linear system in the Tf2 simulations; averaged CPU-time (in seconds) measured on HP and number of iterations at each time level. The stopping criterion $Sc2(10^{-10})$ is used by the iterative methods.

5.3 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 thus 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. test problems PE3 and EB3. 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	0.50	9.24	204.59	1.49e-6·$n^{4.48}$
System size	$7\times7\times7$	$15\times15\times15$	$31\times31\times31$	$63\times63\times63$	CPU model
`CGmilu`	0.02	0.34	6.15	141.81	3.68e-6·$n^{4.20}$
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	0.37	5.22	82.05	4.06e-6·$n^{4.02}$
System size	$7\times7\times7$	$15\times15\times15$	$31\times31\times31$	$63\times63\times63$	CPU model
`CGmilu`	0.01	0.22	2.74	47.95	7.99e-6·$n^{3.75}$
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	0.60	11.78	288.57	1.26e-6·$n^{4.61}$
System size	$7\times7\times7$	$15\times15\times15$	$31\times31\times31$	$63\times63\times63$	CPU model
`CGmilu`	0.02	0.39	8.15	197.36	2.97e-6·$n^{4.34}$
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	0.35	5.23	81.05	3.55e-6·$n^{4.06}$
System size	$7\times7\times7$	$15\times15\times15$	$31\times31\times31$	$63\times63\times63$	CPU model
`CGmilu`	0.01	0.20	2.66	47.16	6.31e-6·$n^{3.81}$

Table 5.11: Comparison between the Diffpack simulator and the specially coded C program `CGmilu` in PE3 and EB3 simulations. The CPU-time consumptions (in seconds) are measured on `SGI`.

6 Construction of the linear systems of equations

In this section, we first analyze the process of constructing the linear system associated with FEM discretization in Diffpack. Then we point out to the reader some important

implementation issues that can improve the performance of the Diffpack simulator.

6.1 The process

In general the CPU-time consumption is proportional to the number of elements and thus proportional to the total number of unknowns. During the construction of the linear system in Diffpack, CPU-time is spent on the storage allocation of **A** plus the following assembly process, 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.

Many different matrix formats are supported by Diffpack. Two formats, banded matrix and sparse matrix in particular, are involved in the simulations of the current report. Arising from uniform grids, **A** has a banded-structure where the bandwidth is typically n in 2D and n^2 in 3D, respectively. But for each row of **A** there are only a small number (independent of n) of nonzeros. This means that a banded matrix format may waste too much storage on zero entries. However, the conventional Gauss elimination requires such a storage structure because of the later "fill-in". For the iterative methods, on the other hand, matrix-vector products become the main operation. To save the storage and the computational cost, a compressed sparse row storage strategy which only stores nonzero entries is preferable. In this case, the determination of the structure of a sparse matrix (locations of the nonzeros) requires more work and a relatively complicated indexing procedure is needed to access each entry of **A**. Steps 1 and 2 in the assembly process are independent of the matrix format of **A**. However the sparse matrix format requires more CPU-time spent on the storage allocation and Step 3 in the assembly process. This can be best illustrated by the comparison between CPU-time consumptions associated with the banded matrix format and the sparse matrix format, respectively.

Grid	17×17	33×33	65×65	129×129
`MatBand`	0.01	0.03	0.12	0.52
`MatSparse`	0.01	0.05	0.18	0.75

Table 6.1: The CPU-time (in seconds) spent on the `makeSystem` function in the Pe2 simulations; different matrix formats result in different CPU consumptions.

Remark. For dynamic problems such as Hc2 and Wa2, the CPU-time spent on storage allocation is only necessary at the first time level and should be avoided for later time levels.

The application of efficient solution methods raises the request that the CPU-time spent on the construction of the linear system be reduced still further. And this is achievable under certain circumstances. More specifically, the performance may be improved 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.

6.2 The mapped Laplace equation in the Wa3 problem

The special transformation (2.7) maps the physical solution domain $\overline{\Omega}(t)$ (2.6) to a stationary computational domain Ω. It thus introduces a new elliptic boundary value

problem. By investigation of the definition of $\boldsymbol{K}$ (2.8), it reveals a relation between element matrices associated with those elements that only differ by z-coordinates. More precisely, if we give a tripled index (I, J, K) to each element matrix $\boldsymbol{A}^{I,J,K} = (A_{i,j}^{I,J,K})$, then we have

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

Suppose 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 $A_{i,j}^{I,J,K_1}$ and $A_{i,j}^{I,J,K_2}$ consists of two parts which only depend on $(K_1 - K_2)\Delta z$; one linearly varying part and one quadratically varying part. More precisely,

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

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 simplifies the construction of the linear system since a 3D assembly process is almost replaced with a 2D assembly process. In this way, major efficiency improvement can be gained.

`makeSystem`	$9 \times 9 \times 9$	$17 \times 17 \times 17$	$33 \times 33 \times 33$	$65 \times 65 \times 65$
standard	0.17	1.35	11.23	89.93
special	0.02	0.19	1.55	12.58

Table 6.2: The performance difference between the standard `FEM::makeSystem` and a special `makeSystem` function for test problem WA3; CPU-time consumptions are measured on `HP`.

6.3 Parabolic problems

Let us consider a general parabolic problem of 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.

Suppose 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 of 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

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

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.

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 is 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 $\mathbf{A}_{\text{mod}}$.

For a Diffpack simulator treating parabolic problems, it is 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.

7 Concluding remarks

In this report we have collected concrete CPU-time measurements and established performance models for a series of Diffpack PDE simulations. During this work analysis of Diffpack computational performance is carried out based on the measurements. The CPU-time measurements prove to be valuable and help to improve several parts of the old Diffpack library considerably.

time-dependency			CPU on `makeSystem`	
$\mathcal{L}$	f	u_D	first call	later calls
NO	NO	NO	3.88	0.11
NO	NO	YES	4.37	0.18
NO	YES	NO	3.89	1.48
NO	YES	YES	4.36	1.56
YES	NO	Y/N	3.86	2.31
YES	YES	Y/N	3.88	3.32

Table 6.3: Analysis of the specially implemented `makeSystem` function in a heat conduction simulator class; the CPU-times (in seconds) are measured on `HP` for simulations on a uniform 257×257 grid.

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

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

[4] J. B. Bell and G. R. Shubin, *Higher-order Godunov methods for reducing numerical dispersion in reservoir simulation*, SPE Reservoir Simulation Symposium, Dallas, Taxes, Feb. 10-13, 1985, SPE 13514, pp. 179-186.

[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] P. Colella, *Multidimensional upwind methods for hyperbolic conservation laws*, J. Comp. Phys. 87, pp. 171-200, 1990.

[11] 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.

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

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

[14] 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.

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

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

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

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

[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] 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.

[22] J. Saltzman, *An unsplit 3D upwind method for hyperbolic conservation laws*, J. Comp. Phys. 115, pp. 153-168, 1994.

[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.