PARALLEL SOLVERS FOR THE TRANSIENT MULTIGROUP NEUTRON DIFFUSION EQUATIONS

R. SCHEICHL*

Abstract

For the safety and the control of a nuclear power plant it is necessary to simulate the constituent processes on a computer system. The three-dimensional multigroup neutron diffusion equations are commonly used to describe the nuclear fission in the reactor core. They form a complicated system of coupled parabolic partial differential equations (PDEs) whose solution can involve very intensive computing. In this paper this system of PDEs is discretised using a special cell-centred mixed finite volume method (NEM-M0) in space, and a method that combines Crank-Nicholson and the BDF(2)-method in time. The linear equation systems which arise are solved with Multi-Grid as well as with preconditioned BiCGStab. The kernel of both solution methods is an effective Block-SOR method that makes use of the particular structure of the linear equation systems. The parallelisation strategy is based on a grid partitioning that distributes the data and the work homogeneously on the processors. Finally the program was tested for three typical reactor simulation problems on grids with differing coarseness. The speedup achieved by parallelising Multi-Grid and preconditioned Bi-CGStab was outstanding for all examples; even superlinear in some cases. Moreover, the parallel execution times were better than the parallel execution times of other established reactor simulation codes.

KEY WORDS: transient multigroup neutron diffusion equations; mixed finite volume discretisation; non-symmetric matrices; parallel Multi-Grid; parallel Bi-CGStab

1 Introduction

The centre point of concern in the simulation of a nuclear power plant is the reactor core. Since it is the source of the energy that is produced in the reactor, a very accurate model of the constituent processes is needed. The nuclear fission in the reactor core can be best described with a neutron balance equation. As neutrons are both the initiator and the result of nuclear fissions, they are a good local measure of the number of fissions taking place. This three-dimensional problem is modelled by a complicated system of coupled partial differential equations, the multigroup

*Department of Mathematical Sciences, University of Bath, Bath BA2 7AY, UK
e-mail: R.Scheichl@maths.bath.ac.uk

neutron diffusion equations, whose solution can involve very intensive computing. Simplification of the mathematical model, like the transition to a two-dimensional model, is one way of reducing the work, but it is not always sufficient. Therefore it is necessary to find effective algorithms for the solution of the three-dimensional model. The continuing progress in the area of multiprocessor technology suggests that we should consider the application of parallel algorithms to this problem.

Numerous parallel core analysis methods have been reported in the literature (Erbacci *et al.*,[7] Finnemann *et al.*,[9] Hammer *et al.*,[13] Joo *et al.*,[14] Sarsour *et al.*,[20] Uematsu *et al.*,[24] Vaudescal,[26] Kim & Cho[27]) all of which use the multigroup neutron diffusion equations as their mathematical model. In this paper we consider the same model, and in particular we investigate the efficient parallel performance of Multi-Grid and Block-SOR-preconditioned Bi-CGStab (P-Bi-CGStab) for this problem. In comparison to the above references our methods achieve optimal (linear) speedup on all but the coarsest grids, for all tested examples, as we shall explain more precisely below.

The equations are discretised using a special mixed finite volume method (NEM-M0) in space, and a method that combines Crank-Nicholson and the BDF(2)-method in time. The parallel methods are based on a simple grid partitioning, and were implemented on a shared memory 12-processor machine (SGI Power Challenge). We tested the methods for three typical reactor simulation problems (including a benchmark problem[10]) on grids with differing coarseness.* Comparison of the speedup results with the results reported in the literature is difficult. Some of the references only treat the steady state case[20,24,27] or the two-dimensional problem,[9] while others use methods that are not competitive in the sequential case.[7] However, the results can be easily compared to the results in Hammer *et al.*,[13] since the latter results were obtained on the exact same computer for the same benchmark problem. The code PANBOX[3] of the Siemens AG Operating Division KWU, that is parallelised in Hammer *et al.*[13] is one of the most efficient for the sequential case to date. However, on the coarsest grid which is currently used for most practical simulations ($\sim$ 4000 nodes) it is only possible for them to achieve a speedup factor of 3.7 on 8 processors. Using P-Bi-CGStab, we are on the other hand able to obtain an optimal speedup factor of 8.0 on 8 processors, and the speedup curves show a good performance even for higher numbers of processors. Moreover, this is of practical interest: The far better speedup of P-Bi-CGStab means that its parallel execution time is better than the parallel execution time of the established reactor simulation code PANBOX. Even if we go to finer grids our method shows superior speedup behaviour. The speedup factor of PANBOX on a grid with about 16000 nodes is still sub-optimal (7.3 on 8 processors). On the same size of grid ($\sim$ 21000 nodes), Joo *et al.*[14] also present results for the same benchmark problem using a different heuristic spatial discretisation and parallel Block-ILU-preconditioned Bi-CGStab, but they only obtain a speedup of 4.4 and 8.0 on 6 and 12 processors respectively. The speedup of the methods presented in this paper (Block-SOR-preconditioned

*Because of the difficult geometrical structure of a reactor core it is presently not possible to run Multi-Grid on the coarsest grid

Bi-CGStab and Multi-Grid) on the other hand is again optimal (linear) on the next finest grid ($\sim$ 32000 nodes).

The outline of the paper is as follows. The mathematical model and its discretisation are described in Section 2. The central parts of the paper are the construction of the iterative solvers in Section 3, their parallelisation on a shared memory machine in Section 4, and the numerical results in Section 5. Finally, we will draw some conclusions in Section 6.

2 The Multigroup Neutron Diffusion Equations and their Discretisation

2.1 Description of the Mathematical Model

Analogously to the approach of Boltzmann for the description of gas dynamics, we use a transport equation in phase space to describe accurately the conservation of the number of neutrons in a nuclear reactor. The fundamental quantities in the transport equation are the neutron flux $\Phi(E, \mathbf{\Omega}, \mathbf{x}, t)$ and the neutron current $\mathbf{j}(E, \mathbf{\Omega}, \mathbf{x}, t)$, where E, $\mathbf{\Omega}$, $\mathbf{x}$, and t denote energy, direction of motion, space, and time respectively. However, for the numerical treatment of realistic three-dimensional problems this transport equation is too complicated, and a standard simplification neglects interactions between the neutrons and assumes that the scattering of a neutron after the collision with a nucleus is almost isotropic. These assumptions enable us to eliminate non-linearities and the dependency on the direction of motion. As a further simplification the phase space is discretised into a small number of energy groups $g = 1...G$, leaving us with a time and space dependent system of partial differential equations (PDEs), the multigroup neutron diffusion equations:

$$\left.\begin{array}{c} v_g^{-1}\dfrac{\partial}{\partial t}\,\phi_g(\mathbf{x},t) \;+\; \nabla\cdot\mathbf{j}_g(\mathbf{x},t) \;+\; \Sigma_{tg}\,\phi_g(\mathbf{x},t) \;= \\ \displaystyle\sum_{g'=1}^{G}\left(\Sigma_{g'g} + \frac{(1-\beta)}{\lambda}\chi_{pg}\nu\Sigma_{fg'}\right)\phi_{g'}(\mathbf{x},t) + \sum_{i=1}^{I}\chi_{dg}^{i}\,\lambda_i C_i(\mathbf{x},t) \end{array}\right\}\; g = 1...G. \qquad (1)$$

The unknowns in this system are the flux density $\phi_g(\mathbf{x}, t)$ and the current density $\mathbf{j}_g(\mathbf{x}, t)$ in each energy group $g = 1...G$, as well as the precursor* concentrations $C_i(\mathbf{x}, t)$ in each precursor group $i = 1...I$. Fick's Law:

$$\mathbf{j}_g(\mathbf{x},t) \;+\; D_g\,\nabla\phi_g(\mathbf{x},t) \;=\; \mathbf{0}, \qquad g = 1...G, \qquad (2)$$

*Not all neutrons emerge immediately from the nuclear fission of a fissionable isotope. In a few cases one of the nuclei in the β-decay sequence of a fission product also emits neutrons, known as delayed neutrons. Their influence is crucial, and it is taken into account through the concentrations $C_i(\mathbf{x}, t)$ of their precursor nuclei.

describes the dependency of the flux density and the current density. The precursor concentrations $C_i(\mathbf{x},t)$ on the other hand are subject to the decay law:

$$\frac{\partial}{\partial t}C_i(\mathbf{x},t) + \lambda_i C_i(\mathbf{x},t) \;=\; \sum_{g=1}^{G} \frac{\beta_i}{\lambda}\,\nu\Sigma_{fg}\,\phi_g(\mathbf{x},t), \qquad i = 1...I. \tag{3}$$

The system parameters that play a role are: the neutron velocities v_g; the scattering cross sections $\Sigma_{g'g} := \Sigma_{g'g}(\mathbf{x},t)$, the fission cross sections $\nu\Sigma_{fg} := \nu\Sigma_{fg}(\mathbf{x},t)$, the total cross sections $\Sigma_{tg} := \Sigma_{tg}(\mathbf{x},t)$; the diffusion coefficients $D_g := D_g(\mathbf{x},t)$; the energy spectra χ_{pg} and χ^i_{dg} of the prompt and the delayed neutrons; the fractions β_i of neutrons appearing from each precursor group (where $\beta = \sum_{i=1}^{I}\beta_i$); the decay constants λ_i of each precursor group; and the multiplication constant λ that describes the criticality of the reactor.

Usually, the first step in a reactor simulation is the solution of a generalised eigenvalue problem to calculate the multiplication constant λ and the initial flux and current density. This process is called criticality calculation (see Scheichl,[21] or Schmid[22]). However, to simplify the presentation in this paper, we will assume the initial flux and current density to be given. Finally, we need to specify the boundary conditions: To guarantee the conservation of flux and current we need continuity on the interface between two fuel assemblies:

$$\lim_{\boldsymbol{\xi}\to\mathbf{x}^-} \phi_g(\boldsymbol{\xi},t) \;=\; \lim_{\boldsymbol{\xi}\to\mathbf{x}^+} \phi_g(\boldsymbol{\xi},t), \qquad \mathbf{x}\in\Gamma_I\,, \tag{4a}$$

$$\lim_{\boldsymbol{\xi}\to\mathbf{x}^-} \mathbf{j}_g(\boldsymbol{\xi},t) \;=\; \lim_{\boldsymbol{\xi}\to\mathbf{x}^+} \mathbf{j}_g(\boldsymbol{\xi},t), \qquad \mathbf{x}\in\Gamma_I\,. \tag{4b}$$

We shall assume that the reactor core is surrounded by air. Thus there is no neutron flux into the core and vacuum boundary conditions apply on the outer boundary:

$$\mathbf{j}_{g,in}(\mathbf{x},t) \;=\; \mathbf{0}, \qquad \mathbf{x}\in\Gamma_O. \tag{5}$$

For a more detailed description of the model and the underlying reactor physics see Duderstadt & Hamilton,[6] or Scheichl.[21]

2.2 Discretisation

We will now briefly describe the discretisation of System (1) – (5). For the discretisation in space the region $\Omega \subset \mathbb{R}^3$ is triangulated with a rectangular grid into evenly sized cuboids (boxes) Ω^m, $m = 1...M$. By substituting Equation (2) into Equation (1), we get a system of second-order parabolic PDEs which can be discretised using standard finite element (FEM) or finite difference (FDM) approaches. However, these approaches do not guarantee the conservation of the current density $\mathbf{j}_g(\mathbf{x},t)$. Therefore it is better to employ a mixed discretisation approach on the original System (1) – (3). In Finnemann *et al.*,[8] mixed cell-centred finite volume methods have been proposed especially for the multigroup neutron diffusion equations. They are called nodal expansion methods (NEMs), and are closely related to

Raviart–Thomas–Nédélec mixed-hybrid finite elements (see Hennart & Del Valle[15]). We will apply the lowest order NEM (NEM-M0).

Equations (1) and (3) are integrated over each box Ω^m, and divided by the volume of the box, as in the case of standard FVMs. Equations (2) on the other hand are integrated over the six faces Γ^m_{us} $(u = x, y, z;\ s = l, r)^\dagger$ of each box, and then divided by the area of the face. The normal derivative $\frac{\partial \phi_g}{\partial \mathbf{n}}(\mathbf{x}, t)$ that occurs on each face is approximated by the derivative of a second-order polynomial that interpolates $\phi_g(\mathbf{x}, t)$ on the box.

The degrees of freedom on each box Ω^m are the zeroth momenta of the flux density and of the precursor concentration, $\phi^m_g(t) := \frac{1}{|\Omega^m|}\int_{\Omega^m}\phi_g(\mathbf{x}, t)d\mathbf{x}$ and $C^m_i(t) := \frac{1}{|\Omega^m|}\int_{\Omega^m} C_i(\mathbf{x}, t)d\mathbf{x}$ respectively. Instead of using the average current density
$j^m_{gus}(t) := \frac{1}{|\Gamma^m_{us}|}\oint_{\Gamma^m_{us}} \mathbf{j}_g(\mathbf{x}, t)\cdot \mathbf{e}_u\, dA$ on each face of the box, we separate it into an outgoing $\hat{j}^m_{gus}(t)$ and an incoming current $\check{j}^m_{gus}(t)$:

$$\begin{aligned} j^m_{gur}(t) &= \hat{j}^m_{gur}(t) - \check{j}^m_{gur}(t)\,, \\ j^m_{gul}(t) &= \check{j}^m_{gul}(t) - \hat{j}^m_{gul}(t)\,. \end{aligned}$$

The semi-discrete equations can then be written in the following form

$$\begin{aligned} v_g^{-1}\frac{d\phi^m_g(t)}{dt} &= \sum_u \frac{4c^m_{gu}}{a^m_u}\Big(\overbrace{\check{j}^m_{gur}(t)}^{\text{see (7)}}+\overbrace{\check{j}^m_{gul}(t)}^{\text{see (7)}}\Big) - \left(\Sigma^m_{tg} + \sum_u \frac{2c^m_{gu}}{a^m_u}\right)\phi^m_g(t) \qquad \text{(6a)} \\ &+ \sum_{g'=1}^{G}\left(\Sigma^m_{g'g} + \frac{(1-\beta)}{\lambda}\chi_{pg}\nu\Sigma^m_{fg'}\right)\phi^m_{g'}(t) + \sum_{i=1}^{I}\chi^i_{dg}\lambda_i C^m_i(t)\,, \end{aligned}$$

$$\begin{aligned} \hat{j}^m_{gur}(t) &= c^m_{gu}\phi^m_g(t) + \underline{c}^m_{gu}\,\overbrace{\check{j}^m_{gur}(t)}^{\text{see (7)}} + \bar{c}^m_{gu}\,\overbrace{\check{j}^m_{gul}(t)}^{\text{see (7)}}\,. \qquad \text{(6b)} \\ \hat{j}^m_{gul}(t) &= c^m_{gu}\phi^m_g(t) + \bar{c}^m_{gu}\,\underbrace{\check{j}^m_{gur}(t)}_{\text{see (7)}} + \underline{c}^m_{gu}\,\underbrace{\check{j}^m_{gul}(t)}_{\text{see (7)}}\,. \qquad \text{(6c)} \end{aligned}$$

$$\frac{dC^m_i(t)}{dt} = -\lambda_i C^m_i(t) + \frac{\beta_i}{\lambda}\sum_{g=1}^{G}\nu\Sigma^m_{fg}\phi^m_g(t)\,. \qquad \text{(6d)}$$

with

$$c^m_{gu} := \frac{6D^m_{gu}}{12D^m_{gu}+1}\,,\quad \underline{c}^m_{gu} := 1 - 4c^m_{gu} - \bar{c}^m_{gu}\,,\quad \bar{c}^m_{gu} := \frac{-4c^m_{gu}}{12D^m_{gu}+3}\,,$$

and a^m_u denoting the length of box Ω^m in direction $u = x, y, z$, respectively.

Using the interface condition (4b) and the boundary condition (5), we can completely omit the incoming currents $\check{j}^m_{gus}(t)$ in Equations (6a–6d). Let Ω^{m^-} and Ω^{m^+} denote the left and right neighbours of box Ω^m in direction u. Then

$$\check{j}^m_{gul}(t) = \begin{cases} 0 & \text{if } \Gamma^m_{ul} \subset \Gamma_O \\ \hat{j}^{m^-}_{gur}(t) & \text{otherwise} \end{cases} \qquad \check{j}^m_{gur}(t) = \begin{cases} 0 & \text{if } \Gamma^m_{ur} \subset \Gamma_O \\ \hat{j}^{m^+}_{gul}(t) & \text{otherwise} \end{cases} \qquad (7)$$

†s ... side, l ... left, r ... right

The second step is to integrate this System (6a) – (6d) of linear ordinary differential equations (ODEs) in time. The ODE system is stiff, and it is necessary to use stable methods for its integration (see Hairer & Wanner[12]). The standard criteria for the stability of an integration scheme are A–stability and L–stability. L–stability is stronger and implies A–stability. Implicit backward–Euler is a first-order L–stable method. The second–order A–stable multi–step method that has the smallest truncation error is Crank–Nicholson (or the trapezoidal rule (TR)). Unfortunately it is not L–stable. A composition of Crank-Nicholson with the second–order backward differentiation formula (TR-BDF(2)) was proposed by Bank *et al.*[1] It combines the desirable properties of both methods. It preserves the L–stability of BDF(2), but it has a smaller truncation error. Moreover, it enables us to estimate the truncation error and the optimal time step adaptively. The simple idea is to divide each time step into two partial steps, and integrate the first step with Crank–Nicholson and the second with BDF(2). For a more detailed definition and analysis of TR-BDF(2) see Bank *et al.*,[1] or Scheichl.[21] Because of the second Dahlquist barrier there are no A–stable multi–step methods of order greater than two.

The application of all the mentioned methods to (6a) – (6d) is similar, and the resulting linear equation systems have all got the same structure. For simplicity we will only demonstrate it briefly for implicit backward–Euler. Let t_n, $n = 1...N$, denote the discrete time points, and $h_n := t_n - t_{n-1}$ the time steps. Furthermore let $\phi_g^{m,n}$, $\hat{j}_{gur}^{m,n}$, and $C_i^{m,n}$ denote the discrete solution values at time t_n, $n = 1...N$, respectively. The integration of Equation (6d) leads to the following explicit equation for $C_i^{m,n}$:

$$C_i^{m,n} = \frac{1}{1 + h_n\lambda_i}\Big(\sum_{g=1}^{G}\frac{\beta_i}{\lambda}\,\nu\Sigma_{fg}^{m,n}\phi_g^{m,n} + C_i^{m,n-1}\Big) \tag{8}$$

Then we integrate Equation (6a) and substitute $C_i^{m,n}$ from (8) to obtain

$$\begin{aligned}\left[v_g^{-1} + h_n\Big(\Sigma_{tg}^{m,n} + \sum_u \frac{2c_{gu}^{m,n}}{a_u^m}\Big)\right]\phi_g^{m,n} - h_n\sum_{g'=1}^{G}\Big(\Sigma_{g'g}^{m,n} + \bar{\chi}_g^n\nu\Sigma_{fg'}^{m,n}\Big)\phi_{g'}^{m,n} - \\ h_n\sum_u\frac{4c_{gu}^{m,n}}{a_u^m}\Big(\check{j}_{gur}^{m,n} + \check{j}_{gul}^{m,n}\Big) \;=\; v_g^{-1}\phi_g^{m,n-1} + \sum_{i=1}^{I}\chi_{dg}^i\frac{h_n\lambda_i}{1+h_n\lambda_i}C_i^{m,n-1}\end{aligned} \tag{9}$$

with $\bar{\chi}_g^n := \chi_{pg}(1-\beta) + \sum_i \chi_{dg}^i\frac{\lambda_i}{1+h_n\lambda_i}\beta_i$. These Equations (9) are coupled with Equations (6b–6c) at time t_n. The incoming currents $\check{j}_{gus}^{m,n}$ in (6b–6c) and (9) can again be substituted using (7). Therefore the main work in each time step is the solution of a $(7*G*M)$-dimensional non symmetric linear equation system for $\phi_g^{m,n}$ and $\hat{j}_{gus}^{m,n}$. The sparsity structure of the system is presented in Figure 1.

$$A \cdot \mathbf{u} := \left(\begin{array}{ccc} \underbrace{\begin{array}{cc|ccccc} * & * & & & 0 & & \\ * & * & & & & & \\ \hline * & 0 & 1 & 0 & \cdots & \cdots & 0 \\ 0 & * & 0 & 1 & \ddots & & \vdots \\ \vdots & & \vdots & \ddots & \ddots & \ddots & \vdots \\ * & 0 & \vdots & & \ddots & 1 & 0 \\ 0 & * & 0 & \cdots & \cdots & 0 & 1 \end{array}}_{A_{11}} & & \begin{array}{l}\text{Coupling via}\\ \text{the outgoing}\\ \text{currents of}\\ \text{adjacent boxes}\end{array} \\ & \bullet\;\bullet\;\bullet & \\ \begin{array}{l}\text{Coupling via}\\ \text{the outgoing}\\ \text{currents of}\\ \text{adjacent boxes}\end{array} & & \underbrace{\begin{array}{cc|ccccc} * & * & & & 0 & & \\ * & * & & & & & \\ \hline * & 0 & 1 & 0 & \cdots & \cdots & 0 \\ 0 & * & 0 & 1 & \ddots & & \vdots \\ \vdots & & \vdots & \ddots & \ddots & \ddots & \vdots \\ * & 0 & \vdots & & \ddots & 1 & 0 \\ 0 & * & 0 & \cdots & \cdots & 0 & 1 \end{array}}_{A_{MM}} \end{array}\right) \bullet \begin{pmatrix} \phi_1^{1,n} \\ \phi_2^{1,n} \\ \hat{j}_{1xl}^{1,n} \\ \hat{j}_{2xl}^{1,n} \\ \vdots \\ \hat{j}_{1zr}^{1,n} \\ \hat{j}_{2zr}^{1,n} \\ \bullet \\ \bullet \\ \bullet \\ \phi_1^{M,n} \\ \phi_2^{M,n} \\ \hat{j}_{1xl}^{M,n} \\ \hat{j}_{2xl}^{M,n} \\ \vdots \\ \hat{j}_{1zr}^{M,n} \\ \hat{j}_{2zr}^{M,n} \end{pmatrix} = \begin{pmatrix} * \\ * \\ 0 \\ 0 \\ \vdots \\ 0 \\ 0 \\ \bullet \\ \bullet \\ \bullet \\ * \\ * \\ 0 \\ 0 \\ \vdots \\ 0 \\ 0 \end{pmatrix} =: \mathbf{b}$$

Figure 1: Pattern of the sparse linear equation system for $G = 2$

3 Iterative Solvers for Non-Symmetric Sparse Linear Equation Systems

Let us consider large sparse non-symmetric linear equation systems (LESs)

$$A\,\mathbf{u} = \mathbf{b}, \qquad A \neq A^T, \tag{10}$$

arising from the discretisation of PDEs, as described in Section 2 (fig. 1). There are two strategies that are currently commonplace in the iterative solution of (10); preconditioned Krylov subspace methods (see Saad[19]) and Multi-Grid methods (see Hackbusch,[11] or Bramble[4]).

3.1 Preconditioned Bi-CGStab

There is a whole class of methods for the solution of (10) that can be interpreted as projection methods onto special m-dimensional subspaces $\mathbf{u}_0 + \mathcal{K}_m$, where $\mathcal{K}_m$ is the Krylov subspace

$$\mathcal{K}_m(A, \mathbf{r}_0) = \mathit{span}\left\{\mathbf{r}_0, A\mathbf{r}_0, A^2\mathbf{r}_0,\ \ldots\ , A^{m-1}\mathbf{r}_0\right\} \tag{11}$$

and $\mathbf{r}_0 := \mathbf{b} - A\mathbf{u}_0$ denotes the initial residual. These methods are called Krylov subspace methods. An extensive description of the most common Krylov subspace methods can be found in Saad.[19] The iterate $\mathbf{u}_m \in \mathcal{K}_m(A, \mathbf{r}_0)$ is chosen such that

$$\mathbf{r}_m \perp \mathcal{L}_m, \tag{12}$$

where $\mathbf{r}_m := \mathbf{b} - A\mathbf{u}_m$ is the residual and $\mathcal{L}_m$ another subspace of dimension m. The different versions of the Krylov subspace methods arise from different choices of the subspace $\mathcal{L}_m$. A very effective and robustly converging method for non-symmetric problems is Bi-CGStab by Van der Vorst,[25] as presented in Figure 2.

Given: initial guess $\mathbf{u}_0$, Preconditioner P_L, $\tilde{\mathbf{r}}_0 = P_L^{-1}(\underbrace{\mathbf{b} - A\,\mathbf{u}_0}_{\mathbf{r}_0})$,
$\bar{\mathbf{r}}_0$ arbitrary, with $\langle \tilde{\mathbf{r}}_0, \bar{\mathbf{r}}_0 \rangle \neq 0$ (e.g. $\bar{\mathbf{r}}_0 = \tilde{\mathbf{r}}_0$)

$\tilde{\mathbf{v}}_0 = \tilde{\mathbf{p}}_0 = \mathbf{0}$;
$\tilde{\rho}_0 = \alpha = \tilde{\omega}_0 = 1$;
<u>for</u> $i = 1, 2, \ldots$ <u>until</u> convergence <u>do</u>

$\tilde{\rho}_i = \langle\, \tilde{\mathbf{r}}_{i-1}, \bar{\mathbf{r}}_0 \,\rangle$;
$\beta = (\tilde{\rho}_i \,/\, \tilde{\rho}_{i-1}) \cdot (\alpha \,/\, \tilde{\omega}_{i-1})$;
$\tilde{\mathbf{p}}_i = \tilde{\mathbf{r}}_{i-1} + \beta\,(\tilde{\mathbf{p}}_{i-1} - \tilde{\omega}_{i-1}\,\tilde{\mathbf{v}}_{i-1})$;
$\tilde{\mathbf{v}}_i = P_L^{-1} A\,\tilde{\mathbf{p}}_i$;
$\alpha = \tilde{\rho}_i \,/\, \langle\, \tilde{\mathbf{v}}_i, \bar{\mathbf{r}}_0 \,\rangle$;
$\mathbf{s} = \tilde{\mathbf{r}}_{i-1} - \alpha\,\tilde{\mathbf{v}}_i$;
$\mathbf{t} = P_L^{-1} A\,\mathbf{s}$;
$\tilde{\omega}_i = \langle\, \mathbf{t}, \mathbf{s} \,\rangle \,/\, \langle\, \mathbf{t}, \mathbf{t} \,\rangle$;
$\mathbf{u}_i = \mathbf{u}_{i-1} + \alpha\,\tilde{\mathbf{p}}_i + \tilde{\omega}_i\,\mathbf{s}$;
$\tilde{\mathbf{r}}_i = \mathbf{s} - \tilde{\omega}_i\,\mathbf{t}$;

<u>end</u>

Figure 2: Left-preconditioned Bi-CGStab

The rate of convergence for these Krylov subspace methods depends largely on the condition of the matrix A. Preconditioning is therefore one of the main keys to a better performance. The LES is transformed as follows:

$$A\,\mathbf{u} = \mathbf{b} \;\longmapsto\; \tilde{A}\,\tilde{\mathbf{u}} := \underbrace{(P_L A P_R)}_{\tilde{A}}\,\underbrace{(P_R^{-1}\mathbf{u})}_{\tilde{\mathbf{u}}} = \underbrace{P_L\,\mathbf{b}}_{\tilde{\mathbf{b}}} =: \tilde{\mathbf{b}}, \tag{13}$$

The preconditioning operator (or matrix) $P = P_L P_R$ in (13) is chosen in a way that the condition of the new (preconditioned) iteration matrix $\tilde{A} = P_L^{-1} A P_R^{-1}$ is significantly better than the condition of A and that the preconditioning systems $P_L\,\mathbf{w} = \mathbf{d}$ and $P_R\,\mathbf{w} = \mathbf{d}$ can be solved quickly. The iterative method, e.g. the Krylov subspace method, is then applied to the transformed (or preconditioned) LES. If P_R (resp. P_L) is chosen to be equal to the identity I, we speak of left (resp. right) preconditioning. Examples of standard preconditioning operators are Jacobi, SOR, or different forms of incomplete LU-decompositions (ILU). More recently Domain Decomposition (Smith *et al.*[23]) and Multilevel methods (Bramble *et al.*,[5] Jung & Langer[16]) have become important.

Here, we apply a special Block-SOR Preconditioner (as presented in Section 3.3) exploiting the particular structure of the LESs (Fig. 1).

3.2 Multi-Grid

Large, sparse LESs (10) arising from the discretisation of PDEs, as seen in Section 2 (Fig. 1) have an underlying geometrical grid. By applying frequency analysis to standard iterative methods, e.g. Jacobi or Gauss-Seidel, we can show that the high frequencies in the error $\mathbf{z}^{(k)} = \mathbf{u} - \mathbf{u}^{(k)}$ and in the residual (or defect) $\mathbf{r}^{(k)} = \mathbf{b} - A\,\mathbf{u}^{(k)} = A\,\mathbf{z}^{(k)}$ are reduced very well, i.e. the methods are smoothing the error and the defect.[11] The low frequencies can be represented on a coarser grid, where a smaller LES has to be solved. This is the basis of a two-grid method. Applying this strategy recursively on a series of grids (levels $q = l, l-1, ..., 1$) with a decreasing number of grid points leads to the Multi-Grid method (MGM, Fig. 3). A typical

<u>main</u> **MGM**	
⋮	
$\mathbf{u} := \mathbf{u}^0$;	Initial solution
<u>do</u> $\mathbf{MGC}(l, \mathbf{u}, \mathbf{b}, A)$ <u>until</u> converged	MG-iteration, convergence test
<u>procedure</u> $\mathbf{MGC}(q, \mathbf{v}_q, \mathbf{r}_q, A_q)$	
<u>if</u> $q = 1$ <u>then</u> $\mathbf{v}_q \overset{\approx}{=} A_q^{-1}\,\mathbf{r}_q$;	Coarse-grid solver
<u>else</u>	
<u>for</u> $j = 1$ <u>to</u> ν_q^{pre} <u>do</u>	
$\mathbf{v}_q := S_q^{pre}(\mathbf{v}_q, \mathbf{r}_q, A_q)$;	Pre-smoothing
$\mathbf{r}_{q-1} := I_q^{q-1}\,(\mathbf{r}_q - A_q\,\mathbf{v}_q)$;	Residual, Restriction
$\mathbf{w}_{q-1} := \mathbf{0}$	
<u>for</u> $k = 1$ <u>to</u> γ_{q-1} <u>do</u>	MG-cycles, $\gamma_{q-1} = 1, 2, ...$
$\mathbf{MGC}(q-1, \mathbf{w}_{q-1}, \mathbf{r}_{q-1}, A_{q-1})$;	Coarse-grid correction
$\mathbf{v}_q := \mathbf{v}_q + I_{q-1}^q\,\mathbf{w}_{q-1}$;	Prolongation, Correction
<u>for</u> $j = 1$ <u>to</u> ν_q^{post} <u>do</u>	
$\mathbf{v}_q := S_q^{post}(\mathbf{v}_q, \mathbf{r}_q, A_q)$;	Post-smoothing

Figure 3: Multi-Grid method (MGM)

coarse grid (level $q = 1$) for the triangulation of a reactor core is formed by the fuel assemblies as shown in Figure 4 (right) and consists of 18 axial layers. The finer grids are obtained by uniform refinement in each coordinate direction (Fig. 4, left). Multi-Grid (as per Fig. 3) is now used to solve the LES on the finest grid

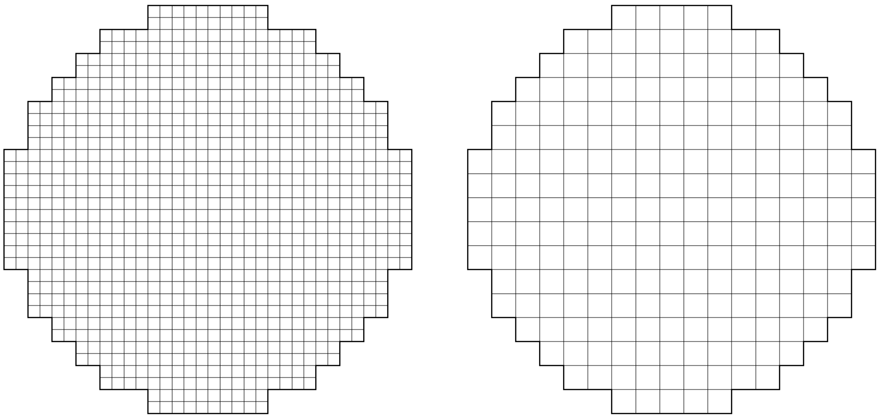

Figure 4: Axial cut of fine and coarse grid for the reactor problem

(level l). However, it has a lot of components and parameters which have to be selected. We will therefore briefly describe the components used for the solution of our LESs (Fig. 1).

Smoother: For the pre- and post-smoothing, S_q^{pre} and S_q^{post}, we use the Block-SOR method which has already been mentioned in connection with Bi-CGStab. It is described in Section 3.3. One Block-SOR iteration does not have a satisfactory smoothing effect, therefore ν_q^{pre} and ν_q^{post} are chosen between 2 and 4.

Coarse-grid correction: The correction $\mathbf{w}_{q-1}$ on level $q-1$ is obtained from γ_{q-1} multi-grid cycles applied to the defect system

$$A_{q-1}\,\mathbf{w}_{q-1} = \mathbf{r}_{q-1}, \qquad \text{for all} \quad q = l, ..., 2. \tag{14}$$

A choice of $\gamma_{q-1} = 1$ (resp. 2), for all $q = l, ..., 2$ gives the well known V–cycle (resp. W–cycle). For the numerical results in Section 5 we apply the W-cycle, and the matrices A_{q-1} are obtained by discretisation of the PDEs on level $q-1$. On the coarsest grid, i.e. level $q = 1$, the LESs of all the practical numerical tests are still too large to be solved directly. Therefore, we solve them iteratively with preconditioned Bi-CGStab, as presented in Figure 2.

Restriction and Prolongation: The vector $\mathbf{u}$ in our LES (see Fig. 1) consists of flux components $\phi_g^{m,n}$ and current components $\hat{j}_{gus}^{m,n}$ which are treated differently in the prolongation I_{q-1}^q. The former is the zeroth momentum of the flux ϕ_g across box m. Therefore we use piecewise constant interpolation. The latter is the zeroth momentum of the current $\mathbf{j}_g$ across the surface Γ_{us}^m, and we use linear interpolation in the direction perpendicular to Γ_{us}^m. For the restriction I_q^{q-1} we use the Galerkin approach

$$I_q^{q-1} := \sigma\,(I_{q-1}^q)^T \tag{15}$$

with σ appropriately chosen.

3.3 Block-SOR Preconditioner / Smoother

The kernel of the two solution methods that we described above is an effective Block-SOR method. It is used for the preconditioning of Bi-CGStab and for the smoothing in the Multi-Grid algorithm. Given a partitioning

$$A := \begin{pmatrix} A_{11} & \cdots & A_{1M} \\ \vdots & \ddots & \vdots \\ A_{M1} & \cdots & A_{MM} \end{pmatrix}, \ \mathbf{u} := \begin{pmatrix} \mathbf{u}_1 \\ \vdots \\ \mathbf{u}_M \end{pmatrix}, \ \mathbf{b} := \begin{pmatrix} \mathbf{b}_1 \\ \vdots \\ \mathbf{b}_M \end{pmatrix} \tag{16}$$

of the matrix and the vectors in 10, Block-SOR can be implemented as shown in Figure 5. If the second for-loop in Figure 5 is run in reverse order we obtain backward

Given: $\mathbf{u}^{(0)}, \omega$
<u>for</u> $k = 0, 1, 2, ...$ <u>do</u>
    <u>for</u> $i = 1$ <u>to</u> M <u>do</u>

$$\mathbf{u}_i^{(k+1)} := \mathbf{u}_i^{(k)} + \omega A_{ii}^{-1} \left(\mathbf{b}_i - \sum_{j=1}^{i-1} A_{ij} \mathbf{u}_j^{(k+1)} - \sum_{j=i}^{M} A_{ij} \mathbf{u}_j^{(k)} \right);$$

Figure 5: Block-SOR method

Block-SOR.

The LESs, arising from the NEM-M0 discretization of the neutron diffusion equations as described in Section 2, lead to a natural partitioning of the matrix A of the system (see Fig. 1). The inversion of the diagonal blocks

$$A_{ii} := \left(\begin{array}{cc|cccccc} * & * & & & 0 & & & \\ * & * & & & & & & \\ \hline * & 0 & 1 & 0 & \cdots & \cdots & \cdots & 0 \\ 0 & * & 0 & 1 & \ddots & & & \vdots \\ \vdots & & \vdots & \ddots & \ddots & \ddots & & \vdots \\ * & 0 & \vdots & & & \ddots & 1 & 0 \\ 0 & * & 0 & \cdots & \cdots & & 0 & 1 \end{array} \right), \qquad i = 1, ..., M \tag{17}$$

is easy and can be done directly, making use of the particular structure of A_{ii}. The relaxation parameter ω is chosen between 1 and 1.2, depending on the method that it is combined with, and depending on the problem size.

4 Parallelisation

4.1 Parallelisation of iterative solvers by grid partitioning

The first step in the direct parallelisation of methods for the solution of LESs arising in the discretisation process of partial differential equations (PDEs) is the partitioning of the grid. The physical domain Ω has to be decomposed into sub-domains Ω_i, $i = 1, .., D$, e.g. like in Figure 6. The sub-domain boundaries have to coincide with

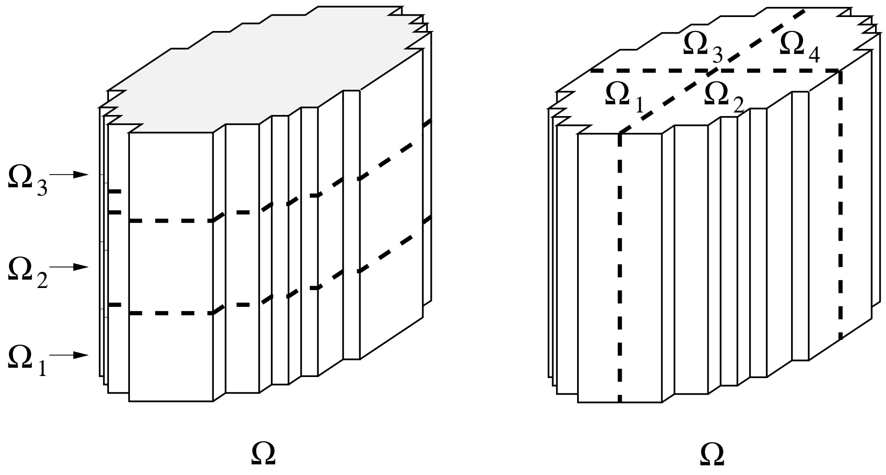

Figure 6: Two possible ways of partitioning Ω

the grid-element boundaries.

The partitioning of the data, i.e the matrices and the vectors, is a direct consequence of the grid partitioning. The advantage that the NEM-M0 discretisation has over other mixed discretisations is that each component of a vector $\mathbf{u}$ belongs to only one box m and therefore to one subdomain Ω_i. By appropriate numbering of the boxes it is possible to always obtain the following partitioning of the matrices and vectors

$$A = \begin{pmatrix} A_1 & \mathcal{A}_{12}^{off} & \cdots & \mathcal{A}_{1D}^{off} \\ \mathcal{A}_{21}^{off} & A_2 & & \mathcal{A}_{2D}^{off} \\ \vdots & & \ddots & \vdots \\ \mathcal{A}_{D1}^{off} & \mathcal{A}_{D2}^{off} & \cdots & A_D \end{pmatrix} \quad \text{and} \quad \mathbf{u} = \begin{pmatrix} \mathbf{u}_1 \\ \mathbf{u}_2 \\ \vdots \\ \mathbf{u}_D \end{pmatrix} \tag{18}$$

N_i denotes the number of local components in domain Ω_i. It follows that

$$A_i \in \mathbb{R}^{N_i \times N_i}, \quad \mathcal{A}_{ij}^{off} \in \mathbb{R}^{N_i \times N_j} \text{ and } \quad \mathbf{u}_i \in \mathbb{R}^{N_i} \qquad \text{for} \quad i, j = 1, .., D.$$

An important consequence of this partitioning is that vectors do not have to be assembled on the subdomain interfaces.

Both the Multi-Grid method (MGM, Fig. 3) and Bi-CGStab (Fig. 2) for the solution of LES (10) can be fully described in terms of vector-vector operations and matrix-vector multiplications together with a small number of scalar operations. The execution time for each component of these matrix-vector and vector-vector operations is about the same. To get a good load balance between the processes, it is therefore necessary to partition the domain evenly. Each process should have about the same number of nodes or boxes of the physical grid and therefore the same number of components of the matrices and vectors (**homogeneous parallelisation**, Bauer[2]).

The whole algorithm (Fig. 3 for Multi-Grid, Fig. 2 for Bi-CGStab) is started in parallel on multiple processors (**coarse-grained parallelisation**, Bauer[2]). The matrix-vector multiplications and vector-vector operations are distributed on the multiple processors, each working on his part of the grid, matrix, or vector.

The third big issue in parallelisation are **data dependencies** and **communication** (or **synchronisation**). Communication points are usually the bottle neck in a parallel algorithm, and it is important to minimise these points and the amount that is sent. For the basic vector-vector operations, like initialising, copying, AXPY, or scaling, it is clear that the iterations (for each component) are independent. Dependencies arise in scalar products and matrix-vector multiplications, as well as in the application of the Block-SOR method (Fig. 5). However, with some communication between (or synchronisation of) the processes they can also be run in parallel.

The scalar product

$$\sigma = \langle \mathbf{x}, \mathbf{y} \rangle \tag{19}$$

can be parallelised by reduction. Local scalar products

$$\sigma_i := \langle \mathbf{x}_i, \mathbf{y}_i \rangle \quad \forall\, i = 1, .., D \tag{20}$$

for each subdomain Ω_i, are introduced and the global scalar product σ is calculated from

$$\sigma = \sum_{i=1}^{D} \sigma_i \tag{21}$$

Communication is needed to derive that sum. The local scalar products σ_i are sent to a “master”, the sum is evaluated there and then the result is sent back to the “slaves”.

The matrix-vector multiplication

$$\mathbf{y} = A \cdot \mathbf{x} \quad \text{or} \quad \begin{pmatrix} \mathbf{y}_1 \\ \mathbf{y}_2 \\ \vdots \\ \mathbf{y}_D \end{pmatrix} = \begin{pmatrix} \mathcal{A}_1 & \mathcal{A}_{12}^{off} & \cdots & \mathcal{A}_{1D}^{off} \\ \mathcal{A}_{21}^{off} & \mathcal{A}_2 & & \mathcal{A}_{2D}^{off} \\ \vdots & & \ddots & \vdots \\ \mathcal{A}_{D1}^{off} & \mathcal{A}_{D2}^{off} & \cdots & \mathcal{A}_D \end{pmatrix} \cdot \begin{pmatrix} \mathbf{x}_1 \\ \mathbf{x}_2 \\ \vdots \\ \mathbf{x}_D \end{pmatrix} \tag{22}$$

can be split into the following sub-tasks

$$\mathbf{y}_i = \mathcal{A}_i\, \mathbf{x}_i \;+\; \sum_{j \neq i} \mathcal{A}_{ij}^{off}\, \mathbf{x}_j \tag{23}$$

on each sub-domain Ω_i, $i = 1, ..., D$. For the evaluation of the off-diagonal products, $\mathcal{A}_{ij}^{off}\, \mathbf{x}_j$, we need communication. However, the matrices $\mathcal{A}_{ij}^{off}$ are zero almost everywhere. They have entries only in columns that relate to outgoing currents $\hat{j}_{gus}^{m,n}$ of boxes $\Omega^m \subset \Omega_j$ along the interface Γ_{ij} of Ω_i and Ω_j (see Figure 7). Therefore, communication is only needed in adjacent subdomains, and only a few components of $\mathbf{x}_j$ (i.e. the $\hat{j}_{gus}^{m,n}$ on the interface) have to be sent from the processor that works on Ω_j to the processor that works on Ω_i.

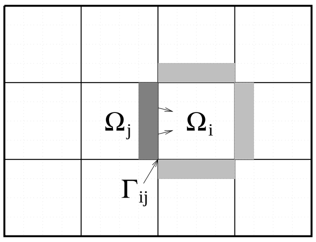

Figure 7: Boundary boxes needed for matrix-vector multiplication

The Block-SOR method The solution of each block system in the Block-SOR method always depends on the solution of the previous blocks. For a straightforward parallelisation of the Block-SOR method it would be necessary for the processors to wait for each other. It is therefore better to modify the method. We partition the system like in (18) and apply a Block-Jacobi relaxation, as presented in Figure 8. For the approximate inversion of the diagonal blocks

Given: $\mathbf{u}^{(0)}$, ω
<u>for</u> $k = 0, 1, 2, ...$ <u>do</u>
  <u>for</u> $i = 1$ <u>to</u> D <u>do</u>
    $$\mathbf{u}_i^{(k+1)} := \mathbf{u}_i^{(k)} + \omega \mathcal{A}_i^{-1} \Big(\mathbf{b}_i - \sum_{j \neq i} \mathcal{A}_{ij}^{off} \mathbf{u}_j^{(k)} \Big);$$

Figure 8: Block-Jacobi method

$\mathcal{A}_i$ (in Fig. 8) we use one iteration of the original Block-SOR method (as per Fig. 5) In that way the inner iterations $i = 1, ..., D$ are independent of each other and can be carried out in parallel. We obtain a hybrid Block-Jacobi – Block-SOR method. If $D \ll M$, it only differs in very few components from the original Block-SOR method. However, communication is once again needed prior to each iteration k of the Block-Jacobi method (Fig. 8) to update the vectors $\mathbf{u}_j$.

4.2 Implementation on a shared memory machine

The code was implemented within the IRIX 6.1 operating system on an SGI Power Challenge shared memory multiprocessor computer with 12 MIPS–R8000 processors (360 MFLOPS peak each), 1.56 GByte main memory and 4 MByte Cache per processor.

For the parallelisation we used the **m-library** (Bauer,[2] ch. 3-4). The main function for effecting the parallelisation is **m_fork**(*function,arg1,...*) which creates multiple threads of the subroutine *function(arg1,...)*. Additional routines are provided to coordinate access to shared memory, provide process synchronization, and manage the spawned slave threads.

All vectors and matrices that are needed in the solution process of (10) have to be defined as shared, e.g. in global structures in C and in COMMON blocks in Fortran. Each thread can then independently execute the local vector-vector and matrix-vector operations on the sub-domain he has been assigned to. Prior to each matrix-vector multiplication and prior to each iteration of the modified Block-SOR method a synchronisation point has to be set with **m_sync()**, replacing the communication between the processes. For the calculation of scalar products, we derive local scalar products σ_i again, and use a **critical section**

$$\mathbf{m\_lock}();$$
$$\sigma = \sigma + \sigma_i;$$
$$\mathbf{m\_unlock}();$$

to add them to the global scalar product σ. A critical section **m_lock...m_unlock** defines a block in the parallelised code in which the calling thread has sole access to the statements contained within it. Any other thread, on encountering the occupied lock block, is forced to wait until the block is cleared.

5 Numerical Results

We applied both solution methods in the simulation process of real three-dimensional reactor problems to test their robustness and efficiency. Data fot three typical problems was available. The cross sections $\Sigma_{\cdot g}$ were given in homogenised form on the grid formed by the fuel assemblies (Fig. 4, right), and the number of energy groups was taken as $G = 2$ for all three examples.

The first example is a Boron dilution case, which may take place as a consequence of a small break coolant loss accident (**Ex. 1**). The second example deals with the insertion of a control rod into a supercritical reactor (**Ex. 2**). The third example has been defined as a benchmark exercise for the verification of the reliability of numerical reactor simulations by the Reactor Physics Committee of the Nuclear Energy Agency (NEACRP).[10] The problem that is simulated is the ejection of a control assembly from the core (**Ex. 3**). For more details about these examples see Scheichl,[21] and Finnemann & Galati.[10]

We solved these problems on three grids of differing fineness. **Grid 1** is formed by the fuel assemblies (Fig. 4, right). **Grid 2** (resp. **3**) is obtained by one (resp. two) uniform refinement(s) in each coordinate direction. The discretisation of the multigroup neutron diffusion equations is carried out as described in Section 2.2, using the spatial discretisation NEM-M0 and the time discretisation TR-BDF(2) with $\gamma = 1/2$. The number of unknowns N of the LESs that arise in each time step is about 55,000 on Grid 1, about 450,000 on Grid 2, and about 3.5 million on Grid 3. Finer grids than Grid 3 were not possible because of a lack of computer memory. For the Multi-Grid method the coarsest grid was always chosen to be Grid 1. It was therefore only possible to run Multi-Grid on Grid 2 and 3.

5.1 Multi-Grid versus Preconditioned Bi-CGStab

The components and parameter for the Multi-Grid method (MG(3/2)*) and Preconditioned Bi-CGStab (P-Bi-CGStab) were chosen as described in Section 3. To check for convergence we use the defect (or residual) test

$$\|\underline{r}^{(k)}\|_2 \leq \varepsilon \, \|\underline{r}^{(0)}\|_2$$

with $\varepsilon = 10^{-6}$.

The parallel algorithms are not equivalent to the sequential ones, since the Block-SOR method that is used in both MG(3/2) and P-Bi-CGStab was modified (Sect. 4.2). The consequence of this modification are slightly different results. When different partitionings of the domain (i.e. different numbers of processes) are used, the number of iterations usually differs as well (compare the results below).

"Asymptotical" behaviour

In Table 1 we present the performance of the two methods in their sequential form

Grid	MG (3/2)	P–Bi-CGStab
1	–	20 s (43)
2	135 s (23)	390 s (86)
3	1672 s (36)	6546 s (169)
1	–	20 s (42)
2	254 s (41)	413 s (85)
3	1742 s (35)	7613 s (183)
1	–	18 s (38)
2	155 s (27)	337 s (73)
3	1818 s (40)	5631 s (148)

Table 1: Execution time and iterations (in brackets) for one time step h: Ex. 1, $h = 0.01$ s (top); Ex. 2, $h = 0.01$ s (middle); Ex. 3, $h = 0.005$ s (bottom)

for one time step† of each of the examples on the different grids.

To discuss the asymptotical behaviour of the two methods with respect to the number of unknowns N we would really need more calculations on finer (or coarser) grids. However, finer grids are not possible because of a lack of computer memory, and coarser grids are hard to introduce because of the difficult geometry of a reactor. Nevertheless, we observe that the number of iterations for MG(3/2) does not depend strongly on the number of unknowns N for a fixed accuracy $\varepsilon = 10^{-6}$. In fact, in

*The numbers in brackets stand for $\nu^{pre} = 3$ and $\nu^{post} = 2$.

†In each time step we solve **two** LESs, one for each partial step (trapezoidal and BDF(2)).

Example 2 the number of iterations even decreases when we go from Grid 2 to Grid 3, and the execution time shows the expected $O(N)$ behaviour for the number of FLOPs. The rate of convergence of MG(3/2) lies between 0.3 and 0.5. The number of iterations for P-Bi-CGStab, on the other hand, increases with a factor 2 from one grid to the next, and the rate of convergence increases from 0.5 on Grid 1 to 0.85 on Grid 3.

In comparison, MG(3/2) is about 2 to 3 times faster on Grid 2, and about 3 to 4 times faster on Grid 3. Therefore we can conclude as expected, the finer the grid becomes the more MG(3/2) outperforms P-Bi-CGStab.

Transient behaviour

To investigate how the methods perform over a whole period of a reactor simulation, and whether the parallelisation has an effect on their performance, we simulated 0.3 seconds of Example 1 on Grid 2. The initial time step was chosen to be $h = 0.01$ s, the following time steps were determined adaptively.[1,21]

The results are given in Figures 9 and 10. The modification of the Block-SOR method in the parallel case resulted in different numbers of iterations in each time step (compare Fig. 9). However the number of iterations in the parallel case is on average not higher than in the sequential case. The modification of the Block-SOR method did therefore not effect the convergence of the algorithms significantly, at least for $D \ll M$.

Both methods are converging robustly. After a short start-up period the iteration counts and the execution times are staying almost constant. As seen before, MG(3/2) is about 2.5 times faster than P-Bi-CGStab on Grid 2 (see Fig. 10). Furthermore, we also observe that this advantage carries over to the parallel case.

Speedup

The most important question is how the methods scale with the number of processors. The execution times for one time step of Example 1 on Grids 1 and 2 are given in Table 2, the speedup graphs are shown in Figure 11. The speedup

Proc.	MG (3/2) (Grid 2)	P-Bi-CGStab (Grid 2)	P-Bi-CGStab (Grid 1)
1	133.0 s (22)	363.5 s (76)	19.47 s (41)
2	68.2 s (22)	163.7 s (71)	10.20 s (45)
4	32.3 s (23)	78.0 s (71)	5.09 s (43)
6	21.1 s (22)	52.8 s (73)	3.45 s (44)
8	16.1 s (23)	38.7 s (74)	2.43 s (41)
10	12.9 s (23)	31.5 s (73)	2.07 s (43)
12	10.5 s (22)	22.9 s (69)	1.87 s (43)

Table 2: One time step ($h = 0.01$ s) of Ex. 1: Execution time and iterations

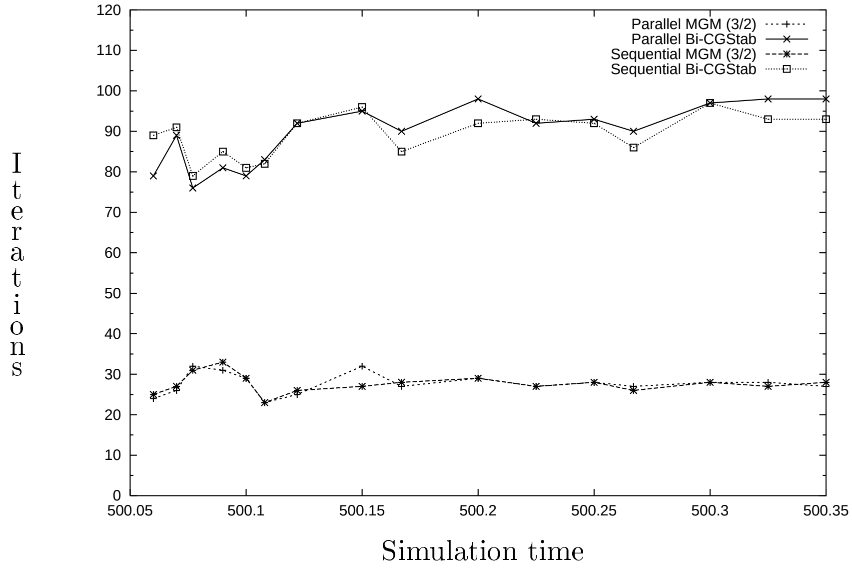

Figure 9: Number of iterations (Ex. 1, Grid 2, parallel: 9 processors)

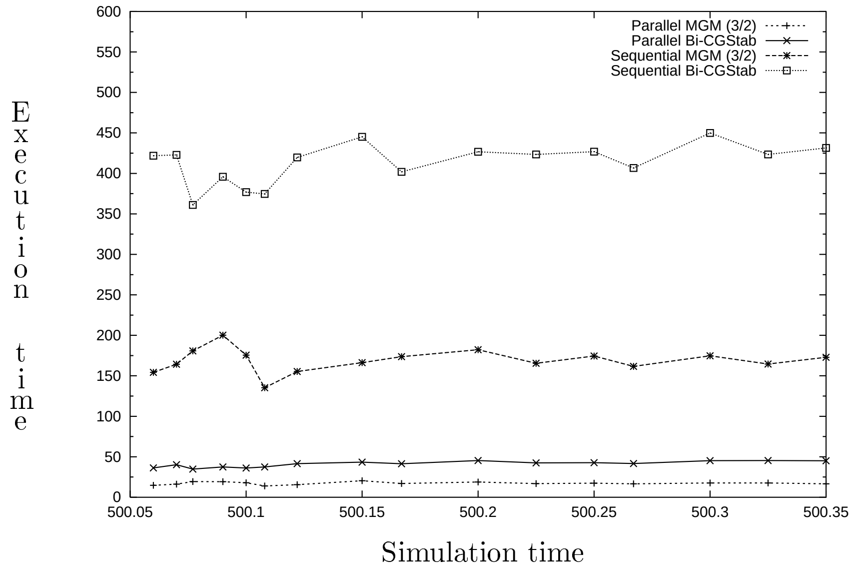

Figure 10: Execution time (in seconds) (Ex. 1, Grid 2, parallel: 9 processors)

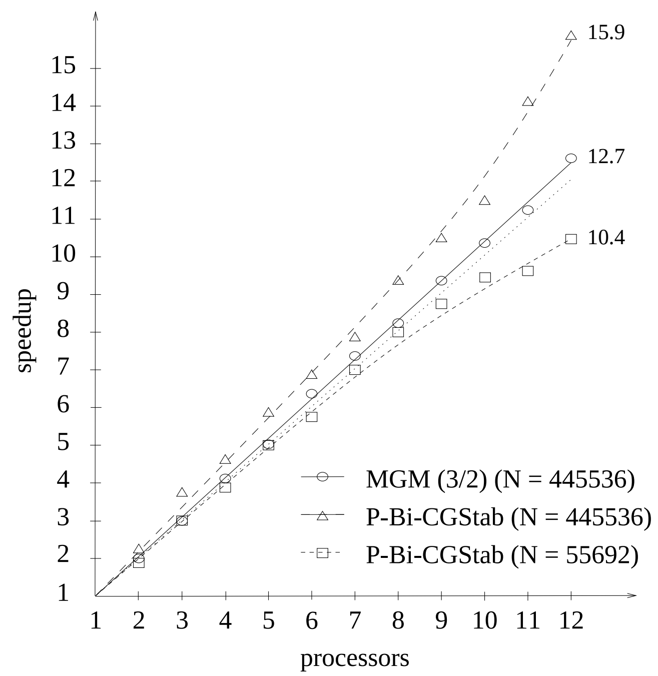

Figure 11: One time step ($h = 0.01$ s) of Ex. 1: Speedup

for P-Bi-CGStab on Grid 1 ($N \approx 55,000$) is optimal, if less than 10 processors are used. For a higher number of processors the speedup curve starts to level off slightly. On Grid 2 ($N \approx 450,000$), the speedup is even superlinear for both methods. The reasons for such unusual behaviour are a different iteration count and cache-effects. Nevertheless it is clear that the speedup for both methods is optimal on Grid 2. The performance for the other examples (2 and 3) is the same (see Tab. 3).

Proc.	MG (3/2) (Grid 2)	P-Bi-CGStab (Grid 2)	P-Bi-CGStab (Grid 1)
1	254 s (41)	413 s (85)	20 s (42)
10	24.6 s (41)	40.8 s (86)	2.1 s (41)
1	155 s (27)	337 s (73)	18 s (38)
10	14.1 s (27)	29.2 s (74)	1.8 s (35)

Table 3: Execution time and iterations (in brackets) for one time step h on Grid 2: Ex. 2, $h = 0.01$ s (top); Ex. 3, $h = 0.005$ s (bottom)

The larger the LES is, the better the speedup results are. Therefore it is not necessary to perform extensive tests on Grid 3. To show that this hypothesis is correct, we give the results for one time step of Example 1 on 10 processors, and compare them to the results given in Table 1. MG(3/2) takes 161 seconds (35 its.), that is a speedup of 10.4 compared to the sequential run. P-Bi-CGStab takes 651 seconds (162 its.), that is a speedup of 10.1. Because of the larger problem size there are no cache effects on Grid 3.

5.2 Comparison to other parallel reactor simulation codes

It is difficult to compare the speedup results with the results reported in the literature,[7,9,14,20,24,26,27] since they are all obtained on different computer systems, for different examples, and for different problem sizes. However the results can be easily compared to the results in Hammer *et al.*,[13] since the latter results were obtained on exactly the same computer for Example 1 with PANBOX[3] of the Siemens AG Operating Division KWU. This code was also available in its sequential form for tests on Grid 1, on the SGI Power Challenge. The grids that can be used in PANBOX are Grid 1, and a grid with the same axial cut but 72 instead of 18 z-layers. In Hammer *et al.*,[13] they are referred to as standard, and large respectively.

For the spatial discretisation PANBOX uses NEM-M0 and NEM-M2, a method from the same family of nodal methods that uses heuristic fourth–order polynomials to interpolate $\phi_g(\mathbf{x}, t)$ on each box (compare Sect. 2.2)[‡]. For the time discretisation it uses an heuristic exponential integration scheme related to the implicit backward-Euler. The resulting linear equation systems are solved with a multilevel method that is based on Red-Black Gauss-Seidel relaxation on the finest grid, and Coarse-Mesh Rebalancing (CMR) techniques on the coarser grids. This method is one of the most efficient methods for the sequential case to date. In Hammer *et al.*[13] the method is parallelised using threads, as described in Section 4.

The speedup of PANBOX on Grid 1 is 2.7 for 4 processors, 3.5 for 6 and 3.7 for 8 (see Hammer *et al.*,[13] Fig. 7). These results imply that even for a higher number of processors only a speedup factor of about 4 is possible. The corresponding speedup factors of P-Bi-CGStab on Grid 1 (see Fig. 11) are 3.8, 5.7 and 8.0, and we also saw that on a higher number of processors speedup factors of more than 10 are possible. The large problem,[13] cannot be compared one-to-one with the results on Grid 2 in this paper, as their problem is only half the size. The speedup factors are 4.1 for 4 processors, 5.8 for 6 and 7.3 for 8. In comparison to the speedup obtained with MG(3/2) and P-Bi-CGStab on Grid 2, as presented in Figure 11, the speedup curve of PANBOX seems to start to level off at 6 processors.

The best speedup factor for a solution method is useless if the method in the sequential case is not competitive to the method that it is compared to. Therefore we compared PANBOX (using NEM-M0) and P-Bi-CGStab on the same computer, calculating 10 fixed time steps of the control rod ejection benchmark problem (Ex. 3) on Grid 1. Because of the heuristic exponential ansatz for the time discretisation method in PANBOX, the local truncation error is smaller than for implicit backward-Euler. However, the integration method is of first order and therefore asymptotically less accurate than the second-order TR-BDF2 method. Furthermore, it does not allow an adaptive time step control as used in TR-BDF2. For a comparison of the efficiency of P-Bi-CGStab with the efficiency of the multilevel method in PANBOX

[‡]The number of DOF of NEM-M2 is the same and the method is of the same order as NEM-M0

it is therefore better to use Implicit backward-Euler (with a fixed time step) as the time discretisation. For a fairer comparison we also used the same convergence test as in PANBOX with a tolerance of $\varepsilon = 10^{-5}$.

The execution times are given in Figure 13, the speedup in Figure 12. We see that the sequential version of P-Bi-CGStab is about 2 times slower than PANBOX (Execution times: 53.5 and 115.0 seconds for PANBOX and P-Bi-CGStab, respectively). P-Bi-CGStab is unable to compete with the multilevel method in PANBOX in the serial case. Nevertheless, the far better speedup of P-Bi-CGStab makes it possible to catch up with PANBOX in the parallel case. It is remarkable, however, that this happens already at 8 processors (Execution times: 14.5 and 15 seconds for PANBOX and P-Bi-CGStab, respectively). Moreover, Figures 13 and 12 suggest that for a higher number of processors (> 10) P-Bi-CGStab will become more efficient than PANBOX.

6 Conclusions

We applied Multi-Grid and preconditioned Bi-CGStab to the non-symmetric LESs arising from the discretisation of the multigroup neutron diffusion equations in the simulation process of a nuclear reactor, and parallelised them on a shared memory multiprocessor computer. For the smoothing/preconditioning we used a hybrid Block-Jacobi - Block-SOR method. The methods converge robustly and the results compare favourably with other published results, in that we were able to achieve optimal (linear) speedup for all tested examples. Even on the coarsest grid the speedup of P-Bi-CGStab was only slightly sub-optimal. The speedup curves also suggest a good performance even for higher numbers of processors which were unfortunately not available. Moreover, this is of practical interest, as the parallel execution times of P-Bi-CGStab are better than the parallel execution times of the established reactor simulation code PANBOX of the Siemens AG Operating Division KWU. In comparison, Multi-Grid is more than twice as fast as P-Bi-CGStab (on grid 2), and scales better with the number of unknowns.

Nevertheless there are many things that could still be investigated. Because of the difficult geometry it was harder to coarsen the grid than to refine it. The introduction of coarser grids, and therefore more levels in the Multi-Grid iteration, would however be an interesting point for further investigations. It would be interesting to see if the efficiency could be further increased. In addition it would then be possible to apply MGM on the grid formed by the fuel assemblies. The application of Algebraic Multi-Grid (Kickinger,[17] or Ruge & Stüben[18]) on grid 1 would be an alternative possibility that avoids the introduction of coarser geometrical grids. Moreover, Algebraic Multi-Grid could be used in combination with MG(3/2) (as the Coarse-Grid Solver) in a hybrid Multi-Grid method for the finer grids.

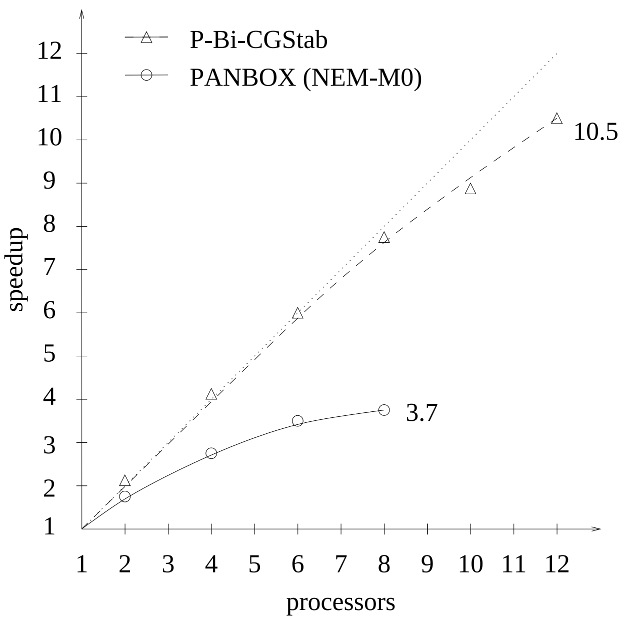

Figure 12: Speedup for 10 time steps of Ex. 3 on Grid 1

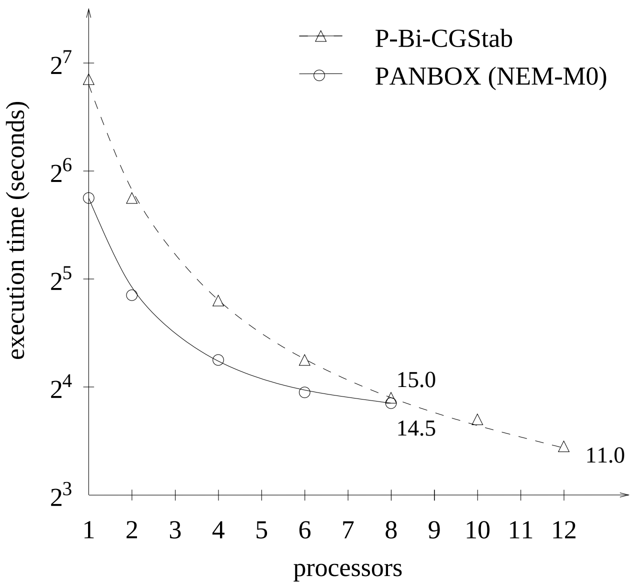

Figure 13: Execution time for 10 time steps of Ex. 3 on Grid 1

Acknowledgements

The work was sponsored by the Siemens AG, Munich, and a Leonardo da Vinci scholarship. The financial support is gratefully acknowledged. Many thanks are furthermore offered to Prof. Dr. Ulrich Langer from the University of Linz, Dr. Meinhard Paffrath from Siemens ZT, Munich, and Dr. Ivan Graham from the University of Bath for their helpful advice and comments.

References

[1] R. Bank, W. M. Coughran, W. Fichtner, E. H. Grosse, D. J. Rose, and R. K. Smith, *Transient simulation of silicon devices and circuits*, IEEE Transactions on Computer Aided Design of integrated circuits **4** (1985), no. 4, 436–451.

[2] B. E. Bauer, *Practical parallel programming*, Academic Press, San Diego, 1992.

[3] R. Böer, R. Böhm, H. Finnemann, and R. Müller, *The coupled neutronics and thermal hydraulics code system PANBOX for PWR safety analysis*, Atomkernenergie/Kerntechnik **57** (1992), no. 1, 49–54.

[4] J. H. Bramble, *Multigrid methods*, Pitman Research Notes in Mathematics, vol. 294, J. Wiley & Sons, New York, 1993.

[5] J. H. Bramble, J. E. Pasciak, and J. Xu, *Parallel multilevel preconditioners*, Mathematics of Computation **55** (1990), 1–22.

[6] J. J. Duderstadt and L. J. Hamilton, *Nuclear reactor analysis*, J. Wiley & Sons, New York, 1976.

[7] G. Erbacci, R. Ansaloni, B. Montagnini, R. Scardovelli, M. Sumini, and D. Zardini, *Porting a coarse-mesh neutron diffusion code on a CRAY T3D massively parallel computer*, Proceedings of the International Conference on High-Performance Computing and Networking, Milan 95 (1995), 318–328.

[8] H. Finnemann, F. Bennewitz, and M. R. Wagner, *Interface current techniques for multidimensional reactor calculations*, Atomkernenergie/Kerntechnik **30** (1977), 123–128.

[9] H. Finnemann, J. Brehm, E. Michel, and J. Volkert, *Mulitgrid solution of diffusion equations on distributed memory multiprocessor systems*, Atomkernenergie/Kerntechnik **52** (1988), no. 3, 169–174.

[10] H. Finnemann and A. Galati, *NEACRP 3-D LWR core transient benchmark*, Final Specifications, NEACRP-L-335, Rev. 1, 1992.

[11] W. Hackbusch, *Multi-grid methods and applications*, Springer, Berlin, 1985.

[12] E. Hairer and G. Wanner, *Solving ordinary differential equations II: Stiff and differntial-algebraic problems*, Springer, Berlin, 1991.

[13] C. Hammer, M. Paffrath, R. Böer, H. Finnemann, and C. J. Jackson, *Performance of the coupled thermalhydraulics/neutron kinetics code R/P/C on workstation clusters and multiprocessor systems*, Proceedings of the International Conference on the Physics of Reactors, Mito, Japan 96 (1996).

[14] Han G. Joo, T. J. Downar, and D. A. Barber, *Methods and performance of a parallel reactor kinetics code PARCS*, Proceedings of the International Conference on the Physics of Reactors, Mito, Japan 96 (1996), J42–J51.

[15] J. P. Hennart and E Del Valle, *On the relationship between nodal schemes and mixed-hybrid finite elements*, Numerical Methods for Partial Differential Equations **9** (1993), 411–430.

[16] M. Jung and U. Langer, *Application of multilevel methods to practical problems*, Surveys on Mathematics for Industry **1** (1991), 217–257.

[17] F. Kickinger, *Algebraic multigrid for discrete elliptic second-order problems*, Institutsbericht 513, University Linz, Institute for Mathematics, Linz, 1995.

[18] J. Ruge and K. Stüben, *Algebraic multigrid*, Multigrid methods (S. McCormick, ed.), Frontiers in Applied Mathematics, vol. 5, SIAM, Philadelphia, 1986.

[19] Y. Saad, *Iterative methods for sparse linear systems*, PWS, Boston, 1996.

[20] H. N. Sarsour, P. J. Turinsky, and C. S. Henkel, *Parallel solution of the few-group neutron diffusion equations*, Proceedings of the 5^{th} SIAM Conference on Parallel Processing for Scientific Computing (1992).

[21] R. Scheichl, *Parallel solution of the transient multigroup neutron diffusion equations with Multi-Grid and preconditioned Krylov-subspace methods*, Schriften der Johannes Kepler Universität Linz, vol. C 21, Trauner, Linz, 1997.

[22] W. Schmid, *Adaptive gemischte und nichtkonforme FE Methoden und Anwendungen auf die Mehrgruppen-Diffusionsgleichungen*, AMNS, vol. 11, Wißner, Augsburg, 1996.

[23] B. Smith, P. Bjørstad, and W. Gropp, *Domain decomposition; parallel multilevel methods for elliptic pdes*, Cambridge University Press, 1996.

[24] M. Uematsu, M. Tsuiki, and K. Kubota, *Parallel processing of boiling reactor core analysis*, Journal of Nuclear Science and Tech. **32** (1995), no. 6, 495–506.

[25] H. A. Van der Vorst, *Bi-CGSTAB: A more smoothly converging variant of Bi-CG for the solution of nonsymmetric linear systems*, SIAM Journal on Scientific and Statistical Computing **13** (1992), 631–644.

[26] J.-L. Vaudescal, *Résolution numérique des équations de la diffusion neutronique multigroupe*, Doctor thesis, University of Paris IX, Paris, 1993.

[27] Yong H. Kim and Nam Z. Cho, *Parallel solution of the neutron diffusion equation with the domain decomposition method on a transputer network*, Nuclear Science and Engineering **114** (1993), 252–270.