Parallel Solution of the Transient Multigroup Neutron Diffusion Equations with Multi-Grid and Preconditioned Krylov-Subspace Methods

Diplomarbeit

zur Erlangung des akademischen Grades
Diplom-Ingenieur der Technischen Mathematik

vorgelegt von
Robert Scheichl

angefertigt am Institut für Mathematik,
Ordinariat für Numerische Mathematik und Optimierung
der Technisch-Naturwissenschaftlichen Fakultät
der **Johannes Kepler Universität Linz**

in Kooperation mit der
Zentralabteilung Technologie der **Siemens AG München**
Betreuer: Dr. Meinhard Paffrath

eingereicht bei
O.Univ.-Prof. Dr. Ulrich Langer

Linz, September 1997

Abstract

For the safety and the control of a nuclear power plant it is necessary to simulate the occurring processes on a computer system. The centre point of concern in the nuclear reactor simulation is the reactor core. The nuclear fission in the reactor core can be best described with a neutron balance. The according three-dimensional problem is described by a complicated system of coupled parabolic partial differential equations and the solution can involve very intensive computing. It is therefore essential to find effective algorithms.

The continuing progress in the area of multiprocessor technology motivates the application of parallel algorithms. Numerous parallel core analysis methods have been reported in literature, but the results are not quite satisfactory yet. The aim of this dissertation was to find an effective solution method that allows for good speedup results even for 10 or more processors and which is competitive to the existing parallel methods, or even better.

The system of partial differential equations is discretized using the physically induced spatial discretization method NEM-M0 and the time discretization TR-BDF2. The arising linear equation systems 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. It can be used for the smoothing in the Multi-Grid algorithm and for the preconditioning of Bi-CGStab.

The parallelization strategy is based on a grid partitioning that distributes the data and the work homogeneously on the processors. The implementation was carried out on an SGI Power Challenge multiprocessor computer with shared memory and 12 MIPS-R8000 processors by spawning threads of the program.

The program was tested for three typical reactor simulation problems on grids with different fineness. The performance of parallel Multi-Grid and parallel Bi-CGStab is outstanding for the particular examples, and for some problem sizes the speedup from the serial to the parallel run is even superlinear. These speedup results are moreover of practical interest, as the parallel execution times are better than the parallel execution times of the established reactor simulation code PANBOX2 of the Siemens AG Operating Division KWU.

Zusammenfassung

Um den sicheren Betrieb eines Kernkraftwerks zu gewährleisten, ist es notwendig, die Abläufe und Prozesse auf Computersystemen zu simulieren. Der zentrale Punkt ist die Simulation des Reaktorkerns. Die Kernspaltung im Reaktorkern kann am besten durch eine Neutronenbilanz beschrieben werden. Das entsprechende dreidimensionale Problem wird durch ein kompliziertes System gekoppelter parabolischer partieller Differentialgleichungen beschrieben, deren Lösung sehr rechenintensiv sein kann. Es ist deshalb notwendig, effektive Algorithmen zu finden.

Wegen des anhaltenden Fortschritts im Bereich der Mehrprozessortechnologie bietet sich die Anwendung von parallelen Algorithmen an. Zahlreiche parallele Methoden zur Simulation der Vorgänge im Reaktorkern sind bereits in der Literatur zu finden, aber die Ergebnisse sind noch nicht zufriedenstellend. Das Ziel dieser Diplomarbeit war es, ein effektives Lösungsverfahren zu finden, das auch für mehr als 10 Prozessoren noch gute Speedup-Resultate zuläßt und besser als die existierenden parallelen Methoden ist.

Für die Ortsdiskretisierung des partiellen Differentialgleichungssytems wird NEM-M0, ein aus der Physik des Problems hergeleitetes Diskretisierungsverfahren, verwendet. Die Zeitdiskretisierung ist TR-BDF2, eine Kombination der Trapezregel mit BDF(2). Die entstehenden linearen Gleichungssyteme werden sowohl mit Mehrgitterverfahren als auch mit vorkonditioniertem Bi-CGStab gelöst, wobei der Kern beider Verfahren aus einem effektiven Block-SOR Verfahren besteht, das die spezielle Struktur der linearen Gleichungssyteme ausnützt und sowohl als Glätter für das Mehrgitterverfahren als auch als Vorkonditionierer in Bi-CGStab dient.

Die Parallelisierung basiert auf einer Gitter-Partitionierung, durch die die Daten und die Arbeit gleichmäßig auf die Prozessoren verteilt wird. Implementiert wurde das Programm auf einer SGI Power Challenge, einem Parallelrechner mit gemeinsamem Speicher und 12 MIPS-R8000 Prozessoren, unter Verwendung von "Threads".

Das Programm wurde anhand von drei typischen Reaktorsimulationsproblemen auf verschieden feinen Gittern getestet und sowohl das parallele Mehrgitterverfahren als auch das parallele Bi-CGStab liefern exzellente Ergebnisse, sowohl beim Speedup als auch bei den Laufzeiten. Der Speedup ist fast immer linear und sogar überlinear für manche Problemgrößen, was auch von praktischem Interesse ist, weil deshalb die parallelen Laufzeiten besser sind als die parallelen Laufzeiten des etablierten Reaktorsimulationsprogramms PANBOX2 des Geschäftsbereichs KWU der Siemens AG.

Contents

Chapter 1

Introduction

The safest way to run a nuclear power plant is definitely the Austrian way: Build a plant and don't switch it on – as happened at Zwentendorf/Austria.

The production of electrical energy in nuclear power plants is one of the most controversial topics in recent history. In Berlin 1938, Otto Hahn managed to carry out the first nuclear fission of a uranium isotope. In 1942 a group of scientists around Enrico Fermi built the first nuclear reactor in Chicago. Since then, there has been huge progress in the field of nuclear physics and nuclear power plants have been built all over the world. But technological advances always bring their own problems, and in this case, safety analysis has to keep ahead of all the new developments.

For the safety and the control of nuclear power plants it is necessary to simulate the occurring processes on a computer system. Computer simulations can be used to determine the design of the nuclear reactor, its fuel rods and control rods, the design of the cooling mechanisms and the steam generators. The interaction of the modules can be simulated at different stages and in different setups. It is also possible to simulate malfunctions and accidents in the nuclear reactor and test whether the procedures in an emergency are appropriate and sufficient.

The centre point of concern in the nuclear reactor simulation is the reactor core. Since it is the source of the energy that is produced in the reactor, a very accurate model of the occurring 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.

The acording three-dimensional problem is modelled by a complicated system of coupled partial differential equations, and therefore the 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. Another way that has been investigated is the exploitation of symmetries within the reactor core.

However the solution of these simplified models is not sufficient and it is necessary to find effective algorithms for the solution of full three-dimensional models. The continuing progress in the area of multiprocessor technology gives rise to investigate the application of parallel algorithms. Numerous parallel core analysis methods have been reported in literature, e.g. Finnemann et al. [10, 11] and Hammer, Paffrath et al. [19], or [7], [20], [24], [32], [34], [36].

For the neutron kinetics in the reactor core they all use the *multigroup neutron diffusion equations* as their mathematical model. In [10], [11] and [19] these equations are discretized using a special finite volume method, NEM-M2 (developed in Finnemann [8]), and a heuristic exponential time integration scheme related to the implicit backward-Euler. The arising linear equation systems are solved by using a multilevel method that is based on a Red-Black Gauss-Seidel relaxation on the finest mesh and Coarse-Mesh Rebalancing (CMR) techniques on the coarser grids.
In the 1980s this method was parallelized on a distributed memory multiprocessor system consisting of Intel-8086/87 processors ([10], [11]). The capacity of these processors can not be compared with modern multiprocessor systems anymore and it is therefore not possible to compare the speedup results. They only considered two-dimensional problems on very fine grids. The good speedup results that they obtained are in fact only possible because of the large problem sizes (~ 7000 nodes) compared to the low processing speed of their processors.
The parallelization of the methods on modern multiprocessor systems has been carried out only recently and is described in [19]. The speedup results were obtained on an HP-725 workstation cluster and on an SGI Power Challenge shared-memory multiprocessor computer. On a three-dimensional grid with ~ 4000 nodes which is currently used for most practical simulations, a speedup factor of 3.7 on 8 processors is the maximum that can be achieved. Finer grids are necessary to improve the results. On a grid with ~ 16000 nodes it is possible to obtain a speedup of 7.3 on 8 processors.

A conclusion of those results, as mentioned in [19, ch. 4], is that it may be more efficient to use a finer grid, but decrease the order of the discretization by using NEM-M0 instead of NEM-M2 (see Finnemann [8]). In this way, the mesh refinement would preserve the accuracy of the NEM-M2 solution. The losses due to the increase in the problem size would be compensated by the better speedup behaviour and the simpler numerical model.

Investigating the feasibility of this proposal gave the impetus to the research in this dissertation. However, to increase the efficiency of the parallel algorithms was not the only motivation for the simulation on finer grids. The data is available on finer grids than the ones used in the moment. The simulation codes that have been developed in the past only used the coarser grids because of a lack of computer memory and computer speed. The aim was therefore to find an effective solution method that allows for good speedup results even for 10 or more processors on coarse and fine grids and which is competitive to the existing parallel methods on the coarser grids.

Chapter 2 gives a brief introduction into the physics of a nuclear reactor (source: Duderstadt/Hamilton [4], Emmendorfer/Höcker [5, 6] and Schmid [33]). Mathematical models are developed to describe the conservation of neutrons and their density-distribution in the reactor. The model that is finally used for the numerical calculations is the multigroup neutron diffusion. The boundary and initial conditions for the multigroup neutron diffusion equations are developed and in the last section, the aspects of an entire reactor simulation are outlined.

Chapter 3 shows the discretization of the three-dimensional time-dependent multigroup neutron diffusion equations with the line method. For the spatial discretization, a nodal method proposed by Finnemann [8], NEM-M0, is used. It is described in full detail in the first section. On a rectangular grid the discretization is obtained by integrating over the boxes and their surfaces. The NEM-M0 method can therefore be interpreted as a variant of the finite volume method. In the second section a time discretization method proposed by D.J. Rose et al. [1], TR-BDF2, is defined. It combines the trapezoidal rule and the second-order backwards-differentiation formula BDF(2). The error can be easily estimated and allows a straightforward control of the time step. The TR-BDF2 method is then used to obtain a full discretization of the multigroup neutron diffusion equations.

Chapter 4 presents the solution methods that were applied to the arising linear equation systems. The linear equation systems are very large with sparse, nonsymmetric matrices. Multi-Grid methods (see Hackbusch [17], Langer [28] or Wesseling [39]) are known to perform best for a wide range of problems that involve the solution of sparse linear equation systems arising from partial differential equations in three dimensions, at least for large problem sizes. For symmetric positive definite problems the optimality is even proved. Therefore Multi-Grid was chosen for the solution of the linear equation systems. For comparison, a preconditioned Krylov-Subspace method, namely Bi-CGStab (see Van der Vorst [35]), was implemented. The kernel of both solution methods is an effective Block-SOR method that makes use of the particular structure of the linear equation systems. It can be used for the smoothing in the Multi-Grid algorithm and for the preconditioning of Bi-CGStab. Convergence tests are presented in the last section of the chapter.

Chapter 5 shows the parallelization of the algorithms that were defined in the previous chapter and the implementation on a shared-memory machine. The parallelization is achieved by partitioning the grid and distributing the data on the processors (section 5.1). Because of the NEM-M0 discretization, no components of the vectors belong to two subdomains. Therefore the basic vector operations that are needed for the Multi-Grid method and for Bi-CGStab can be executed in parallel. The scalar product and the matrix-vector multiplication are the only operations that have data dependencies and require communication between the processes. Finally the Block-SOR method needs to be modified at the subdomain boundaries. The implementation has been carried out on a multiprocessor computer with shared memory using system functions to spawn and control the subprocesses, called threads (section 5.2).

Chapter 6 finally presents the numerical results. The described solution methods, Multi-Grid and preconditioned Bi-CGStab, were applied to the linear equation systems arising in the simulation process of realistic reactor problems to test their robustness and efficiency. The performance of the two methods is compared for one time step and over a whole period of the simulation to test the behaviour of the integration method and the time step control. To assess the performance of the parallel methods, the speedup from the serial to the parallel run is calculated and compared to the speedup results reported in the literature (Hammer, Paffrath et al. [19]). A comparison of the execution times with the execution times of the established reactor simulation code PANBOX2 of the SIEMENS AG Operating Division KWU [3] is given in the final section of the chapter.

Acknowledgments

The dissertation was developed in cooperation with Siemens AG in Munich.

My thanks go first of all to my supervisor in Munich, Dr. Meinhard Paffrath from Siemens ZT-AN2, who gave me the idea of the subject for the dissertation. He remained supportive throughout my work by providing helpful background literature and data and his advice was invaluable for the presentation and format of my results.

Great thanks are also offered to Prof. Dr. Ulrich Langer, who was always available to discuss my work and progress and give me fresh new ideas.

Finally I would like to thank Prof. Dr. Albert Gilg from Siemens ZT-AN2, Dr. Herbert Finnemann from Siemens KWU-BTMC and all my colleagues at Siemens for their friendly support. A special thank-you to all of them and especially Georg who never seemed to tire of my relentless questioning about LaTeX.

Last but certainly not least, I would like to thank my girlfriend Julie Fraser for correcting my English.

Chapter 2

Simulation of a nuclear reactor

The following sections are intended to give a brief introduction into the physics of a nuclear reactor and to develop a mathematical model to describe the conservation of neutrons and their density-distribution in the reactor. The last section of this chapter gives a brief overview of an entire reactor simulation. The whole chapter relates a lot to chapters 4 and 5 of Schmid [33]. For an extensive and for a more detailed description of the physics see Emmendorfer/Höcker [5, 6] or Duderstadt/Hamilton [4].

2.1 Basic concepts of reactor physics

The starting point for the description of a nuclear reactor is the fact that a *nuclear fission* of a few isotopes with large numbers of protons and neutrons, e.g. Uranium ^{235}U or Plutonium ^{239}Pu[1], can be achieved through energetic induction. In this process more energy is set free, than is necessary for the induction. That is what makes a nuclear power plant possible.

In a nuclear reactor the nuclear fission is achieved by a chain reaction. Through an interaction with a free neutron a fissionable isotope can be induced to decay into different fission products and thereby release, mostly kinetic, energy. Amongst the resulting products are, once again, free neutrons that can induce new nuclear fissions, etc.

A good way to describe this chain reaction is 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.

Before this balance can be formed, nuclear fission and the resulting fission products have to be discussed in more detail (see Emmendorfer/Höcker [5, ch. I.3]):

[1]In the following ^{235}U will always be used as an example of a fissionable isotope

When a neutron is absorbed by the Uranium nucleus, it splits mostly into two fission products with high energy: e.g. Strontium ^{97}Sr and Xenon ^{139}Xe. They emit their energy almost immediately by "evaporating" so called *prompt neutrons* and by other radioactive decays. The number of prompt neutrons per fission lies between two and three on average,but it depends on the type of isotope and the energy, etc. They are generated 10^{-4} - 10^{-7} seconds after the fission and are therefore termed prompt. The resulting nuclei follow the β-decay sequence and end up in stable isotopes.

In a few cases one of the nuclei in the β-decay sequence of a fission product also emits neutrons, known as *delayed neutrons*. Depending on the half-life of the neutron *precursor*, e.g. Bromine ^{87}Br ($\tau_{1/2} = 55.7$s), they are generated about 10^{-1} - 10^{1} seconds after the fission (therefore delayed).
Only about 1% of the generated neutrons are delayed, but they are the ones that make it possible to control the chain reaction in the reactor because they increase the average generation-time of neutrons to 10^{-2} - 10^{-1} seconds. The difference in the time scales between prompt and delayed neutrons is important for the numerical treatment of the transient simulation because it has an effect on the *stiffness* of the differential equations.

To quantify the nuclear chain reaction, a *multiplication constant*

$$\lambda_0 := \frac{\text{neutron production}}{\text{neutron loss}} \text{ (per unit} - \text{time)}$$

is defined for the reactor. A reactor is called: *subcritical*, if the constant $\lambda_0 < 1$ and the chain-reaction diminishes, *critical*, if $\lambda_0 = 1$ and the chain-reaction is constant, *supercritical*, if $\lambda_0 > 1$ and the chain-reaction grows. The aim is to always maintain the reactor in the critical phase.

The factors that play a role in the construction of the neutron balance are nuclear fission, the interactions of the free neutrons with the atomic nuclei and the leakage through the surfaces. They all depend on the energy and the direction of motion of the free neutrons and furthermore they are space- and time-dependent.
One has to distinguish between two different sorts of interactions, which are either the *absorption* of the neutron by the nucleus, or the *scattering* of the neutron, where the neutron collides with the nucleus and changes its energy and direction. For the description of these phenomena only the number of interactions per unit-time are important. In reactor physics, so called *cross sections* are used to measure these interactions. Microscopic cross sections $\sigma(E, \vec{x}, t, \vec{\Omega})$ are functions that depend on the energy E, the location $\vec{x}$, the time t and the direction $\vec{\Omega}$. They can be thought of as the fictitious areas, with a nucleus at location $\vec{x}$, at time t, that are presented

to a stream of neutrons with energy E in direction $\vec{\Omega}$. But to be more exact they measure the probability of the specified interactions taking place.
For the construction of an equation for the neutron balance only macroscopic cross sections $\Sigma(E, \vec{x}, t, \vec{\Omega}) = \sigma(E, \vec{x}, t, \vec{\Omega}) \cdot N$ are important. They are defined as the product of the microscopic cross sections with the number N of atomic nuclei per cm^3 and make it possible to quantify the fission- and scattering processes on a macroscopic scale. These functions have in general jumps at the boundaries of two different regions and this is again important for the numerical treatment. The neutron balance can now be modelled using an approach similar to *Boltzmann's transport equation* in gas dynamics.

2.2 Neutron-balance: A transport-theoretical approach

Similar to the approach Boltzmann took for the description of gas dynamics, a transport equation in the phase space is used to describe the conservation of the number of neutrons in a nuclear reactor.
The fundamental quantity of the transport equation is the *angular neutron flux*

$$\Phi(E, \vec{x}, t, \vec{\Omega}) := v \cdot n(E, \vec{x}, t, \vec{\Omega}), \tag{2.1}$$

where $n(E, \vec{x}, t, \vec{\Omega}) dE d\vec{\Omega} dV$ is the number of neutrons in the volume dV at location $\vec{x}$, with energy $E \pm dE$ and whose direction of motion is in the cone formed by $d\vec{\Omega}$ around $\vec{\Omega}$. v is the magnitude of the velocity vector.
The already mentioned cross sections Σ will be used to quantify the relevant influences on Φ. In contrast to the Boltzmann equation external forces and interactions between two neutrons can be neglected. A detailed description of the terms can be found in Duderstadt/Hamilton [4, ch. 4] or Emmendorfer/Höcker [5, ch. VI].

The relevant terms for the production of neutrons are:

- *isotropic* fission:
  - prompt neutrons

$$(1 - \beta_j)\chi_p^j(E) \int_{E'} \frac{1}{4\pi} \nu^j(E') \Sigma_f^j(E', \vec{x}, t) \int_{\vec{\Omega}'} \Phi(E', \vec{x}, t, \vec{\Omega}') d\vec{\Omega}' dE'$$

  - delayed neutrons

$$\chi_d^i(E) \lambda_i C_i(\vec{x}, t)$$

The unknown functions $C_i(\vec{x},t)$ describe the concentrations of the different precursor nuclei i $(= 1, ..., I)$. The balance equations for these precursor concentrations have to be solved simultaneously (see below equation (2.3)). On the other hand the index j $(= 1, ..., J)$ represents the different types of fissionable isotopes. The parameter $\beta^j = \sum_{i=1}^{I} \beta_i^j$. β_i^j denotes the fraction of all fission neutrons (both prompt and delayed) that appear from the ith precursor group and from the jth fissionable isotope. The energy dependent parameters χ denote the *fission spectra*, i.e. the amount of neutrons that are set free with energy E. Finally the parameter $\nu(E')$ denotes the average number of neutrons emitted per fission triggered by a neutron of energy E'. The factor $1/4\pi$ is due to the normalization on a 3D-element in the case of an isotropic distribution of the generated neutrons.

- *inscattering*: scattering from other energy-levels E' and other directions $\vec{\Omega}'$,

$$\int\limits_{E'=0}^{\infty} \int_{\vec{\Omega}'} \Sigma_s(\vec{x}, t, E', \vec{\Omega}' \to E, \vec{\Omega}) \Phi(E', \vec{x}, t, \vec{\Omega}') dE' d\vec{\Omega}'.$$

The term $\Sigma_s(\vec{x}, t, E', \vec{\Omega}' \to E, \vec{\Omega})$ is called the *differential scattering cross section* and describes the probability of a neutron changing its energy and direction from $(E', \vec{\Omega}')$ to $(E, \vec{\Omega})$.

Terms for the loss of neutrons are:

- leakage through the surfaces, due to Gausses theorem:

$$div(\vec{\Omega} \cdot \ \Phi(E, \vec{x}, t, \vec{\Omega})) = \vec{\Omega} \cdot grad \ \Phi(E, \vec{x}, t, \vec{\Omega})$$

- (total) collisions; absorption and scattering,

$$\Sigma_t(E, \vec{x}, t) \Phi(E, \vec{x}, t, \vec{\Omega})$$

Altogether the neutron balance reads as

"Fission + Inscattering - Leakage - Loss through collisions
= Temporary change of angular neutron flux",

or as an integrodifferential equation for the angular neutron flux:

$$\begin{aligned}\frac{1}{\lambda_0}\sum_{j=1}^{J}(1-\beta_j)\chi_p^j(E)\int_{E'}\frac{1}{4\pi}\nu^j(E')\Sigma_f^j(E',\vec{x},t)\int_{\vec{\Omega}'}\Phi(E',\vec{x},t,\vec{\Omega}')d\vec{\Omega}'dE' +\\ \sum_{i=1}^{I}\chi_d^i(E)\lambda_i C_i(\vec{x},t)+\int_{E'}\int_{\vec{\Omega}'}\Sigma_s(\vec{x},t,E',\vec{\Omega}'\to E,\vec{\Omega})\Phi(E',\vec{x},t,\vec{\Omega}')dEd\vec{\Omega}' - \\ \vec{\Omega}\cdot grad\ \Phi(E,\vec{x},t,\vec{\Omega})-\Sigma_t(E,\vec{x},t)\Phi(E,\vec{x},t,\vec{\Omega}) = v^{-1}\frac{\partial}{\partial t}\Phi(E,\vec{x},t,\vec{\Omega})\end{aligned} \tag{2.2}$$

The constellation that the reactor is exactly critical (i.e. the multiplication constant $\lambda_0 = 1$) is obviously not stable. It is therefore necessary to multiply the fission terms in equations (2.2) and (2.3) with $1/\lambda_0$ to guarantee the solution (compare remark 2.2).

The balance equations for the unknown precursor concentrations $C_i(\vec{x},t)$ are given by

$$\frac{\partial C_i}{\partial t}+\lambda_i C_i = \frac{1}{\lambda_0}\sum_{j=1}^{J}\beta_i^j\int_{E'}\frac{1}{4\pi}\nu^j(E')\Sigma_f^j(E')\int_{\vec{\Omega}'}\Phi(E',\vec{\Omega}')d\vec{\Omega}'dE', \tag{2.3}$$

with the decay constant λ_i for the precursor nucleus i.

Remark 2.1 *To simplify the following reflections only one fissionable isotope, $J = 1$, will be considered and the multiplication constant λ_0 as well as the delayed neutrons are left out for the moment. They will be included later on in section 2.5. In the following sections the fission terms in the balance equation are therefore replaced by*

$$\chi(E)\int_{E'}\frac{1}{4\pi}\nu(E')\Sigma_f(E',\vec{x},t)\int_{\vec{\Omega}'}\Phi(E',\vec{x},t,\vec{\Omega}')d\vec{\Omega}'dE'$$

and the equations for the precursor concentrations do not need to be considered.

2.3 Simplifications and the transition to the neutron diffusion equation

The transport equation is a very accurate description of the processes taking place in the reactor core. Since it is often too difficult and too costly to solve it, the equation has to be simplified:
As a first step the dependence on the direction of motion has to be simplified. Henceforth the integral quantities

$$\text{neutron flux density:} \qquad \phi(E,\vec{x},t) = \int_{\vec{\Omega}} \Phi(E,\vec{x},t,\vec{\Omega})d\vec{\Omega}, \tag{2.4}$$

$$\text{neutron current density:} \qquad \vec{j}(E,\vec{x},t) = \int_{\vec{\Omega}} \vec{\Omega}\cdot\Phi(E,\vec{x},t,\vec{\Omega})d\vec{\Omega}. \tag{2.5}$$

are used.

2.3.1 Isotropic scattering

The easiest assumption is that the scattering of the neutrons is isotropic and that it dominates the absorption, i.e. absorption is almost 0. Integration of (2.2) with respect to $\vec{\Omega}$ results in a neutron balance equation that does not depend on $\vec{\Omega}$ anymore:

$$\begin{aligned} &\chi(E)\int_{E'}\nu(E')\Sigma_f(E',\vec{x},t)\,\phi(E',\vec{x},t)dE' + \int_{E'}\Sigma_s(E'\to E,\vec{x},t)\,\phi(E',\vec{x},t)dE' - \\ &\quad div\;\vec{j}(E,\vec{x},t) - \Sigma_t(E,\vec{x},t)\,\phi(E,\vec{x},t) \;=\; v^{-1}\frac{\partial}{\partial t}\,\phi(E,\vec{x},t) \end{aligned} \tag{2.6}$$

To find a relation between the flux and the current density of the neutrons, an element dA in the centre of the xy-plane of the coordinate system is taken and the downward current j_{z-} through the surface element is derived. To achieve that, the flux is developed in a Taylor Series and cut off after the linear term. That approximation is used to evaluate the integral that defines the downward current and it is approximately found to be

$$j_{z-} \approx \left.\frac{\phi}{4}\right|_{z=0} + \frac{1}{6\,\Sigma_t}\left.\frac{\partial\phi}{\partial z}\right|_{z=0}. \tag{2.7}$$

In the same way the upward current j_{z+} is found as

$$j_{z+} \approx \left.\frac{\phi}{4}\right|_{z=0} - \frac{1}{6\,\Sigma_t}\left.\frac{\partial\phi}{\partial z}\right|_{z=0}. \tag{2.8}$$

The surplus current j_z in positive z-direction, commonly referred to as the current density, is the difference between j_{z+} and j_{z-}:

$$j_z = j_{z+} - j_{z-} = \frac{1}{3\,\Sigma_t}\left.\frac{\partial\phi}{\partial z}\right|_{z=0}$$

Analogously the current density for the other directions can be found and the flux and the current density can be related by *Fick's law*

$$\vec{j}(E,\vec{x},t) = -D(E,\vec{x},t)\,grad\;\phi(E,\vec{x},t), \tag{2.9}$$

where the factor $D(E,\vec{x},t) := 1/(3\,\Sigma_t(E,\vec{x},t))$ is called the elementary *neutron diffusion coefficient.*

2.3.2 A more accurate first order approximation of the transport equation

If the scattering is not isotropic, a more accurate approximation of the transport equation is necessary. Therefore the quantities that depend on the direction $\vec{\Omega}$, i.e. the angular flux density and the differential scattering cross section, are developed in *Legendre polynomials* and cut off at index 1 (P_1*-approximation*, Emmendorfer/Höcker [5, ch. VI.2]).

If the angular flux density Φ only depends on one spatial coordinate, e.g. z, it can be developed in Legendre polynomials with respect to $\mu = cos\ \vartheta$, where ϑ denotes the angle between $\vec{\Omega}$ and $\vec{i_z}$. As all directions of motion with *polar angle* ϑ occur with the same probability, the flux does not depend on the *azimuth angle* φ:

$$\Phi(E, z, t, \vec{\Omega}) = \frac{1}{2\pi} \int\limits_{\varphi=0}^{2\pi} \Phi(E, z, t, \vec{\Omega}) d\vec{\Omega} = \frac{1}{2\pi} \Phi(E, z, t, \mu) \tag{2.10}$$

The dependence of $\Phi(E, z, t, \vec{\Omega})$ on the direction of motion ,i.e. the variable μ, can now be separated by using Legendre polynomials $P_l(\mu)$ and then cutting off at index 1:

$$\begin{aligned} \Phi(E, z, t, \mu) &= \sum_{l=0}^{\infty} \frac{2l+1}{2} \phi_l(E, z, t) P_l(\mu) \\ &\approx \frac{1}{2}\phi_0(E, z, t) P_0(\mu) + \frac{3}{2}\phi_1(E, z, t) P_1(\mu) \end{aligned} \tag{2.11}$$

As $P_0(\mu) = 1$ and $P_1(\mu) = \mu$ the coefficient functions can be found as

$$\begin{aligned} \phi_0(E, z, t) &= \int_{-1}^{1} \Phi(E, z, t, \mu) d\mu = \int_{\vec{\Omega}} \Phi(E, z, t, \vec{\Omega}) d\vec{\Omega} = \phi(E, z, t), \\ \phi_1(E, z, t) &= \int_{-1}^{1} \mu\, \Phi(E, z, t, \mu) d\mu = \int_{\vec{\Omega}} \vec{i_z} \cdot \vec{\Omega}\, \Phi(E, z, t, \vec{\Omega}) d\vec{\Omega} = j_z(E, z, t). \end{aligned}$$

The coefficients are exactly the integral flux density (2.4) and the z-component of the integral current density (2.5) already used in equation (2.6) of the previous section 2.3.1. If these coefficients are inserted in equation (2.11) and (2.10) is used, the following linear approximation of Φ can be stated:

$$\Phi(E, z, t, \vec{\Omega}) = \frac{1}{2\pi}\Phi(E, z, t, \mu) = \frac{1}{4\pi}\phi(E, z, t) + \frac{3}{4\pi} \underbrace{\vec{\Omega} \cdot \vec{j}(E, z, t)}_{=\mu\, j_z(E,z,t)} \tag{2.12}$$

Since the selection of the coordinate direction was arbitrary, the linearization can be generalized by symmetry to

$$\Phi(E,\vec{x},t,\vec{\Omega}) = \frac{1}{4\pi}\phi(E,\vec{x},t) + \frac{3}{4\pi}\vec{\Omega}\cdot\vec{j}(E,\vec{x},t). \tag{2.13}$$

The differential scattering cross section $\Sigma_s(\vec{x},E',\vec{\Omega}' \rightarrow E,\vec{\Omega})$ on the other hand only depends on $cos\ \psi = \vec{\Omega}\cdot\vec{\Omega}'$, where ψ describes the angle between the two directions $\vec{\Omega}$ and $\vec{\Omega}'$. Therefore Legendre polynomials can be used again to find a linear approximation for Σ_s in the form:

$$\Sigma_s(\vec{x},t,E',\vec{\Omega}' \rightarrow E,\vec{\Omega}) = \frac{1}{4\pi}\Sigma_s(E' \rightarrow E,\vec{x},t) + \frac{3}{4\pi}\Sigma_{s1}(E' \rightarrow E,\vec{x},t)(\vec{\Omega}\cdot\vec{\Omega}') \tag{2.14}$$

Once more $\Sigma_s(E' \rightarrow E,\vec{x},t)$ is by definition the same function as in equation (2.6).

Φ and Σ_s in the transport equation (2.2) are replaced with its approximations (2.13) and (2.14) and integrated with respect to $\vec{\Omega}$. Because of the orthogonality of the Legendre polynomials, the resulting equation is again the neutron balance equation (2.6) found in the previous section 2.3.1.

Unlike the heuristic approach of section 2.3.1, the dependency of the flux and the current density can now be found, by multiplying equation (2.2) with $\vec{\Omega}$ and then integrating it with respect to $\vec{\Omega}$. The result is

$$\frac{1}{3}grad\ \phi(E,\vec{x},t) + \Sigma_t(E,\vec{x},t)\vec{j}(E,\vec{x},t) = \int_{E'}\Sigma_{s1}(E' \rightarrow E,\vec{x},t)\vec{j}(E',\vec{x},t)dE'. \tag{2.15}$$

In the isotropic case ($\Sigma_{s1} = 0$), the equation reduces to Fick's law (2.9) again. In reality the conditions for isotropic scattering never occur, but for all neutrons that are not too fast at least $\Sigma_{s1} \ll \Sigma_s$ holds, i.e. the scattering is just slightly anisotropic. The scattering term in (2.15) can then be interpreted as a correction term for Fick's law (2.9) and, as in the isotropic case, (2.15) reduces to Fick's law (2.9), only with a different diffusion coefficient[2]

$$D(E,\vec{x},t) := \frac{1}{3\left(\Sigma_t(E,\vec{x},t) - \Sigma_s(E,\vec{x},t)\cdot\overline{cos\ \psi}\right)}. \tag{2.16}$$

ψ in this definition describes the angle between the direction of the neutron before and after the collision and $\overline{cos\ \psi}$ denotes the mean value over all cosines.

[2] To be more exact the cross section Σ_s has to be replaced by the cross section for elastic scattering Σ_e, i.e. scattering, where the nuclei are not changing their energy.

2.3.3 The neutron diffusion equation

If $\vec{j}$ from Fick's law (2.9) is substituted in (2.6), the *neutron diffusion equation* is found

$$\begin{aligned} v^{-1}\frac{\partial}{\partial t}\,\phi(E,\vec{x},t) \;&=\; div(D(E,\vec{x},t)\,grad\;\phi(E,\vec{x},t)) - \Sigma_t(E,\vec{x},t)\,\phi(E,\vec{x},t) + \\ \chi(E)\int_{E'}\nu(E')\Sigma_f(E',\vec{x},t)\,\phi(E',\vec{x},t)dE' &+ \int_{E'}\Sigma_s(E'\to E,\vec{x},t)\,\phi(E',\vec{x},t)dE'. \end{aligned} \tag{2.17}$$

The diffusion coefficient $D(E,\vec{x},t)$ is always chosen as the corrected term (2.16) from section 2.3.2 to get the best possible approximation of the transport equation (2.2).

It should be noted that near the boundary the diffusion theory is a poor representation of the neutron flux. Nevertheless boundary conditions have to be found for the interface of different regions of the reactor, as well as for the outer boundaries, for a full mathematical description of the problem. The used boundary conditions are intended to yield the proper flux in the "interior" of the reactor, that is, some small distance away from the reactor boundary.

2.4 Boundary conditions for the neutron diffusion

There are three different boundary conditions that need to be considered: interface conditions for the connection of two different regions, *vacuum boundary conditions* and *Albedo boundary conditions* for reflecting media (see [4, p. 141ff], [33, p. 89ff], [5, p. 248ff] and [6, p. 300]).
As the diffusion theory is only an approximation of the transport theory for the neutron balance, the boundary conditions for the neutron transport equation should be developed first.

Interface conditions

At the interface $\mathbf{\Gamma}_{12}$ between a region 1 and a region 2 the angular flux $\Phi(E,\vec{x},t,\vec{\Omega})$ of section 2.2 shall be continuous. If $u|_{\to\mathbf{\Gamma}}$ and $u|_{\mathbf{\Gamma}\leftarrow}$ are defined as the limits of function u from the left and from the right at a boundary $\mathbf{\Gamma}$ respectively, the interface condition can be formulated as

$$\Phi(E,\vec{x},t,\vec{\Omega})\Big|_{\to\mathbf{\Gamma}_{12}} = \Phi(E,\vec{x},t,\vec{\Omega})\Big|_{\mathbf{\Gamma}_{12}\leftarrow} \tag{2.18}$$

In the diffusion approximation only the continuity of the integral quantities of the flux (2.4) and the current density (2.5) of section 2.3 can be required at the interface, i.e.

$$
\begin{aligned}
& \phi(E,\vec{x},t)|_{\rightarrow\mathbf{\Gamma}_{12}} = \phi(E,\vec{x},t)|_{\mathbf{\Gamma}_{12}\leftarrow} \\
\text{and} \quad & \vec{j}(E,\vec{x},t)\Big|_{\rightarrow\mathbf{\Gamma}_{12}} = \vec{j}(E,\vec{x},t)\Big|_{\mathbf{\Gamma}_{12}\leftarrow} \\
\text{or equivalently} \quad & D(E,\vec{x})\tfrac{\partial\phi}{\partial n}(E,\vec{x},t)\Big|_{\rightarrow\mathbf{\Gamma}_{12}} = D(E,\vec{x})\tfrac{\partial\phi}{\partial n}(E,\vec{x},t)\Big|_{\mathbf{\Gamma}_{12}\leftarrow} .
\end{aligned}
\tag{2.19}
$$

These conditions require discretization methods for the numerical treatment that take both the flux Φ and the current density $\vec{j}$ into account. A well known method in that context is the *mixed finite element method*, but in this dissertation another class of methods will be applied that was especially developed for the solution of the neutron diffusion equation, known as the *nodal methods*.

Vacuum boundary conditions

The most important boundary condition at an outer boundary $\mathbf{\Gamma}_V$ of the reactor is the vacuum boundary condition. Since the density of air is very small and the space around the reactor is suitably large, there is no neutron flux into the reactor. For all directions $\vec{\Omega}_{in}$, $\Phi(E,\vec{x},t,\vec{\Omega}_{in})\Big|_{\mathbf{\Gamma}_V} = 0$ is required.
Again this can not be fulfilled in the diffusion theory, but the incoming current density $\vec{j}_{in}$ has to vanish at the boundary:

$$
\vec{j}_{in}(E,\vec{x},t)\Big|_{\mathbf{\Gamma}_V} = \vec{0} \tag{2.20}
$$

If the approximation (2.7) that was used to find Fick's law in section 2.3.1 is taken and $\vec{n}$ is defined to be the outward unit normal on the boundary, (2.20) is equivalent to the following boundary condition :

$$
\phi(E,\vec{x},t) + 2\,D(E,\vec{x},t)\frac{\partial\phi}{\partial n}(E,\vec{x},t) = \vec{0} \quad \forall\vec{x}\in\mathbf{\Gamma}_V \tag{2.21}
$$

Albedo boundary conditions

Albedo boundary conditions are used to model boundaries $\mathbf{\Gamma}_A$, where a fraction α of the neutrons are reflected back into the reactor. Therefore the appropriate boundary condition for the diffusion equation has to be

$$
\vec{j}_{in}(E,\vec{x},t)\Big|_{\mathbf{\Gamma}_A} = \alpha\,\vec{j}_{out}(E,\vec{x},t)\Big|_{\mathbf{\Gamma}_A} . \tag{2.22}
$$

Again a boundary condition with respect to ϕ can be found by using the linear approximations (2.8) and (2.7) for the incoming and outgoing currents,

$$
\frac{(1-\alpha)}{(1+\alpha)}\phi(E,\vec{x},t) + 2\,D(E,\vec{x},t)\frac{\partial\phi}{\partial n}(E,\vec{x},t) = \vec{0} \quad \forall\vec{x}\in\mathbf{\Gamma}_A. \tag{2.23}
$$

On interfaces between different regions of the reactor continuity of the flux and the current density is required. On the outer boundaries on the other hand boundary conditions of mixed Dirichlet-Neumann type, (2.21) or (2.23), have to be fulfilled. Another way is to take both the incoming and the outgoing current densities, $\vec{j}_{in}$ and $\vec{j}_{out}$, as unknowns, which is possible in the framework of the nodal methods. The boundary conditions that have to be fulfilled then, (2.20) and (2.22), are a lot easier.

2.5 The multigroup neutron diffusion equations

To complete the mathematical modelling equation (2.17) has finally got to be discretized with respect to the energy E. *Energy groups* $g = [E_g, E_{g-1}[$ with $0 = E_G \leq ... \leq E_g \leq E_{g-1} \leq ... \leq E_0 \to \infty$ are introduced and the *group flux* ϕ_g is defined to be

$$\phi_g(\vec{x},t) := \int\limits_{E_g}^{E_{g-1}} \phi(E,\vec{x},t)dE \tag{2.24}$$

The group-averaged velocity v_g, diffusion coefficient D_g and cross sections $\Sigma_{\cdot g}$, referred to as the *multigroup constants*, are defined as

$$C_g(\vec{x},t) := \frac{\displaystyle\int_{E_g}^{E_{g-1}} C(E,\vec{x},t)\phi(E,\vec{x},t)dE}{\displaystyle\int_{E_g}^{E_{g-1}} \phi(E,\vec{x},t)dE}. \tag{2.25}$$

The existence of different fissionable isotopes j and the production of delayed neutrons that was left out during the modelling is now included and the multiplication constant λ_0, see remark 2.1, is introduced. The result is a system of *parabolic partial differential equations (PDE)* of second order: the *multigroup diffusion equations*

$$\left.\begin{array}{r} v_g^{-1}\dfrac{\partial}{\partial t}\phi_g(\vec{x},t) - div(D_g(\vec{x},t)\, grad\ \phi_g(\vec{x},t)) + \\ \left(\Sigma_{ag}(\vec{x},t) + \displaystyle\sum_{g'=1}^{G}\Sigma_{gg'}(\vec{x},t)\right)\phi_g(\vec{x},t) \;=\; \displaystyle\sum_{i=1}^{I}\chi_{dg}^{i}\lambda_i C_i(\vec{x},t) + \\ \displaystyle\sum_{g'=1}^{G}\left(\Sigma_{g'g}(\vec{x},t) + \frac{1}{\lambda_0}\sum_{j=1}^{J}(1-\beta^j)\chi_{pg}^{j}\nu\Sigma_{fg'}^{j}(\vec{x},t)\right)\phi_{g'}(\vec{x},t) \end{array}\right\} \quad g = 1...G, \tag{2.26}$$

together with the balance equations for the delayed neutron precursor concentrations

$$\frac{\partial}{\partial t}C_i(\vec{x},t) + \lambda_i C_i(\vec{x},t) = \frac{1}{\lambda_0}\sum_{j=1}^{J}\beta_i^j \sum_{g=1}^{G}\nu\Sigma_{fg}^j(\vec{x},t)\phi_g(\vec{x},t), \qquad i = 1...I. \tag{2.27}$$

The multigroup constant $\Sigma_{g'g}$ describes the scattering from energy group g' into energy group g and replaces the differential scattering cross section $\Sigma_s(E' \to E)$. The scattering cross section Σ_s on the other hand can be replaced by $\sum_{g'=1}^{G}\Sigma_{gg'}$.

The definition of the multigroup constants (2.25) is obviously theoretical since it involves the knowledge of the unknown function $\phi(E,\vec{x},t)$. In practice, only few energy groups are used in reactor calculations, usually between 2 and 8. A common approach to get the multigroup constants for these *coarse group calculations* is to first perform a very finely structured multigroup calculation, neglecting the spatial and time dependence. The resulting approximate flux $\tilde{\phi}$ is then used in definition (2.25) to derive the coarse group constants. This procedure is called *group data collapsing* and has to be performed separately for each region of the reactor in which the material composition varies appreciably. Furthermore these group constants will have to be recalculated whenever these region-wise core properties change, such as during fuel burnup or reactivity adjustment via the movement of control rods. How these calculations are accomplished in an entire reactor simulation is shown in section 2.7.

2.6 Initial conditions, reactor criticality calculations

In the previous section 2.5 a system of parabolic PDEs was found, (2.26) and (2.27), to describe the neutron balance. The boundary conditions that were set up for the energy dependent diffusion equation in section 2.4 can be easily altered for the group equations. The final stage to complete the mathematical model is to find an initial condition for the system.

This initial condition is found by performing a quasi-stationary calculation at the beginning (see Appendix A). In the stationary case (2.27) can be substituted into (2.26) and the system of equations reduces to

$$\left.\begin{aligned} -div(D_g(\vec{x})\,grad\;\phi_g(\vec{x})) + \Big(\Sigma_{ag}(\vec{x}) + \sum_{g'}\Sigma_{gg'}(\vec{x})\Big)\phi_g(\vec{x}) \;- \\ \sum_{g'}\Sigma_{g'g}(\vec{x})\phi_{g'}(\vec{x}) \;=\; \frac{1}{\lambda}\left(\sum_j \bar{\chi}_g^j \sum_{g'}\nu\Sigma_{fg'}^j(\vec{x})\phi_{g'}(\vec{x})\right) \end{aligned}\right\} g = 1...G, \tag{2.28}$$

with $\bar{\chi}_g^j = (1 - \beta^j)\chi_{pg}^j + \sum_i \beta_i^j \chi_{dg}^i$.

The solution of a *generalized eigenvalue problem* is required and the dominant eigenvalue λ_0 (i.e. $|\lambda_0| > |\lambda|$ for all eigenvalues $\lambda \neq \lambda_0$) and the corresponding eigenfunction (see remark 2.2) have to be found. This process is called criticality calculation, as the dominant eigenvalue corresponds exactly to the multiplication constant defined in the first section 2.1 and is therefore a measure of the criticality of the reactor. The eigenvector provides a distribution of "hot spots" and regions where the power is low. The criticality search is therefore an important tool for the design of a reactor that is evenly critical.
As the solution of the eigenvalue problem is not unique, the eigenvector has to be scaled with respect to the measured output power of the reactor to get the initial condition for the transient calculations. That involves once again a coupling of the neutron flux computations with other modules that play a role in the reactor simulation. Therefore it is necessary to take a closer look at all the processes taking place in a nuclear reactor and their interactions with the neutron flux simulation.

Remark 2.2 *There exists a unique dominant eigenvalue for the stationary problem (2.28), it is real and positive and the corresponding eigenfunction is also real and positive. Furthermore there is no second eigenfunction that is everywhere positive. A proof for that statement can be found in Wachspress [37, ch. 3.2.4] or in Schmid [33, ch. 6.1].*
In Wachspress [37, ch. 3.3.3] the same statement is also proofed for the discrete problem using finite differences for the discretization. The fact that the eigenvalue λ_0 is dominant, guarantees the convergence of the inverse iteration for the solution of the discrete problem, if the shift is chosen smaller than $1/\lambda_0$ (compare Appendix A).

2.7 Aspects of an entire reactor simulation

Due to the complexity of the processes occurring in a nuclear power plant, the solution of an entire reactor simulation consists of a number of interacting modules, describing the various processes. Only the perfect interaction of the different modules guarantees a useful simulation.
An example for a software package that can simulate an entire reactor is the coupled code system RELAP5 / PANBOX2 / COBRA 3-CP (R/P/C) of the Siemens AG Operating Division KWU [25]. The different modules that are needed in the simulation and their interactions are outlined in figure 2.1.

The starting point for the simulation is the generation of the input data for the neutronics. To generate them, libraries with the cross sections of the different materials and the geometric data of the reactor core, including the distribution of the fuel rods in the reactor (see figure 2.2) are needed. Finally, the initial conditions, i.e. the temperature and the density of the cooling system and the position of the control rods, have to also be taken into account. Usually the input data is homogenized on a relatively coarse grid formed by the fuel assemblies (see remark 2.3).
In the next step the neutronics module is started to calculate the neutron flux and the multiplication constant λ_0. The results are used in the thermal-hydraulics module to derive the temperature and the density of the coolant. The input data for the neutronics on the other hand has to be corrected with respect to the results of the thermal-hydraulics, etc.
Both the results from the neutronics and from the thermal-hydraulics module are needed in the simulation and control of the plant. In the control adjustment module the control rod movements are determined that are necessary to keep the reactor in the critical phase. In the fuel depletion calculations the long-term effects of the nuclear fission on the composition of the reactor core are evaluated. The fuel depletion and the control rod movements have naturally again an effect on the input data for the neutronics. Due to fission product poisoning, e.g. with Xenon ^{135}Xe, the absorption cross sections grow and the reactivity of the reactor decreases, i.e. the fission cross sections are smaller. If on the other hand a control rod is withdrawn from the core the reactivity increases and the absorptions diminish.
A more detailed description of the modules that are mentioned only briefly here can be found in Duderstadt/Hamilton [4, part 4].

Remark 2.3 *To allow for large 3D-simulations of nuclear reactors, it is common to homogenize the input data in a pre-processing step, i.e. the cross sections for the individual fuel- and control-rods are averaged over an entire fuel assembly (see figure 2.2). In the second step the multigroup diffusion equations are solved numerically on this relatively coarse grid of fuel assemblies, using the locally constant cross sections. In the post-processing step the neutron flux in the individual fuel rods is derived from the results of the calculations on the coarse grid.*

Remark 2.4 *The subject of the dissertation is only the simulation of the neutronics in the reactor using the multigroup diffusion equations. For the calculation and homogenization of the input data and for the coupling of the multigroup diffusion with the other modules the program PANBOX2 [3] was used.*

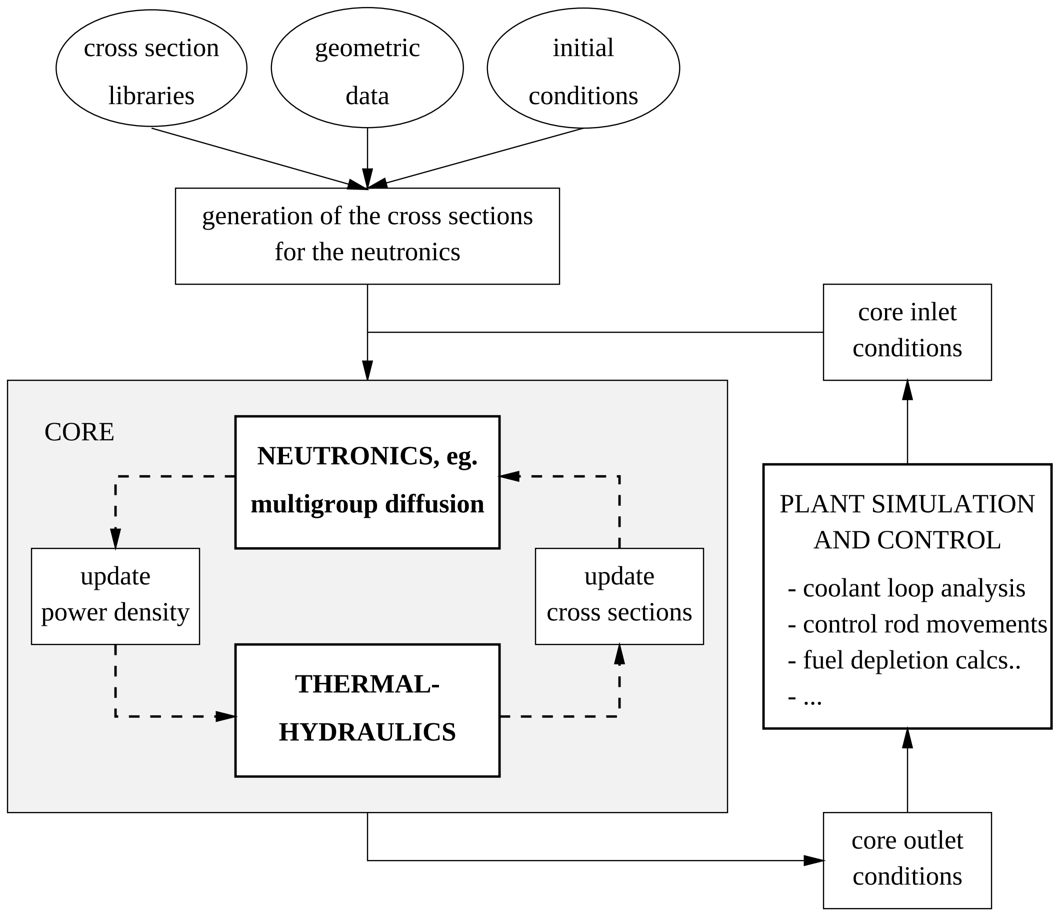


Figure 2.1: Outline of an entire reactor simulation

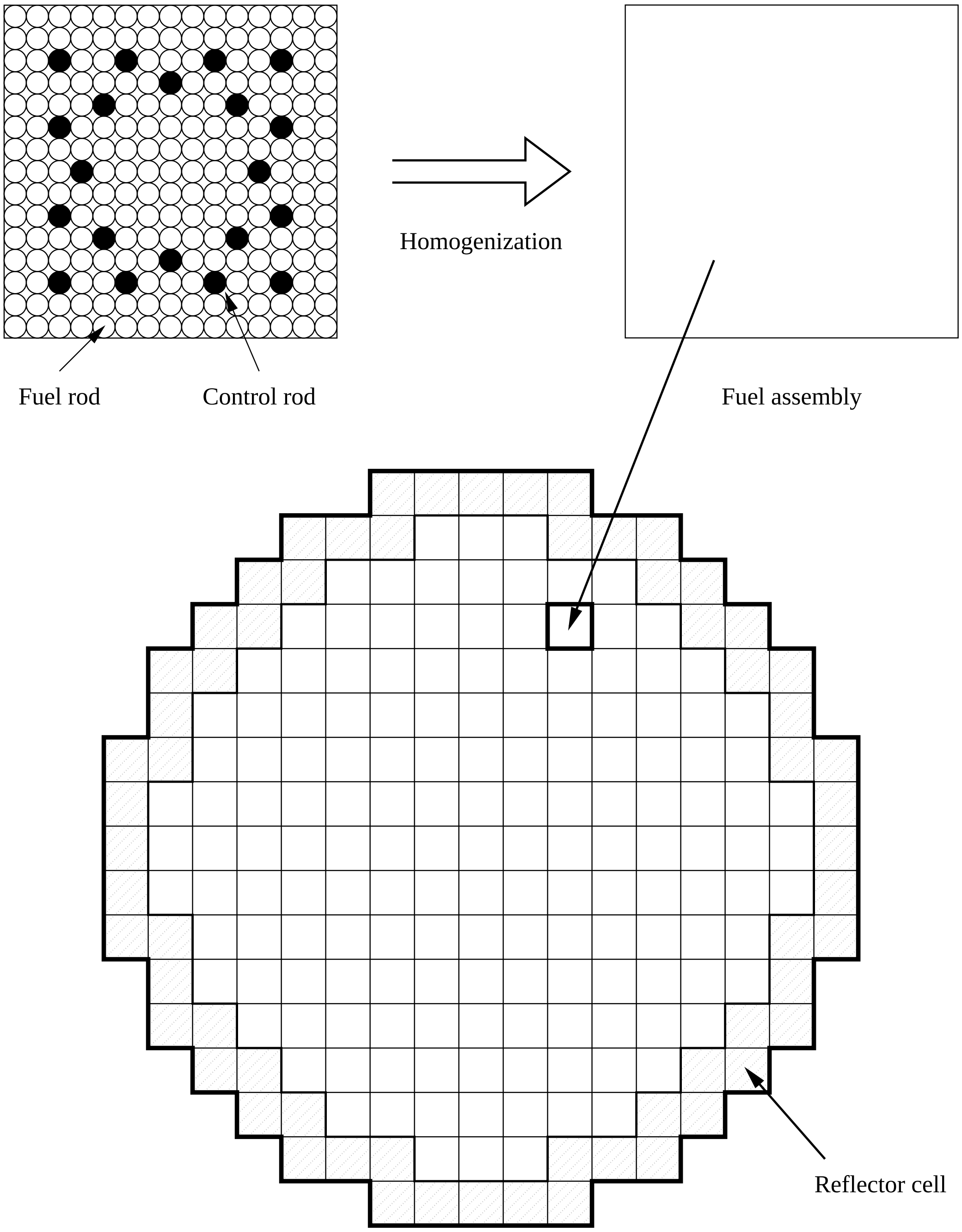


Figure 2.2: Axial cut of the reactor core in a pressurized water reactor PWR

Chapter 3

Discretization of the transient problem

In the process of the discretization of parabolic partial differential equations the first important question is, whether the time or the space is discretized first.
In the *(vertical) line-method*, the space is discretized first. The discretization of the spatial variable leads to a system of *stiff ordinary differential equations (ODE)* in time. For the integration (discretization in time) of such a system of ODEs the theory (including the control of the time step) is well developed (e.g. in Langer [27, ch. 3] or Hairer/Wanner [18]).
In the *Rothe-method* (horizontal line-method) the reverse approach is chosen. The time is discretized first, leaving an elliptic PDE for each time step. The spatial discretization and especially the selection of the grid can be chosen separately in each time step. Therefore the Rothe-Method is a good approach for full-adaptive methods. An example of a full-adaptive Rothe-method for the solution of the multigroup neutron diffusion equations can be found in Wagner [38].
If the grid does not change during the integration, both methods are equivalent and it is not necessary to use the more complex Rothe-method. Therefore the line-method is applied to the multigroup diffusion equations in this dissertation. The discretization of the equations will be outlined in the next sections.
For the spatial discretization a *nodal method* proposed by Finnemann [8], the *nodal expansion method (NEM)* (see section 3.1) is used. The nodal methods appeared in the sixties in relation with the solution of the neutron diffusion equations. On a rectangular grid the discretization is obtained by integrating over the boxes and their surfaces. The NEM can therefore be interpreted as a variant of the *finite volume method (FVM)*. For the time discretization a method, *TR-BDF2* (see section 3.2), proposed by D.J. Rose, et al. [1] that combines the *trapezoidal rule* (TR) and the

second-order *backwards-differentiation formula* (BDF2) is used. It is a second-order *L-stable* method that allows a straightforward control of the time step.

3.1 NEM, a physically induced spatial discretization

Let us recall the system of parabolic PDEs that was modelled in chapter 2, the multigroup neutron diffusion equations (2.26) together with the equations for the precursor concentrations (2.27). For the nodal expansion method, equations (2.26) are transformed in a system of PDEs of first order as in the case of *mixed finite element methods*. The full system can then be written as

$$\left.\begin{array}{r} v_g^{-1}\dfrac{\partial}{\partial t}\,\phi_g(\vec{x},t) + div\ \vec{j}_g(\vec{x},t)\ + \\ \left(\Sigma_{ag}(\vec{x},t) + \displaystyle\sum_{g'=1}^{G}\Sigma_{gg'}(\vec{x},t)\right)\phi_g(\vec{x},t)\ =\ \displaystyle\sum_{i=1}^{I}\chi_{dg}^{i}\lambda_i C_i(\vec{x},t)\ + \\ \displaystyle\sum_{g'=1}^{G}\left(\Sigma_{g'g}(\vec{x},t) + \frac{1}{\lambda_0}\sum_{j=1}^{J}(1-\beta^j)\chi_{pg}^{j}\nu\Sigma_{fg'}^{j}(\vec{x},t)\right)\phi_{g'}(\vec{x},t) \end{array}\right\}\ g = 1...G \tag{3.1}$$

$$\vec{j}_g(\vec{x},t) + D_g(\vec{x},t)\,grad\ \phi_g(\vec{x},t)\ =\ \vec{0},\qquad g = 1...G \tag{3.2}$$

$$\frac{\partial}{\partial t}C_i(\vec{x},t) + \lambda_i C_i(\vec{x},t)\ =\ \frac{1}{\lambda_0}\sum_{j=1}^{J}\beta_i^j\sum_{g=1}^{G}\nu\Sigma_{fg}^{j}(\vec{x},t)\phi_g(\vec{x},t),\qquad i = 1...I \tag{3.3}$$

Equation (3.2) denotes Fick's law (2.9) in the multigroup case. The unknown functions are the neutron flux ϕ_g and the current density $\vec{j}_g$ in each energy group g as well as the precursor concentrations C_i for each precursor type i. A description of all the coefficient functions and parameters of the system can be found in chapter 2.

Since the second-order system (2.26) was not symmetric anyway, this transformation into a system of first order does not cause any further numerical problems. On the contrary the current density $\vec{j}_g$, a quantity of physical interest, is calculated automatically and the boundary conditions (2.19) and (2.20) (see section 2.4) are easier to be fulfilled. The boundary conditions that have to be fulfilled on the interfaces are

$$\phi_g(\vec{x},t)|_{\to\mathbf{\Gamma}_I}\ =\ \phi_g(\vec{x},t)|_{\mathbf{\Gamma}_I\leftarrow}, \tag{3.4}$$

$$\left.\vec{j}_g(\vec{x},t)\right|_{\to\mathbf{\Gamma}_I}\ =\ \left.\vec{j}_g(\vec{x},t)\right|_{\mathbf{\Gamma}_I\leftarrow}. \tag{3.5}$$

On the outer boundaries vacuum boundary conditions are used

$$\vec{j}_{g,in}(\vec{x},t)\Big|_{\mathbf{\Gamma}_V} = \vec{0}. \tag{3.6}$$

The unknown functions ϕ_g, $\vec{j}_g$ and C_i and the coefficient functions are defined on

$$\mathbf{\Omega} \times \mathbf{T}, \text{ with } \mathbf{\Omega} \subset \mathbb{R}^3 \text{ a union of cuboids and } \mathbf{T} := [0,T]$$

Figure 2.2 shows an axial cut of a typical region $\mathbf{\Omega}$ of a nuclear reactor core. The region $\mathbf{\Omega}$ is triangulated with a rectangular grid into evenly sized cuboids (*boxes*) $m = 1...M$. Each one of these boxes together with its surfaces is called a *node*. Often the edges of the triangulation are chosen to coincide with the interfaces between the fuel assemblies (see fig. 2.2). In remark 2.3 it has already been mentioned that for realistic 3D-calculations the coefficient functions, i.e the cross sections $\Sigma_{\cdot g}$, are usually homogenized for each fuel assembly anyway. For the practical calculations it was in fact only possible to rely on such homogenized data. To solve the equations on finer grids, the data had to be interpolated.

Equations (3.1 - 3.3) are discretized using the basic variant of the nodal expansion method proposed by Finnemann [8]: *NEM-M0*.
For the discretization of equation (3.1) a similar approach as with the FVM (compare Langer [26, ch. 5.2]) is chosen. It is integrated over each box m and divided through the volume

$$V^m = a_x^m a_y^m a_z^m$$

of the box. If

$$\begin{aligned}
\phi_g^m(t) &:= \frac{1}{V^m}\int_{\mathbf{\Omega}^m} \phi_g(\vec{x},t)d\vec{x} && g = 1...G\\
C_i^m(t) &:= \frac{1}{V^m}\int_{\mathbf{\Omega}^m} C_i(\vec{x},t)d\vec{x} && i = 1...I\\
j_{gus}^m(t) &:= \frac{1}{A_u^m}\int_{\mathbf{\Gamma}_{us}^m} \vec{j}_g(\vec{x},t)\cdot\vec{e}_u\, dA && g = 1...G;\; u = x,y,z;\; s = l,r^1
\end{aligned}$$

are defined as the zeroth momenta of ϕ_g, C_i and $\vec{j}_g$ on box m and its surfaces respectively, the terms of equation (3.1) can be discretized as follows

- $\displaystyle \frac{1}{V^m}\int_{\mathbf{\Omega}^m} v_g^{-1}\frac{\partial}{\partial t}\phi_g(\vec{x},t)d\vec{x} = v_g^{-1}\frac{\partial}{\partial t}\left(\frac{1}{V^m}\int_{\mathbf{\Omega}^m}\phi_g(\vec{x},t)d\vec{x}\right) = v_g^{-1}\frac{\partial}{\partial t}\phi_g^m(t)$

[1] s ... side, l ... left, r ... right

- $\dfrac{1}{V^m}\displaystyle\int_{\mathbf{\Omega}^m} div\ \vec{j}_g(\vec{x},t)\ d\vec{x} = \dfrac{1}{V^m}\int_{\partial\mathbf{\Omega}^m} \vec{j}_g(\vec{x},t)\cdot\vec{n}\,dA =$
$= \displaystyle\sum_{u=x,y,z} \dfrac{1}{V^m}\left(\int_{\mathbf{\Gamma}^m_{ur}} \vec{j}_g(\vec{x},t)\cdot\vec{e}_u\,dA + \int_{\mathbf{\Gamma}^m_{ul}} \vec{j}_g(\vec{x},t)\cdot(-\vec{e}_u)\,dA\right) =$
$= \displaystyle\sum_{u=x,y,z} \dfrac{1}{a^m_u}\left(j^m_{gur}(t) - j^m_{gul}(t)\right)$

- $\dfrac{1}{V^m}\displaystyle\int_{\mathbf{\Omega}^m} \Sigma_{\cdot g}(\vec{x},t)\phi_g(\vec{x},t)\ d\vec{x} = \Sigma^m_{\cdot g}(t)\left(\dfrac{1}{V^m}\int_{\mathbf{\Omega}^m}\phi_g(\vec{x},t)d\vec{x}\right) = \Sigma^m_{\cdot g}(t)\phi^m_g(t)$
where $\Sigma^m_{\cdot g}(t) = \dfrac{1}{V^m\phi^m_g}\displaystyle\int_{\mathbf{\Omega}^m}\Sigma_{\cdot g}(\vec{x},t)\phi_g(\vec{x},t)\ d\vec{x}$ are the box-homogenized values

The semi-discrete balance equation for each energy-group g and each box m then state

$$
\begin{aligned}
&v_g^{-1}\frac{\partial\phi^m_g}{\partial t}(t) + \sum_{u=x,y,z}\frac{1}{a^m_u}\left(j^m_{gur}(t) - j^m_{gul}(t)\right) + \left(\Sigma^m_{ag}(t) + \sum_{g'=1}^{G}\Sigma^m_{gg'}(t)\right)\phi^m_g(t)\\
&= \sum_{g'=1}^{G}\left(\Sigma^m_{g'g}(t) + \frac{1}{\lambda_0}\sum_{j=1}^{J}(1-\beta^j)\chi^j_{pg}\nu\Sigma^{jm}_{fg'}(t)\right)\phi^m_{g'}(t) + \sum_{i=1}^{I}\chi^i_{dg}\lambda_i C^m_i(t)
\end{aligned}
\tag{3.7}
$$

Analogously the semi-discrete form of the equations for the precursor concentrations (see equation (3.13) below) can be found by integrating equations (3.3) over each box.

For the discretization of equation (3.2), it is integrated over the six surfaces ($u = x, y, z;\ s = l, r$) of a box m and divided through the area A^m_u. The first term is discretized in the same way as in (3.7)

- $\dfrac{1}{A^m_u}\displaystyle\int_{\mathbf{\Gamma}^m_{us}} \vec{j}_g(\vec{x},t)\cdot\vec{e}_u\,dA = j^m_{gus}(t)$

For the second term an approximation of the normal derivative $\frac{\partial\phi_g}{\partial\vec{n}}$ on the surface is needed. In the FVM this term is approximated with the central difference quotient between the box centre and the centre of the neighbouring box (Langer [26, p. 152f]). Since the diffusion constant D_g has in general jumps at surfaces that coincide with interfaces between different fuel assemblies, it is better to choose another way of approximating the normal derivative, not involving neighbouring values. In the

NEM-M0 method the neutron flux ϕ_g is expanded into a quadratic polynomial for each direction ($u = x, y, z$). At each side ($s = l, r$) the average surface flux

$$\psi^m_{gus}(t) := \frac{1}{A^m_u} \int_0^{a^m_y} \int_0^{a^m_z} \phi_g\left((x^m_s, y, z), t\right)\, dy\, dz \tag{3.8}$$

is used for the approximation. In the centre the zeroth momentum ϕ^m_g is used. The quadratic approximation of ϕ_g on the unit-interval $\tilde{u} \in [0, 1]$ is then given by

$$\psi^m_{gu}(\tilde{u}, t) := \phi^m_g(t) h_0 + \frac{\psi^m_{gur}(t) - \psi^m_{gul}(t)}{2} h_1(\tilde{u}) + \left(\phi^m_g(t) - \frac{\psi^m_{gur}(t) + \psi^m_{gul}(t)}{2}\right) h_2(\tilde{u})$$

where

$$\left.\begin{aligned} h_0(u) &= 1 \\ h_1(u) &= 2u - 1 \\ h_2(u) &= 6u(1-u) - 1 \end{aligned}\right\} \quad u \in [0, 1]$$

For each surface $\mathbf{\Gamma}^m_{us}$ the normal derivative in the second term of equation (3.2) can then be approximated with the derivative of $\psi^m_{gu}(\tilde{u})$ at $\tilde{u} = 0$ (for $s = l$) and $\tilde{u} = 1$ (for $s = r$):

- $$\frac{1}{A^m_u} \int_{\mathbf{\Gamma}^m_{us}} D_g(\vec{x}, t)\, grad\ \phi_g(\vec{x}, t) \cdot \vec{e}_u\, dA = D^m_g(t) \frac{1}{A^m_u} \int_{\mathbf{\Gamma}^m_{us}} \frac{\partial \phi_g}{\partial u}(t) dA \approx$$
  $$\approx D^m_g(t) \left(\frac{\partial}{\partial u} \psi^m_{gu}\Big(\underbrace{\frac{u - u^m_l}{a^m_u}}_{=:\tilde{u}}\Big)\right)\Bigg|_s \underbrace{\frac{1}{A^m_u} \int_{\mathbf{\Gamma}^m_{us}} dA}_{=1} = \underbrace{D^m_g(t) \frac{1}{a^m_u}}_{=:D^m_{gu}(t)} \frac{\partial \psi^m_{gu}(\tilde{u})}{\partial \tilde{u}}\Bigg|_s$$

In the semi-discrete form, Fick's law (3.2) changes to a system of six algebraic equations

$$j^m_{gus}(t) + D^m_{gu}(t) \left.\frac{\partial \psi^m_{gu}(\tilde{u})}{\partial \tilde{u}}\right|_s = 0 \qquad u = x, y, z;\ s = l, r \tag{3.9}$$

for the current-densities in each energy-group g and each box m. But new unknown functions ψ^m_{gus} have been introduced.
Recalling equations (2.7) and (2.8) that were used in section 2.3.1 to find Fick's law, these average surface fluxes can be related to the current across the surface. Adding equations (2.7) and (2.8) the "up"-ward and the "down"-ward current, j_{u-} and j_{u+}, fulfill $(j_{u-} + j_{u+})\, dA = (\phi/2|_{u=0}\, dA$ for $u = x, y, z$. If *incoming* and *outgoing currents*[2] j^{m+}_{gus} and j^{m-}_{gus} are used instead of the current density (or surplus current)

[2]The + or − in the notation indicates that the partial current is in positive or negative coordinate direction respectively. j^{m+}_{gul} and j^{m-}_{gur} are therefore incoming, j^{m-}_{gul} and j^{m+}_{gur} are outgoing currents.

j^m_{gus} in the discretization, an approximation for the unknown functions can be found from this relation:

$$\psi^m_{gus}(t) = 2\left(j^{m+}_{gus}(t) + j^{m-}_{gus}(t)\right) \tag{3.10}$$

Another advantage of incoming and outgoing currents is that on the outer boundary the homogeneous boundary condition (3.6) is easily fulfilled (see below). In the semi-discrete equations (3.7) and (3.9) j^m_{gus} simply has to be replaced by

$$j^m_{gus}(t) = j^{m+}_{gus}(t) - j^{m-}_{gus}(t)$$

The current-density is substituted, the derivatives of ψ^m_{gu} in equation (3.9) are calculated and then a simple transformation of the system (3.7, 3.9) is carried out (see remark 3.1). As a result the semi-discrete neutron balance equations, the semi-discrete equations for the currents and the semi-discrete equations for the precursor concentrations, in the form that will be used in the numerical calculations, are established: For each box $m = 1...M$ and for $t \in [0,T]$

$$\left.\begin{aligned} v_g^{-1}\frac{\partial \phi^m_g}{\partial t}(t) = &-\left(\Sigma^m_{ag}(t) + \sum_{g'=1}^{G}\Sigma^m_{gg'}(t) + \sum_{u=x,y,z}\frac{2c^m_{1gu}(t)}{a^m_u}\right)\phi^m_g(t) + \\ &\sum_{g'=1}^{G}\left(\Sigma^m_{g'g}(t) + \frac{1}{\lambda_0}\sum_{j=1}^{J}(1-\beta^j)\chi^j_{pg}\nu\Sigma^{jm}_{fg'}(t)\right)\phi^m_{g'}(t) + \\ &\sum_{u=x,y,z}\frac{4c^m_{1gu}(t)}{a^m_u}\left(j^{m-}_{gur}(t) + j^{m+}_{gul}(t)\right) + \sum_{i=1}^{I}\chi^i_{dg}\lambda_i C^m_i(t) \end{aligned}\right\} g = 1...G \tag{3.11}$$

$$\left.\begin{aligned} j^{m+}_{gur}(t) - c^m_{1gu}(t)\phi^m_g(t) + c^m_{2gu}(t)j^{m-}_{gur}(t) + c^m_{3gu}(t)j^{m+}_{gul}(t) &= 0 \\ j^{m-}_{gul}(t) - c^m_{1gu}(t)\phi^m_g(t) + c^m_{3gu}(t)j^{m-}_{gur}(t) + c^m_{2gu}(t)j^{m+}_{gul}(t) &= 0 \end{aligned}\right\}\left\{\begin{aligned} g &= 1...G \\ u &= x,y,z \end{aligned}\right. \tag{3.12}$$

$$\frac{\partial C^m_i}{\partial t}(t) = -\lambda_i C^m_i(t) + \frac{1}{\lambda_0}\sum_{j=1}^{J}\beta^j_i\sum_{g=1}^{G}\nu\Sigma^{jm}_{fg}(t)\phi^m_g(t) \qquad i = 1...I \tag{3.13}$$

where

$$c^m_{1gu}(t) := \frac{6D^m_{gu}(t)}{12D^m_{gu}(t)+1}, \quad c^m_{2gu}(t) := 1 - 4c^m_{1gu}(t) - c^m_{3gu}(t), \quad c^m_{3gu}(t) := \frac{-4c^m_{1gu}(t)}{12D^m_{gu}(t)+3}$$

The boundary and interface conditions (3.4 - 3.6) can easily be fulfilled now. The interface condition (3.4) is not needed, since the ϕ^m_g are defined in the interior of the boxes. On the interfaces only the currents can be required to be continuous (3.5). Therefore the incoming currents of one box equal the outgoing currents of the neighbouring box (see figure 3.1):

$$\left.\begin{aligned} j_{gur}^{m_1-}(t) &= j_{gul}^{m_2-}(t) \\ j_{gul}^{m_2+}(t) &= j_{gur}^{m_1+}(t) \end{aligned}\right\} \quad g = 1...G;\ u = x, y, z \tag{3.14}$$

Since the incoming currents vanish on the outer boundaries by condition (3.6), they can be completely omitted and substituted by the outgoing currents of the neighbouring box. The fact that only outgoing currents are used also supports the approach that was chosen for the approximation of the normal derivative $\frac{\partial \phi_g}{\partial \vec{n}}$ earlier on in the discretization of equation (3.2)
The result is a large system of ODEs for the average box fluxes ϕ_g^m and the precursor concentrations C_i^m in the boxes together with algebraic equations for the outgoing currents j_{gul}^{m-} and j_{gur}^{m+} across the box-surfaces. The integration of these ODEs is the last step in the discretization process.

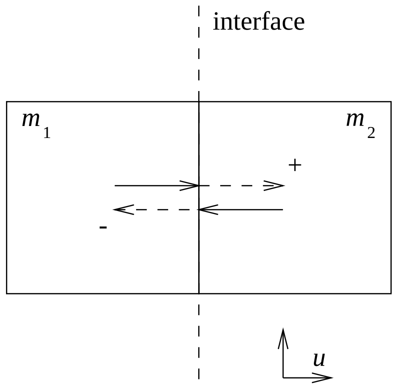


Figure 3.1: Continuity of the currents on the interfaces

Remark 3.1 *The transformation of the system that was performed as the final step of the discretization process reduces the coupling between the balance equations (3.11) and the equations for the currents (3.12) in one box to a minimum. At the end of the next section 3.2 the "local" block of the matrix of the arising linear equation system for one box is presented. Due to the transformation it is almost triangular. This is important for the construction of an effective* smoothing *and* preconditioning method *of* Block-SOR *type (see section 4.1).*

Remark 3.2 *In actual practice heuristic ansatz-functions of 4th order are often used for ψ^m_{gu} to improve the quality of the approximation. The two extra coefficients in this ansatz are determined via solving special one-dimensional diffusion equations (in variational form). This method is called* NEM-M2 *(for a detailed description see Finnemann, et al. [8]). In PANBOX2 [3] both methods, NEM-M0 and NEM-M2, are implemented.*

3.2 Time discretization of the semi-discrete equations with the composite TR-BDF2 method

In the discretization process of parabolic equations with the line-method the second step is the integration of the semi-discrete equations with respect to the time. In the introduction to this chapter it has already been pointed out that the spatial discretization of parabolic equations always leads to stiff ordinary differential equations. Moreover the underlying physics (see section 2.1) for the particular neutronics problem shows that there is a striking difference in the time scales between the production of prompt and delayed neutrons. This fact still causes an increase in the stiffness of the differential equations. It is therefore absolutely necessary to use stable methods for the integration.

An integration scheme is called *A-stable*, if it has no growing solutions when it is applied to the test problem $\dot{y} = \lambda y$ with $Re\,\lambda < 0$. Unfortunately A-stable implicit linear multistep methods can only be of order less or equal than two. A stronger stability condition that implies the A-stability is the *L-stability*. An integration scheme is called L-stable, if it dampens the approximate solution of $\dot{y} = \lambda y$ with the stronger condition $lim\,(Re\,\lambda h) \to -\infty$ (h denotes the time step). For an exact definition see Langer [27, ch. 3] or Hairer/Wanner [18].

An easy L-stable method is the *implicit backward-Euler* method and it was used as a first approach (see also the results in section 6.4). It is nevertheless only of first order and allows therefore only small time steps. The second-order A-stable implicit multistep method that allows the largest time steps is the *trapezoidal rule* (or *Crank-Nicholson method*), because it has the smallest truncation error. But unfortunately the trapezoidal rule (TR) is not L-stable. The second-order *backward-differentiation formula* (BDF2) on the other hand is L-stable and even *B-stable*[3]. Compared to TR, however, BDF2 has a higher truncation error and is more difficult to implement since it involves three consecutive solution values.

The composition of the TR and the BDF2 schemes results in a linear multistep method TR-BDF2 (Rose, et al. [1]) that combines the desirable properties of both

[3] B-stability is a stability condition for highly nonlinear stiff problems. It is stronger than the L-stability.

methods. In addition it allows a straightforward control of the time step. In practical applications this is very important, since the quality of an integration scheme depends highly on the quality of the scheme that determines the next time step.

3.2.1 Definition of the TR-BDF2 method

In the following a general *initial-value problem* for a system of ODEs of first order will be considered. The classical formulation for such a problem is

$$\begin{aligned} &u'(t) = f(t, u(t)), \qquad t \in [0, T], \\ &\text{IC: } \; u(0) = u_0 \end{aligned} \tag{3.15}$$

with $u_0 \in \mathbb{R}^N$ and $f : D \subseteq ([0,T] \times \mathbb{R}^N) \mapsto \mathbb{R}^N \in C^0(D)$ for the unknown function $u : [0,T] \mapsto \mathbb{R}^N \in C^1([0,T])$.
For the integration of such a system the interval $[0,T]$ is divided into m parts $[t_{j-1}, t_j]$, $j = 1...m$ with $0 = t_0 < t_1 < ... < t_J = T$. The t_j represent the discrete time-points, the $h_j := t_{j+1} - t_j$ represent the time steps and the vectors u_j represent the approximations of the solution at the discrete moments.

One step of the implicit trapezoidal rule applied to such a problem says

$$u_{j+1} = u_j + \frac{h_j}{2}\left(f(t_j, u_j) + f(t_{j+1}, u_{j+1})\right) \tag{3.16}$$

One step of the second-order backwards-differentiation formula says

$$\alpha_2 u_{j+2} = \alpha_1 u_{j+1} - \alpha_0 u_j + f(t_{j+2}, u_{j+2}) \tag{3.17}$$

with

$$\alpha_0 = \frac{h_{j+1}}{h_j(h_j + h_{j+1})}, \qquad \alpha_1 = \frac{h_j + h_{j+1}}{h_j h_{j+1}}, \qquad \alpha_2 = \frac{h_j + 2h_{j+1}}{h_{j+1}(h_j + h_{j+1})}$$

The TR-BDF2 method (Rose, et al. [1]) is now defined as a combination of the two previous methods. The time step h_j is divided into two parts

$$h_{j,1} = \gamma h_j, \qquad h_{j,2} = (1-\gamma) h_j \qquad \gamma \in [0,1].$$

The first time step is calculated with the trapezoidal rule, the second with the BDF2 method:

$$\begin{aligned} &\text{1. step}: \quad & u_{j+\gamma} &= u_j + \frac{\gamma h_j}{2}\left(f(t_j, u_j) + f(t_j + \gamma h_j, u_{j+\gamma})\right) \\ &\text{2. step}: \quad & \alpha_2 u_{j+1} &= \alpha_1 u_{j+\gamma} - \alpha_0 u_j + f(t_{j+1}, u_{j+1}) \end{aligned} \tag{3.18}$$

$$\alpha_0 = \tfrac{(1-\gamma)}{\gamma} h_j^{-1}, \quad \alpha_1 = \tfrac{1}{\gamma(1-\gamma)} h_j^{-1}, \quad \alpha_2 = \tfrac{(2-\gamma)}{(1-\gamma)} h_j^{-1}$$

The result is again a special 3-stage implicit Runge-Kutta method symbolized by the following table

$$\begin{array}{c|ccc} 0 & 0 & 0 & 0 \\ \gamma & \frac{\gamma}{2} & \frac{\gamma}{2} & 0 \\ 1 & \frac{1}{2(2-\gamma)} & \frac{1}{2(2-\gamma)} & \frac{(1-\gamma)}{(2-\gamma)} \\ \hline & \frac{1}{2(2-\gamma)} & \frac{1}{2(2-\gamma)} & \frac{(1-\gamma)}{(2-\gamma)} \end{array}$$

with the stability function

$$R(z) = \frac{2(2-\gamma) + (\gamma^2 - 2\gamma + 2)\, z}{2(2-\gamma) + (\gamma^2 - 2)\, z + (\gamma - \gamma^2)\, z^2}$$

It can easily be shown that this method is again of second order and L-stable. It is not B-stable anymore. The method however can be easily implemented and unlike some other second-order L-stable methods, the presence of the first TR-step produces a self-starting method. A final advantage lies in the fact that the Jacobian matrices for both steps have the same form provided $\gamma = 2 - \sqrt{2}$, which can be beneficial with Newton-like techniques for nonlinear problems.
For the actual calculations, a scheme is needed that determines the next time step. Therefore an error estimate for the *local truncation error* has to be developed. For the trapezoidal step followed by the backward-differentiation step the *principal error term* of the local truncation error is given by

$$\tau := \Big(u(t_{j+1}) - u_{j+1}\Big) + O(h_j^4) = C h_j^3 f^{(2)} \tag{3.19}$$

where the constant C is given by

$$C = \frac{3\gamma^2 - 4\gamma + 2}{12(2-\gamma)}$$

(see Rose, et al. [1]). The error involves the second derivative $f^{(2)}$ of the right-hand side f of the continuous problem (3.15). It can be obtained from a divided difference of the function values,

$$h_j^2 f^{(2)} = 2\Big[(1-\gamma)^{-1} f(t_{j+1}, u_{j+1}) - \gamma^{-1}(1-\gamma)^{-1} f(t_j + \gamma h_j, u_{j+\gamma}) + \gamma^{-1} f(t_j, u_j)\Big]$$

Using (3.18) and (3.19) the error τ is found to be

$$\tau = \frac{3\gamma^2 - 4\gamma + 2}{6(1-\gamma)^2}\left[u_{j+1} - \frac{1}{\gamma^2}\, u_{j+\gamma} + \frac{1-\gamma^2}{\gamma^2}\, u_j + h_j\, \frac{1-\gamma}{\gamma}\, f(t_j, u_j)\right] \tag{3.20}$$

If the tolerance for the norm of the relative error $\|\tau_R\|_2$ is ε_R, the "optimal" step-size, h_{opt}, can be estimated by

$$h_{opt} = h_j\, \sqrt[3]{\frac{\varepsilon_R}{\|\tau_R\|_2}} \tag{3.21}$$

The relative error τ_R is calculated from

$$(\tau_R)_i = \frac{|\tau_i|}{|(u_{j+1})_i| + \delta}, \qquad i = 1...N \tag{3.22}$$

where δ depends on the problem and represents a lower limit for the values of u_{j+1}. The norm of the relative error $\|\tau_R\|_2$ can be interpreted as a special weighted norm $\|.\|_W$[4] of the absolute error τ. It can be defined as

$$\|\Delta u\|_W := \|\mathbf{W}\Delta u\|_2, \quad \text{with } \mathbf{W} := (w_{ii}),\ w_{ii} := \frac{1}{|u_i| + \delta} \tag{3.23}$$

a diagonal matrix.

If the error $\|\tau_R\|_2 \geq \varepsilon_R$, the step $j \to j+1$ is accepted and the next time step is chosen as $h_{j+1} = h_{opt}$. Otherwise the time step is rejected and the calculation is started again with time step $h_j = h_{opt}$.

For practical calculations it is often better not to reject any time step, but always continue with the next one. If ε_R is made a bit smaller, the error would still be acceptable, but no extra time step would have to be calculated. To avoid slightly optimistic values, the step-size h_{opt} is usually reduced by a multiplicative factor fac (e.g. $fac = 0.9$). Furthermore the step-size is constrained to a maximum increase of a factor fac_{max} (e.g. $fac_{max} = 2$), to keep it from increasing too rapidly. Finally there can be breakpoints that have to be exactly sampled (e.g. the coupling to the thermo-hydraulics in the presented problem). If t_B is the next breakpoint that occurs after the current time t_{j+1},

$$h_{j+1} = \frac{t_B - t_{j+1}}{\left\lceil \frac{t_B - t_{j+1}}{h_{opt}} \right\rceil}$$

is used to get appropriate step-sizes to arrive at t_B with low overhead. $\lceil\ \rceil$ denotes the ceiling function.

Remark 3.3 *The TR-BDF2 method that was described above can of course be generalized. Usually it is not necessary to apply an L-stable method in every single time step. It has already been mentioned that the trapezoidal rule allows for the largest step-sizes of all second-order A-stable methods. Therefore a more general method would use the trapezoidal rule on its own and introduce BDF steps, only if stability problems occur.*

[4]This norm will be needed later on for a convergence test of the solution methods.

3.2.2 Application to the neutron diffusion equations

To apply the integration scheme to the semi-discrete multigroup neutron diffusion equations (3.11) and the semi-discrete equations for the precursor concentrations (3.13) the notation for the problem is simplified slightly. The index m for the box number is dropped and the system is written as

$$\frac{\partial \phi_g}{\partial t}(t) = v_g \left[f_g\Big(t, \phi_{g'}, j^{\pm}_{gus}\Big) + h\Big((1-\beta^j)\chi^j_{pg}\,;\, t, \phi_{g'}\Big) + \sum_i \chi^i_{dg} \lambda_i C_i(t) \right] \tag{3.24}$$

$$\frac{\partial C_i}{\partial t}(t) = -\lambda_i C_i(t) + h\Big(\beta^j_i\,;\, t, \phi_{g'}\Big) \tag{3.25}$$

with

$$f_g\Big(t, \phi_{g'}, j^{\pm}_{gus}\Big) := -\underbrace{\hat{\sigma}_g(t)}_{\text{see eq. (3.11)}} \phi_g(t) + \sum_{g'} \Sigma_{g'g}(t)\phi_{g'}(t) + \sum_u \underbrace{\hat{c}_{gu}(t)}_{\text{see eq. (3.11)}} \Big(j^-_{gur}(t) + j^+_{gul}(t)\Big)$$

$$h\Big(\kappa\,;\, t, \phi_{g'}\Big) := \sum_{g'} \left(\frac{1}{\lambda_0} \sum_j \kappa^j\, \nu\Sigma^j_{fg'}(t) \right) \phi_{g'}(t)$$

At first equations (3.25) are integrated using the abbreviations $t^n := t_{j+n}$, $\phi^n_g := \phi_g(t_{j+n})$ and $C^n_i = C_i(t_{j+n})$. The trapezoidal step and the BDF step lead to the following explicit equations for C^γ_i and C^1_i respectively:

$$C^\gamma_i = \frac{2-\lambda_i\gamma h}{2+\lambda_i\gamma h}\, C^0_i + h\Big(\frac{\lambda_i\gamma h}{2+\lambda_i\gamma h}\,\beta^j_i\,;\, t^0, \phi^0_{g'}\Big) + h\Big(\frac{\lambda_i\gamma h}{2+\lambda_i\gamma h}\,\beta^j_i\,;\, t^\gamma, \phi^\gamma_{g'}\Big) \tag{3.26}$$

$$C^1_i = \frac{\alpha_1}{\alpha_2+\lambda_i}\, C^\gamma_i - \frac{\alpha_0}{\alpha_2+\lambda_i}\, C^0_i + h\Big(\frac{1}{\alpha_2+\lambda_i}\,\beta^j_i\,;\, t^1, \phi^1_{g'}\Big) \tag{3.27}$$

Next equations (3.24) are integrated with the TR-BDF2 method using the same abbreviations as above and C^γ_i and C^1_i are substituted from equations (3.26) and (3.27). The result is a system of implicit equations for the time-discrete fluxes ϕ^γ_g and ϕ^1_g at $t = t^\gamma$ and $t = t^1$ respectively:

$$\begin{gathered} \phi^\gamma_g - \frac{\gamma h}{2} v_g \left[f_g\Big(t^\gamma, \phi^\gamma_{g'}, j^{\pm\,\gamma}_{gus}\Big) + h\Big(\hat{\chi}^{Trap}_g\,;\, t^\gamma, \phi^\gamma_{g'}\Big) \right] = \\ \phi^0_g + \frac{\gamma h}{2} v_g \left[f_g\Big(t^0, \phi^0_{g'}, j^{\pm\,0}_{gus}\Big) + h\Big(\hat{\chi}^{Trap}_g\,;\, t^0, \phi^0_{g'}\Big) \right] + v_g \sum_i \chi^i_{dg} \frac{2\lambda_i\gamma h}{2+\lambda_i\gamma h}\, C^0_i \end{gathered} \tag{3.28}$$

$$\begin{gathered} \alpha_2\, \phi^1_g + v_g \left[f_g\Big(t^1, \phi^1_{g'}, j^{\pm\,1}_{gus}\Big) + h\Big(\hat{\chi}^{BDF}_g\,;\, t^1, \phi^1_{g'}\Big) \right] = \\ \alpha_1 \left[\phi^\gamma_g + v_g \sum_i \chi^i_{dg} \frac{\lambda_i}{\alpha_2+\lambda_i}\, C^\gamma_i \right] - \alpha_0 \left[\phi^0_g + v_g \sum_i \chi^i_{dg} \frac{\lambda_i}{\alpha_2+\lambda_i}\, C^0_i \right] \end{gathered} \tag{3.29}$$

with

$$\hat{\chi}_g^{Trap,j} = \chi_{pg}^j(1-\beta^j) + \sum_i \chi_{dg}^i \frac{\lambda_i \gamma h}{2+\lambda_i \gamma h}\, \beta_i^j$$
$$\hat{\chi}_g^{BDF,j} = \chi_{pg}^j(1-\beta^j) + \sum_i \chi_{dg}^i \frac{\lambda_i}{\alpha_2+\lambda_i}\, \beta_i^j$$

The equations for the currents (3.12), have to be solved simultaneously at each time step t_j and $t_{j+\gamma}$. The result is a *linear equation system* for the fluxes ϕ_g^m and the currents $j_{gus}^{m\pm}$ in each box m in each partial time step. Since there are only $N_m = G(1+2Dim)$ equations for $G(1+4Dim)$ unknowns per box, the equation systems are under-determined. Recalling the continuity condition (3.14) for the currents of adjacent boxes on the interface and using only outgoing currents j_{gul}^{m-} and j_{gur}^{m+}, a coupling between the linear equation systems for the different boxes is introduced and the systems become regular.
The resulting *matrices* are sparse nonsymmetric indefinite $N \times N$ - matrices, with

$$A \cdot \underline{u} := \begin{pmatrix} \underbrace{\begin{array}{cc|ccccc} * & * & & & 0 & & \\ * & * & & & & & \\ \hline * & 0 & 1 & 0 & \cdots & \cdots & 0 \\ 0 & * & 0 & 1 & \ddots & & \vdots \\ \vdots & & \vdots & \ddots & \ddots & \ddots & \vdots \\ * & 0 & \vdots & & \ddots & 1 & 0 \\ 0 & * & 0 & \cdots & \cdots & 0 & 1 \end{array}}_{A_{11}} & & \begin{array}{l}\text{Coupling via} \\ \text{the outgoing} \\ \text{currents of} \\ \text{adjacent boxes}\end{array} \\ & \bullet\;\bullet\;\bullet & \\ \begin{array}{l}\text{Coupling via} \\ \text{the outgoing} \\ \text{currents of} \\ \text{adjacent boxes}\end{array} & & \underbrace{\begin{array}{cc|ccccc} * & * & & & 0 & & \\ * & * & & & & & \\ \hline * & 0 & 1 & 0 & \cdots & \cdots & 0 \\ 0 & * & 0 & 1 & \ddots & & \vdots \\ \vdots & & \vdots & \ddots & \ddots & \ddots & \vdots \\ * & 0 & \vdots & & \ddots & 1 & 0 \\ 0 & * & 0 & \cdots & \cdots & 0 & 1 \end{array}}_{A_{MM}} \end{pmatrix} \bullet \begin{pmatrix} \phi_1^1 \\ \phi_2^1 \\ j_{1xl}^{1-} \\ j_{2xl}^{1-} \\ \vdots \\ j_{1zr}^{1+} \\ j_{2zr}^{1+} \\ \bullet \\ \bullet \\ \bullet \\ \phi_1^M \\ \phi_2^M \\ j_{1xl}^{M-} \\ j_{2xl}^{M-} \\ \vdots \\ j_{1zr}^{M+} \\ j_{2zr}^{M+} \end{pmatrix} = \begin{pmatrix} * \\ * \\ 0 \\ 0 \\ \vdots \\ 0 \\ 0 \\ \bullet \\ \bullet \\ \bullet \\ * \\ * \\ 0 \\ 0 \\ \vdots \\ 0 \\ 0 \end{pmatrix} =: \underline{b}$$

Figure 3.2: Pattern of the sparse linear equation systems[5]

[5]In comparison to the rest of this section the upper index for ϕ_g^m and $j_{gus}^{m\pm}$ denotes the box-number as in section 3.1 and not the time step

$N = M \cdot N_m$. The pattern of each system in the case of two energy groups, i.e. $G = 2$, is shown in figure 3.2. This particular block structure is important for the construction of an effective block solution method in section 4.1 of the next chapter. Typical values for N in practical problems vary between 10^5 and 10^7.

The solution of these linear equation systems is the main concern of the dissertation. The next chapter will show the development of suitable algorithms for such linear equation systems.

Remark 3.4 *It has already been mentioned in section 2.6 that the start solution* $\underline{u}^0$ *for the first time step can be found by performing a quasi-stationary calculation at the beginning. Since it is not the primary concern of the dissertation, this problem is discussed only briefly in Appendix A.*

Chapter 4

Solution of large, structured, sparse, nonsymmetric linear equation systems

Two factors have to be considered for an effective solution of *large, structured, sparse linear equation systems* (LESs)

$$A \cdot \underline{u} = \underline{b} \tag{4.1}$$

as they arise for example in the discretization process of partial differential equations (PDEs). The first and most important factor is the *number* $\mathbf{Q}$ *of arithmetic operations* (FLOPs) that have to be performed to solve the LES, or more generally the CPU-time. The second one is the memory $\mathbf{M}$ that is needed. For an optimal method both the number of FLOPs and the memory should grow linearly with the number of unknowns N, i.e. $\mathbf{Q} = O(N)$ and $\mathbf{M} = O(N)$ (see Langer [26, ch. 6.1]).
If the LESs arise from PDEs in 3D, *standard direct methods* (Golub, Van Loan [14, ch. 3-4] or Jung, Langer [22, ch. 5.1], e.g. Gauss, LU - decomposition, ...) are generally inefficient. Their advantage is that they are very robust, but the number of FLOPs and the memory are too high. There are only very specific problems concerning simple geometries that can be solved quickly with direct methods (e.g. FFT, ...).
Standard iterative methods (Saad [31], Hackbusch [17, ch. 3-9] or Jung, Langer [22, ch. 5.2], e.g. Jacobi, Gauss-Seidel, SOR, Block variants, Conjugate Gradient (CG) like methods, ...) are optimal regarding the memory, i.e. $\mathbf{M} = O(N)$. The number of FLOPs per iteration is $O(N)$, but the method does not necessarily have to converge. If the method converges, the number of iterations $\mathbf{I}(\varepsilon)$ to attain a fixed relative accuracy ε depends on the *condition* of the matrix A of the LES. In general the condition of A becomes worse, when the number of unknowns grows. The total

number of FLOPs (i.e. $\mathbf{Q}(\varepsilon) = \mathbf{I}(\varepsilon) * \text{FLOPs per iteration}$), necessary to attain an accuracy ε, is therefore not optimal.
One way of overcoming this problem is *preconditioning* (Saad [31] or Langer [26, ch. 6.2]). A matrix $P = P_L P_R$ is chosen in a way that the preconditioning systems $P_L \cdot \underline{w} = \underline{d}$ and $P_R \cdot \underline{w} = \underline{d}$ can be solved quickly ($O(N)$ operations) and that the condition of the new iteration matrix $P_L^{-1} A P_R^{-1}$ is significantly better than the condition of A. Especially in connection with *Krylov-Subspace methods*, e.g. CG-like methods, preconditioning is very effective (Saad [31] or Jung, Langer [22, 5.2.3] and [21]).
Another way of reducing the number of FLOPs is the application of *Multi-Grid Techniques* (see Langer [28] or Hackbusch [16], [17, ch. 10]). Standard iterative methods, e.g. Jacobi, SOR,..., reduce the high frequencies in the error very well, i.e. they are *smoothing* the error. To treat the low frequencies a coarser grid is introduced. The smoothed residual is restricted to the coarser grid, where the same procedure is applied recursively (*Coarse Grid Correction*). The LES on the coarsest grid is smaller than the original system and can be solved by any direct or iterative method.

In the following sections both techniques are applied to the LESs arising from the discretization of the multigroup diffusion equations as developed in the previous chapters.
In the first section a *Block-SOR method* is developed that takes the particular structure of the LESs into account. It is a very important component, as it can be used both as an effective preconditioning step for CG-like methods and as an effective smoothing step for Multi-Grid methods.
In the second section a preconditioned Krylov-Subspace method is used for the solution of the LESs. Since the LESs are *nonsymmetric*, special variants of the CG method are needed. A very effective and robustly converging method is *Bi-CGStab* by H.A. Van der Vorst [35].
In the main section of this chapter a Multi-Grid algorithm is defined and the components are described. The Block-SOR method from the first section is used for the smoothing step and the preconditioned Bi-CGStab method from the second section is used for the solution of the coarse-grid system. The *prolongation* and the *restriction* are mixed linear and constant (compare Wesseling [39, ch. 5]).
In practice convergence tests are needed to decide whether a solution of an iterative method is accurate enough or not. They are described in the final section.

Remark 4.1 *The solution of the generalized eigenvalue problems in the stationary case (which is described in Appendix A) leads to regular LESs with the same structure as in the transient case. Therefore the same methods can be applied.*

4.1 The Block-SOR method

4.1.1 Relaxation schemes: Jacobi, Gauss-Seidel and SOR

A basic approach to solve a linear equation system (4.1) iteratively is to annihilate the components of the residual vector one or a few at a time and in a certain order. These methods are called *relaxation schemes.* At the beginning the matrix A is decomposed into

$$A := L + D + R, \tag{4.2}$$

where D is the diagonal of A, L its strict lower part and R its strict upper part, i.e.

$$D := \begin{pmatrix} a_{11} & & \\ & \ddots & \\ & & a_{NN} \end{pmatrix}, \; L := \begin{pmatrix} 0 & & & \\ a_{21} & 0 & & \\ \vdots & \ddots & \ddots & \\ a_{N1} & \cdots & a_{NN-1} & 0 \end{pmatrix}, \; R := \begin{pmatrix} 0 & a_{12} & \cdots & a_{1N} \\ & 0 & \ddots & \vdots \\ & & \ddots & a_{N-1N} \\ & & & 0 \end{pmatrix}$$

The *Jacobi iteration* determines the i-th component of the next approximation $\underline{u}^{(k+1)}$ so as to annihilate the i-th component of the residual vector

$$(\underline{b} - A\underline{u}^{(k+1)})_i = 0 \tag{4.3}$$

Thus the component-wise form of the Jacobi iteration reads as

$$u_i^{(k+1)} := a_{ii}^{-1} \left(b_i - \sum_{j \neq i} a_{ij} u_j^{(k)} \right) \qquad i = 1, ..., N \tag{4.4}$$

The Jacobi iteration (4.4) can be rewritten in vector form using the above decomposition of A as

$$\underline{u}^{(k+1)} := D^{-1} \left[\underline{b} - (L + R)\underline{u}^{(k)} \right] \tag{4.5}$$

Similarly, the *(forward) Gauss-Seidel iteration* corrects the i-th component of the current approximate solution, in the order $i = 1, 2, .., N$, again to annihilate the i-th component of the residual. However, this time the approximate solution is updated immediately after the component is determined. This leads to the following component-wise Gauss-Seidel iteration scheme

$$u_i^{(k+1)} := a_{ii}^{-1} \left(b_i - \sum_{j=1}^{i-1} a_{ij} u_j^{(k+1)} - \sum_{j=i+1}^{N} a_{ij} u_j^{(k)} \right) \qquad i = 1, ..., N \tag{4.6}$$

and the vector form

$$\underline{u}^{(k+1)} := (D + L)^{-1} \left[\underline{b} - R\underline{u}^{(k)} \right] \tag{4.7}$$

The order $i = 1, 2, .., N$, in which the components are corrected plays an important role. Other orderings that are common are $i = N, N{-}1, .., 1$, which leads to the *backward Gauss-Seidel iteration*, or $i = 1, 3, .., N{-}1, 2, 4, .., N$, which leads to a *red-black Gauss-Seidel iteration.*
Damping (or extrapolating) the correction of the approximate solution in a Gauss-Seidel step with a (*relaxation*) parameter ω, i.e.

$$u_i^{(k+1)} := (1-\omega)u_i^{(k)} + \omega\, u_i^{GS} \qquad i = 1, 2, .., N \tag{4.8}$$

with u_i^{GS} defined by the right-hand side of (4.6), leads to the *Successive Over-Relaxation (SOR) method.* Again it can be written in vector form as

$$\underline{u}^{(k+1)} := (D + \omega L)^{-1} \left[\omega\, \underline{b} + [(1-\omega)D - \omega R]\, \underline{u}^{(k)}\right] \tag{4.9}$$

A *backward SOR sweep* or a *red-black SOR sweep* can be defined as in the Gauss-Seidel case using a different ordering of the components. Another important variant of SOR is the *Symmetric SOR (SSOR) method* that is defined as the SOR sweep defined in (4.9) followed by a backward SOR sweep.

4.1.2 Block relaxation schemes, the Block-SOR method

An important generalization of the methods described above are *Block relaxation schemes.* They update a whole set of components at one time, typically a subvector of the solution vector, instead of only one component. The matrix A, the right-hand side $\underline{b}$ and the solution vector $\underline{u}$ are partitioned as follows:

$$A := \begin{pmatrix} A_{11} & A_{12} & \cdots & A_{1M} \\ A_{21} & A_{22} & \cdots & A_{2M} \\ \vdots & \vdots & \ddots & \vdots \\ A_{M1} & A_{M2} & \cdots & A_{MM} \end{pmatrix}, \ \underline{u} := \begin{pmatrix} \underline{u}_1 \\ \underline{u}_2 \\ \vdots \\ \underline{u}_M \end{pmatrix}, \ \underline{b} := \begin{pmatrix} \underline{b}_1 \\ \underline{b}_2 \\ \vdots \\ \underline{b}_M \end{pmatrix} \tag{4.10}$$

where the partitionings of $\underline{b}$ and $\underline{u}$ into subvectors $\underline{b}_i$ and $\underline{u}_i$ are identical and compatible with the partitioning of A. The diagonal blocks A_{ii} are square and assumed nonsingular.
Similarly to the point-wise case, a splitting

$$A := L + D + R \tag{4.11}$$

is defined with

$$D := \begin{pmatrix} A_{11} & & \\ & \ddots & \\ & & A_{MM} \end{pmatrix}, \ L := \begin{pmatrix} O & & & \\ A_{21} & O & & \\ \vdots & \ddots & \ddots & \\ A_{M1} & \cdots & A_{MM-1} & O \end{pmatrix}, \ R := \begin{pmatrix} O & A_{12} & \cdots & A_{1M} \\ & O & \ddots & \vdots \\ & & \ddots & A_{M-1M} \\ & & & O \end{pmatrix}$$

With these definitions it is easy to generalize the previous three iterative procedures, namely Jacobi, Gauss-Seidel and SOR. For example, the *Block-SOR method* is now defined by the same equation as before

$$\underline{u}^{(k+1)} := (D + \omega L)^{-1} \left[\omega \, \underline{b} + [(1-\omega)D - \omega R] \, \underline{u}^{(k)} \right] \tag{4.12}$$

except that the meanings of D, L and R have changed to their block analogues. Algorithm 4.1 shows the way the Block-SOR method is implemented. If the second

> Given: $\underline{u}^{(0)}$, ω
> <u>for</u> $k = 0, 1, 2, ...$ <u>do</u>
>   <u>for</u> $i = 1$ <u>to</u> M <u>do</u>
>
> $$\underline{u}_i^{(k+1)} := \underline{u}_i^{(k)} + \omega A_{ii}^{-1} \left(\underline{b}_i - \sum_{j=1}^{i-1} A_{ij} \underline{u}_j^{(k+1)} - \sum_{j=i}^{M} A_{ij} \underline{u}_j^{(k)} \right);$$

Algorithm 4.1 : Block-SOR method

for-loop in algorithm 4.1 is run in reverse order the *backward Block-SOR method* is obtained. One sweep of algorithm 4.1 followed by one backward sweep leads to the *Block-SSOR method.* The block systems that have to be solved for each subvector $\underline{u}_i^{(k+1)}$, namely

$$A_{ii} \underline{v}_i = \underline{d}_i \qquad i = 1, .., M \tag{4.13}$$

can be small, even 2×2, or large systems depending on the partitioning (4.10), i.e. the number of blocks M, chosen. They can be solved directly or again with an iterative technique. If direct methods are applied, the number of iterations required to achieve convergence often decreases rapidly as the block-size increases.

4.1.3 Application of Block-SOR to systems arising from the NEM-M0 discretization of the neutron diffusion equations

The LESs, arising from the NEM-M0 discretization of the neutron diffusion equations as described in the previous chapter, lead to a natural partitioning of the matrix A of the system. In figure 3.2 the pattern of the matrix A in the case of two energy groups, i.e. $G = 2$, is presented. The block systems (4.13) are as follows

$$A_{ii} \cdot \underline{v}_i := \left(\begin{array}{cc|cccccc} * & * & & & & 0 & & \\ * & * & & & & & & \\ \hline * & 0 & 1 & 0 & \cdots & \cdots & 0 \\ 0 & * & 0 & 1 & \ddots & & \vdots \\ \vdots & & \vdots & \ddots & \ddots & \ddots & \vdots \\ * & 0 & \vdots & & \ddots & 1 & 0 \\ 0 & * & 0 & \cdots & \cdots & 0 & 1 \end{array}\right) \cdot \begin{pmatrix} \phi_1^i \\ \phi_2^i \\ j_{1xl}^{i-} \\ j_{2xl}^{i-} \\ \vdots \\ j_{1zr}^{i+} \\ j_{2zr}^{i+} \end{pmatrix} = \underline{d}_i \qquad i = 1, ..., M \tag{4.14}$$

where A_{ii} is a 14×14 matrix and M corresponds to the number of boxes of the triangulation.
Algorithm 4.1 can be applied to the LES of figure 3.2 and the block systems (4.14) can be solved directly. In a first step the left upper 2×2 block for the fluxes ϕ_g^i is inverted. In a second step the currents $j_{gus}^{i\pm}$ are calculated.

A separate application of this method to the LESs is not likely to be successful. Many iterations are usually necessary to find the solution and the convergence also is not guaranteed.
One iteration of the method is, however, rather inexpensive and, when combined with the more efficient methods described in the next two sections, it can be very successful.
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 (compare section 6.2).

4.2 Preconditioned Bi-CGStab

There is a whole class of methods for the solution of LESs (4.1) that can be interpreted as projection methods on special subspaces $\underline{u}_0 + \mathcal{K}_m$ of dimension m, where $\mathcal{K}_m$ is the *Krylov subspace*

$$\mathcal{K}_m(A, \underline{r}_0) = span \left\{ \underline{r}_0, A\underline{r}_0, A^2\underline{r}_0, \, ... \, , A^{m-1}\underline{r}_0 \right\} \tag{4.15}$$

and $\underline{r}_0 := \underline{b} - A\underline{u}_0$ 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 [31, ch. 6-7].
The iterate $\underline{u}_m$ is chosen such that

$$\underline{r}_m \perp \mathcal{L}_m, \tag{4.16}$$

where $\underline{r}_m := \underline{b} - A\underline{u}_m$ 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 $\mathcal{L}_m$. If A is a *symmetric positive definite* (s.p.d.) matrix, a choice of $\mathcal{L}_m := \mathcal{K}_m(A, \underline{r}_0)$ is possible and the result is the well known *Conjugate Gradient* (CG) *method* (1952, Hestenes and Stiefel).
In the general nonsymmetric case $\mathcal{L}_m$ can be chosen to be the Krylov subspace associated with the dual system, namely $\mathcal{L}_m := \mathcal{K}_m(A^T, \underline{r}_0^*)$ with $\underline{r}_0^* := \underline{b}^* - A^T\underline{u}_0^*$. The result is the *Biconjugate Gradient* (Bi-CG) *method* (1952, Lanczos). The problem with Bi-CG is that it uses the transpose of A, which is very rarely available due to compact storing techniques. Therefore variants of the Bi-CG method, like the *Conjugate Gradient Squared* (CGS) *algorithm* (1984, Sonneveld), have been developed that do not use the transpose A^T. For ill-conditioned or time-dependent problems CGS tends to be subject to strong oscillations in the residual norm that can lead to convergence problems and to substantial build-up of rounding errors, or possibly even overflow.
The *Biconjugate Gradient Stabilized* (Bi-CGStab) *algorithm*, proposed by Van der Vorst [35] in 1992, is a variation of CGS which was developed to remedy this difficulty and "stabilize" or "smooth" the convergence behaviour of the original algorithm. Today Bi-CGStab is an established and very competitive method for solving nonsymmetric LESs. Algorithm 4.2 shows the method in the left-preconditioned form.

It has already been mentioned in the introduction to this chapter that the rate of convergence for almost all iterative methods depends largely on the condition of the matrix A and that preconditioning techniques are one of the main keys to a better performance. A preconditioning operator $P = P_L P_R$ is chosen in a way that the preconditioning systems $P_L \cdot \underline{w} = \underline{d}$ and $P_R \cdot \underline{w} = \underline{d}$ can be solved quickly and that the condition of the new (preconditioned) iteration matrix $P_L^{-1} A P_R^{-1}$ is significantly better than the condition of A. The underlying LES (4.1) is then transformed as follows:

$$A\,\underline{u} = \underline{b} \;\longmapsto\; \tilde{A}\,\underline{\tilde{u}} := \underbrace{(P_L A P_R)}_{\tilde{A}}\,\underbrace{(P_R^{-1}\underline{u})}_{\underline{\tilde{u}}} = \underbrace{P_L\,\underline{b}}_{\underline{\tilde{b}}} =: \underline{\tilde{b}} \tag{4.17}$$

and the iterative method, e.g. the Krylov subspace method, is applied to the transformed (or *preconditioned*) LES

$$\tilde{A}\,\underline{\tilde{u}} = \underline{\tilde{b}} \tag{4.18}$$

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 methods are Jacobi, SOR, or different forms of *incomplete LU-decompositions* (ILU) (see Saad [31, ch. 10]). More recently *Domain Decomposition* (DD) (Saad [31, ch. 13]) and *Multilevel* methods (Jung, Langer

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

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

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

<u>end</u>

Algorithm 4.2 : Left-preconditioned Bi-CGStab[1]

[21]) have become important for the preconditioning of Krylov subspace methods.

For the solution of the special nonsymmetric LESs arising from the NEM-M0 discretization of the neutron diffusion equations, as presented in the previous chapter, preconditioned Bi-CGStab was used. The left-preconditioned version as presented in algorithm 4.2 outperformed other versions. For the preconditioning, two steps of the Block-SOR method as per algorithm 4.1 were employed.

[1] In this form the algorithm delivers the vectors $\underline{u}_i$ corresponding to the original system (4.1) and the vectors $\underline{\tilde{r}}_i$ corresponding to the preconditioned system (4.17), i.e. $\underline{\tilde{r}}_i = P_L^{-1}(\underline{b} - A\,\underline{u}_i)$.

4.3 Multi-Grid

Large, sparse LESs (4.1) arising from the discretization of PDEs, as seen in the previous chapter (see figure 3.2) have an underlying geometrical grid. A *frequency analysis* of standard iterative methods, e.g. Jacobi or Gauss-Seidel (see section 4.1), shows that the high frequencies in the error $\underline{z}^{(k)} = \underline{u} - \underline{u}^{(k)}$ and in the *defect* (or residual) $\underline{d}^{(k)} = \underline{b} - A\,\underline{u}^{(k)} = A\,\underline{z}^{(k)}$ are reduced very well, i.e. the methods are *smoothing* the error and the defect (see Langer [28, ch. 1], Hackbusch [17]). Using results from approximation theory the low frequencies can be represented on a coarser grid, where a smaller LES has to be solved (*Two-Grid Method* (TGM)). Applying this strategy recursively on a series of grids with a decreasing number of grid points leads to the *Multi-Grid Method* (MGM).

4.3.1 The Two-Grid Method (TGM)

The easiest way to introduce a *coarse grid* is simply to double the mesh-size in each coordinate direction. For the three-dimensional (3D) neutronics problem the two

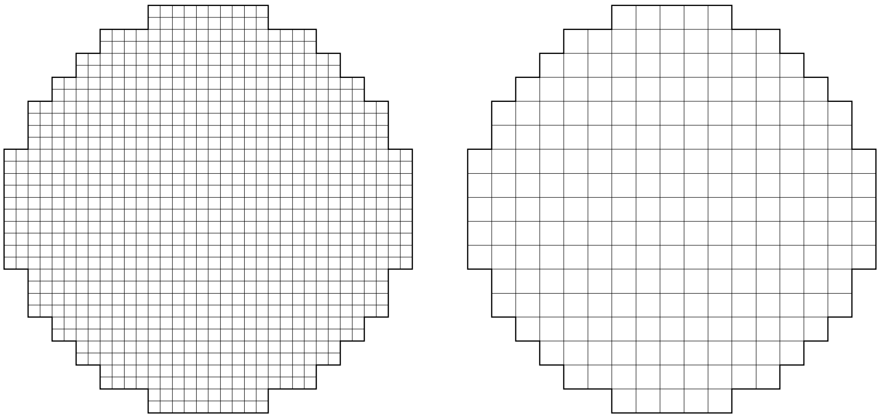


Figure 4.1: Axial cut of fine and coarse grid for the reactor problem

grids presented in figure 4.1, with 36 and 18 axial layers respectively, could for example be used.

Algorithm 4.3 shows the TGM in an algorithmic form, the coarse-grid quantities are denoted by a hat. Algorithm 4.3 has a lot of components that can still be selected:

- *Initial solution* $\underline{u}_0$*:* Basically any choice, is possible, e.g. $\underline{u}_0 = \underline{0}$. In transient problems with moderate time steps, e.g. the discrete neutron diffusion equations from chapter 3, the solution at the preceding time step is a better choice.

Initial solution:	Given: $\underline{u}_0$
	<u>for</u> $i = 0, 1, ...$ <u>until</u> convergence <u>do</u>
Smoothing Method:	<u>for</u> $k = 1$ <u>to</u> ν <u>do</u>
	$\underline{u}_i^{(k)} = S\,\underline{u}_i^{(k-1)};$
Defect-Calculation:	$\underline{d}_i = \underline{b} - A\,\underline{u}_i^{(\nu)};\ [= A\,\underline{z}_i^{(\nu)}]$
Restriction:	$\underline{\hat{d}}_i = R\,\underline{d}_i;$
Coarse-Grid-Approx.:	$\underline{\hat{w}}_i \ \underset{\text{direct}}{\overset{\text{iterative}}{\underset{:=}{\approx}}}\ \hat{A}^{-1}\,\underline{\hat{d}}_i;$
Prolongation:	$\underline{w}_i = P\,\underline{\hat{w}}_i;$
Correction:	$\underline{u}_{i+1} = \underline{u}_i^{(\nu)} + \underline{w}_i;$
	<u>end</u>

Algorithm 4.3 : Two-Grid Method (TGM)

In other problems, e.g. in the LESs arising from the generalized eigenvalue problem of appendix A, an initial solution can be generated with the *nested iteration principle* on the coarse grid and interpolated onto the fine grid. This technique will be described later on in section 4.3.2 (*Full-MGM*).

- *Smoothing method S:* ω*-Jacobi*, SOR and ILU are the most common smoothing methods (see Wesseling [39, ch. 7]). For the neutronics problem (see chapter 3) the Block-SOR method (algorithm 4.1) from section 4.1 can serve as an efficient smoothing method. The *number of smoothing steps* ν depends on the quality of the smoothing method. Usually a choice of $\nu = 1, 2$ or 3 is sufficient.

- *"Grid-transfer" operators, i.e. restriction and prolongation:* They usually are different for *vertex-centred* (e.g. FDM) and *cell-centred* (e.g. FVM) *triangulations*. The easiest choices for the prolongation are *pointwise constant, pointwise linear, bilinear or trilinear interpolation* (see Wesseling [39, ch. 5]). For more accuracy *interpolations of higher order (i.e. quadratic or cubic)* or *operator-dependent prolongations* have to be applied. The restriction is usually chosen as the adjoint of the prolongation, i.e. $R = \sigma\,P^T$, with a suitable scaling factor σ.

For the neutronics problem (from chapter 3) the prolongation is mixed constant and linear. For the fluxes ϕ_g^m a pointwise constant interpolation is used, for the currents $j_{gus}^{m\pm}$ a mixed pointwise constant and pointwise linear interpolation. For a detailed description see section 4.3.3.

- *Coarse-Grid- (CG-)Approximation:* To obtain a coarse-grid-approximation, an approximation $\hat{A}$ of A on the coarse grid is needed. There are basically two ways to choose $\hat{A}$:

  1. $\hat{A}$ is obtained by discretization of the PDEs on the coarse grid.
  2. *Galerkin-approach:* $\hat{A} := RAP$

  The coarse-grid-system

  $$\hat{A}\,\underline{\hat{w}}_i = \underline{\hat{d}}_i \tag{4.19}$$

  can then be solved with an appropriate direct or iterative method. For the neutronics problem $\hat{A}$ was obtained by discretization and preconditioned Bi-CGStab (algorithm 4.2) was used to solve (4.19).

The smoothing step in algorithm 4.3 is also referred to as the *pre-smoothing step*. In practical implementations a *post-smoothing step* is often introduced after the correction. It will be introduced in section 4.3.2 within the multi-grid algorithm.

4.3.2 The Multi-Grid Method (MGM)

The idea of how to construct a *Multi-Grid Method* (MGM) by using this Two-Grid Method (TGM) is now clear. A series of coarser grids (*levels* $q = l, l-1, .., 1$) is introduced by recursively doubling the mesh-size in each coordinate direction (compare section 4.3.3). To solve the coarse grid system (4.19) on the levels $q = l-1, .., 2$ γ_q iterations of the TGM with initial solution $\underline{w}_q = \underline{0}$ are applied. On the "coarsest" grid (level $q = 1$) the coarse grid system (4.19) will then be significantly smaller than the original system (4.1) (especially in 3D) and can therefore be solved a lot faster with an iterative or direct method.

Algorithm 4.4 shows in a pseudo-programming language, how the MGM can be used to solve an LES

$$A_l\,\underline{u}_l = \underline{b}_l \tag{4.20}$$

on the finest grid (level l). The components are the same as described for the TGM in section 4.3.1.

In comparison to algorithm 4.3, a post-smoothing step is included as well. The post-smoothing method $\mathbf{S}_q^{post}$ can be chosen differently to the pre-smoothing one

main **MGM**	
⋮	
$\underline{u}_l := \underline{u}_l^0$;	Initial Solution
for $i = 1, 2, ...$ until convergence do	
$\mathbf{MGC}(l, \underline{u}_l, \underline{b}_l, A_l)$;	
procedure $\mathbf{MGC}(q, \underline{v}_q, \underline{d}_q, A_q)$	
if $q = 1$ then $\underline{v}_q \overset{\approx}{:=} A_q^{-1}\,\underline{d}_q$;	Coarse-Grid Solver
else	
for $j = 1$ to ν_q^{pre} do	
$\underline{v}_q := \mathbf{S}_q^{pre}(\underline{v}_q, \underline{d}_q, A_q)$;	Pre-Smoothing
$\underline{d}_{q-1} := I_q^{q-1}\,(\underline{d}_q - A_q\,\underline{v}_q)$;	Defect, Restriction[2]
$\underline{w}_{q-1} := \underline{0}$	
for $k = 1$ to γ_{q-1} do	MG-Cycles, $\gamma_{q-1} = 1, 2, 3$
$\mathbf{MGC}(q-1, \underline{w}_{q-1}, \underline{d}_{q-1}, A_{q-1})$;	Coarse-Grid-Approximation
$\underline{v}_q := \underline{v}_q + I_{q-1}^q\,\underline{w}_{q-1}$;	Prolongation[2], Correction
for $j = 1$ to ν_q^{post} do	
$\underline{v}_q := \mathbf{S}_q^{post}(\underline{v}_q, \underline{d}_q, A_q)$;	Post-Smoothing
endif	

Algorithm 4.4 : Multi-Grid Method (MGM)

$\mathbf{S}_q^{pre}$ (e.g. $\mathbf{S}_q^{pre}$ is SOR and $\mathbf{S}_q^{post}$ is backward SOR).
The index q for $\mathbf{S}_q^{pre}$ and $\mathbf{S}_q^{post}$, for ν_q^{pre} and ν_q^{post} and for the restriction I_q^{q-1} and the interpolation I_{q-1}^q indicates moreover that the components on each level may differ. The only new component in the MG-algorithm 4.4 is the parameter γ_q. It is very important for the behaviour of the algorithm:

$$
\begin{array}{ll}
V-\text{cycle}: & \gamma_q := \gamma := 1, \quad \forall q = l-1, .., 2 \\
W-\text{cycle}: & \gamma_q := \gamma := 2, \quad \forall q = l-1, .., 2
\end{array}
$$

[2] I_q^{q-1} denotes the restriction R from level q to level $q-1$. I_{q-1}^q denotes the prolongation P from level $q-1$ to level q (compare section 4.3.1)

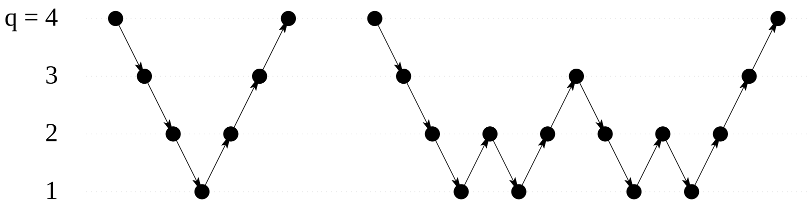


Figure 4.2: V-cycle ($\gamma = 1$) and a W-cycle ($\gamma = 2$) for the 4-Grid Method

Figure 4.2 shows a V- and a W-cycle in the case of four grids (levels). By setting $\gamma_q = 1$ for some q and $\gamma_q = 2$ for the rest, other cycles that lie between the V- and the W-cycle can also be obtained. A choice of $\gamma_q = 3$ or even higher is usually not necessary.

It has already been mentioned with the TGM in section 4.3.1 that for a transient problem the solution at the last time step is a good initial solution. For problems, where no good approximation $\underline{u}_l^0$ is available, it is better to apply the *nested iteration principle*. The nested iteration principle starts on the coarsest grid, on level $q = 1$, and recursively uses the solution $\underline{u}_{q-1}$ of level $q-1$ as an initial solution on level q. The resulting algorithm 4.5 is called *Full-Multi-Grid Method* (Full-MGM).

main **Full-MGM**	
$\vdots$	
$\underline{u}_1 \overset{\approx}{:=} A_1^{-1}\,\underline{b}_1;$	Solution on level 1
for $q = 2$ to l do	
$\underline{u}_q := \tilde{I}_{q-1}^{q}\,\underline{u}_{q-1};$	Initial Solution for level q
for $i = 1, 2, ...$ until convergence do	
MGC$(q, \underline{u}_q, \underline{b}_q, A_q);$	

Algorithm 4.5 : Full-Multi-Grid Method (Full-MGM)

The interpolation operator $\tilde{I}_{q-1}^{q}$ for the full approximation can be chosen as $\tilde{I}_{q-1}^{q} = I_{q-1}^{q}$. However it is better to use an interpolation operator of higher order.
Theoretically, the inner loop of algorithm 4.5 on each grid q should be iterated until the error is smaller than the discretization error on that grid. In practice, 1 or 2 *nested MG-iterations* for the levels $q = l-1, .., 2$ are applied and a *convergence test*

is used for the level $q = l$ (see section 4.4).

4.3.3 Application of MGM to systems arising from the NEM-M0 discretization of the neutron diffusion equations

In general the LESs, arising from the NEM-M0 discretization of the transient neutron diffusion equations as described in chapter 3, are nonsymmetric. Therefore the components of the MGM 4.4, especially the smoothing methods $\mathbf{S}_q^{pre}$ and $\mathbf{S}_q^{post}$ and the methods for the coarse-grid solution $\underline{v}_1 \overset{\approx}{:=} A_1^{-1}\,\underline{d}_1$ have to be nonsymmetric.

For the smoothing of the neutronics problem the Block-SOR method (algorithm 4.1) as described in section 4.1.2 and the variants, backward Block-SOR and Block-SSOR have been applied. One Block-SOR iteration did not have a satisfactory smoothing effect, therefore ν_q^{pre} and ν_q^{post} were chosen between 2 and 4 (see chapter 6).

The coarse-grid correction was not accurate enough with a choice of $\gamma_q = \gamma = 1$, i.e. a V-cycle. Therefore the W-cycle, i.e. $\gamma_q = \gamma = 2$, was always used. The coarse grid matrices $A_q,\ q = l-1,..,1$ were obtained by discretization on the according grid. On the coarsest grid, i.e. level $q = 1$, the LESs of all the practical numerical tests (compare chapter 6) were still too large to be solved directly. Therefore a fast iterative method for nonsymmetric problems had to be used. The method that was used for the coarse-grid solution was preconditioned Bi-CGStab, as presented in algorithm 4.2.

The best choice for an initial solution for the LESs (see figure 3.2) arising in chapter 3 from the transient problem is the solution of the problem in the preceding time step. For the solution of the LESs (A.9) arising in the stationary case, as derived in Appendix A the Full-MGM (algorithm 4.5) is the best choice (compare the numerical results in section A.3).

The grid-transfer operators have to be investigated in a bit more detail:

4.3.3.1 The Prolongation

Figure 4.1 shows 2D-cuts of a 3D-grid $\mathcal{G}$ with boxes $m = 1,..,M(= 8 \cdot \hat{M})$ and a coarser grid $\hat{\mathcal{G}}$ with double mesh-size and boxes $\hat{m} = 1,..,\hat{M}$. One coarse-grid box consists of 8 fine-grid boxes as shown in figure 4.3. For the prolongation P from the coarse to the fine grid the fluxes $\phi_g^m,\ g = 1,..,G$ and the currents $j_{gus}^{m\pm},\ g = 1,..,G,\ u = x,y,z,\ s = l,r$ are treated differently.

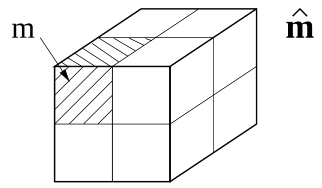


Figure 4.3: Uniform refinement of a rectangular 3D-grid

To consider one coarse-grid box $\hat{m} \in \{1, .., \hat{M}\}$. As defined in section 3.1 using the same notation, the discrete unknowns $\phi_g^{\hat{m}}$ represent the *zeroth momenta*

$$\phi_g^{\hat{m}} := \phi_g^{\hat{m}}(t_j) = \frac{1}{V^{\hat{m}}} \int_{\mathbf{\Omega}^{\hat{m}}} \phi_g(\vec{x}, t_j) d\vec{x}$$

of the fluxes $\phi_g(\vec{x}, t)$ over the coarse 3D-box $\mathbf{\Omega}^{\hat{m}}$ at the discrete time t_j. Because of the weighted integral formulation it makes sense to choose the zeroth momenta ϕ_g^m over the fine-grid boxes $\mathbf{\Omega}^m \subset \mathbf{\Omega}^{\hat{m}}$ simply as

$$\phi_g^m = \phi_g^{\hat{m}}, \quad m \in \{m | \, \mathbf{\Omega}^m \subset \mathbf{\Omega}^{\hat{m}}\}, \tag{4.21}$$

i.e. *piecewise constant interpolation.*

The discrete unknowns $j_{gus}^{\hat{m}\pm}$ on the other hand are defined as the zeroth momenta

$$j_{gus}^{\hat{m}+} - j_{gus}^{\hat{m}-} := j_{gus}^{\hat{m}+}(t_j) - j_{gus}^{\hat{m}+}(t_j) = \frac{1}{A_u^{\hat{m}}} \int_{\mathbf{\Gamma}_{us}^{\hat{m}}} \vec{j}_g(\vec{x}, t_j) \cdot \vec{e}_u \, dA$$

of the currents $\vec{j}_g(\vec{x}, t)$ over the surfaces $\mathbf{\Gamma}_{us}^{\hat{m}}, \; u = x, y, z, \; s = l, r$ of the coarse-grid box. In the same way as above, piecewise constant interpolation can therefore be applied on the surfaces of the fine-grid boxes that coincide with the surfaces of the coarse-grid box:

$$j_{gus}^{m\pm} = j_{gus}^{\hat{m}\pm}, \quad m \in \{m | \, \mathbf{\Gamma}_{us}^m \subset \mathbf{\Gamma}_{us}^{\hat{m}}\}.$$

For the currents of the fine-grid boxes m on the inner surfaces $\mathbf{\Gamma}_{us}^m \subset \mathbf{\Omega}^{\hat{m}} \backslash \mathbf{\Gamma}_{us}^{\hat{m}}$ that point in direction u from left to right, the average of the incoming and the outgoing current of the coarse-grid box in direction u from left to right are used, i.e. for the right outgoing current j_{gur}^{m+} of the fine-grid box m, as shown in figure 4.4, the prolongation

$$j_{gur}^{m+} = \frac{1}{2}\left(j_{gur}^{\hat{m}+} + j_{gul}^{\hat{m}+}\right) = \frac{1}{2}\left(j_{gur}^{\hat{m}+} + j_{gur}^{\hat{m}'+}\right).$$

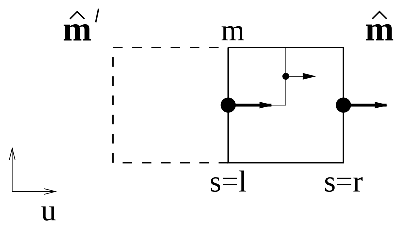


Figure 4.4: Interpolation of the current j_{gur}^{m+} on an inner surface

is used. The second identity is a consequence of condition (3.14), since the currents are continuous on the interfaces. The prolongation for the currents that point in direction u from right to left can be defined analogously and together the following *piecewise linear interpolation* for the currents $j_{gus}^{m\pm},\ g = 1,..,G,\ u = x,y,z,\ s = l,r$ in direction u results:

$$j_{gus}^{m\pm} = \begin{cases} j_{gus}^{\hat{m}\pm} & \text{for } m \in \mathcal{A}^{\hat{m}} := \{m |\, \mathbf{\Gamma}_{us}^{m} \subset \mathbf{\Gamma}_{us}^{\hat{m}}\} \\ \frac{1}{2}\left(j_{gur}^{\hat{m}+} + j_{gur}^{\hat{m}'+}\right) & \text{for } m \in \mathcal{I}^{\hat{m}} := \{m |\, \mathbf{\Gamma}_{us}^{m} \subset \mathbf{\Omega}^{\hat{m}} \backslash \mathbf{\Gamma}_{us}^{\hat{m}}\} \end{cases} \tag{4.22}$$

$\hat{m}'$ denotes the left neighbouring box of $\hat{m}$ for the right currents and the right neighbouring box for the left currents respectively.

4.3.3.2 The Restriction

For the restriction R the following approach (compare section 4.3.1) is used:

$$R = \sigma\, P^T \tag{4.23}$$

Because of the particular discretization with finite volumes and the non-dimensionalization, i.e. the division through the volume and the area respectively, the problem does not depend on the grid size. As a consequence the restriction $R\,\underline{u}$ has to be a coarse-grid-approximation of $\underline{u}$. Therefore the row sum of the restriction matrix has to be 1, i.e. it has to fulfill

$$\sum_j R_{ij} = 1 \tag{4.24}$$

For a coarse-grid box $\hat{m} \in \{1,..,\hat{M}\}$ the flux $\phi_g^{\hat{m}},\ g = 1,..,G$ on the coarse-grid is constructed from the values on the fine-grid as

$$\phi_g^{\hat{m}} = \sigma_\phi \left(\sum_{\{m|\, \mathbf{\Omega}^m \subset \mathbf{\Omega}^{\hat{m}}\}} \phi_g^m \right) \tag{4.25}$$

Since one coarse-grid box consists of 8 fine-grid boxes (in the form that was used in equation (4.25): $|\{m|\,\mathbf{\Omega}^m \subset \mathbf{\Omega}^{\hat{m}}\}| = 8$), it follows from (4.24) that

$$\sigma_\phi := 1/8.$$

The currents $j_{gus}^{\hat{m}\pm}$, $g = 1,..,G$, $u = x,y,z$, $s = l,r$ on the coarse-grid box $\hat{m}$ are constructed similarly. Figure 4.5 shows that for each coarse-grid current 12 fine-grid

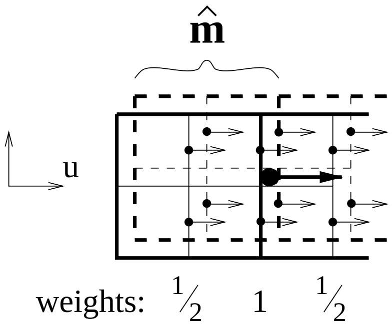


Figure 4.5: Restriction of the currents (with weights)

currents are used. As indicated in figure 4.5 the weights are 1/2 for the currents on inner surfaces and 1 on surfaces that coincide with the coarse-grid surface. In the form that was used before, the $j_{gus}^{\hat{m}\pm}$ are derived as

$$j_{gus}^{\hat{m}\pm} = \sigma_j \left(\sum_{m\in\mathcal{A}^{\hat{m}}} 1/2 \cdot j_{gus}^{m\pm} + \sum_{m\in\mathcal{I}^{\hat{m}}} j_{gus}^{m\pm} + \sum_{m\in\mathcal{A}^{\hat{m}'}} 1/2 \cdot j_{gus}^{m\pm} \right) \tag{4.26}$$

where $\hat{m}'$ denotes the appropriate neighbouring box again.

The parameter σ_j follows from (4.24) again. The sum of the weights for the currents is $4 \cdot 1/2 + 4 \cdot 1 + 4 \cdot 1/2 = 8$. For the row sum of R to be 1,

$$\sigma_j := 1/8.$$

Therefore the general formula (4.23) for the restriction holds with

$$\sigma := \sigma_\phi = \sigma_j = 1/8 \tag{4.27}$$

4.4 Convergence tests

In the previous sections iterative methods have been described for the solution of the LES (4.1), but it has not been specified, how to find out, whether the iterate $\underline{u}^{(k)}$ is close enough to the exact solution $\underline{u}$ of the LES or not.

Theoretically the described methods, i.e. Bi-CGStab (algorithm 4.2), MGM (algorithm 4.4) or Full-MGM (algorithm 4.5), are iterated *"until convergence occurs"*, but the accuracy that can be obtained depends on the order of the spatial discretization. It makes no sense to reduce the error below the *discretization error* caused by the applied discretization. Therefore the norm of the *iteration error*

$$\|\underline{z}^{(k)}\| = \|\underline{u} - \underline{u}^{(k)}\| \tag{4.28}$$

after k iterations, in an appropriate norm $\|\cdot\|$, has got to be of the same order as the norm of the discretization error.

In practice, the exact solution $\underline{u}$ is usually not available though and an approximative test is needed to check the accuracy of the iterate. One approach for a practical convergence test is, to ask for the reduction of the error to be small in a particular norm:

$$\|\underline{z}^{(k)}\| \leq \varepsilon_R \, \|\underline{z}^{(0)}\| \tag{4.29}$$

with an appropriate $\varepsilon_R > 0$. Since the standard *Euclidean norm* $\|\cdot\|_2$ of the error can not be derived in practice, another norm is needed. A norm that can be derived is the A^TA-energy norm that is defined as

$$\|\underline{u}\|_{A^TA} := \left\langle A^T A\, \underline{u}, \underline{u} \right\rangle^{\frac{1}{2}} \quad \left[= \langle A\,\underline{u}, A\underline{u}\rangle^{\frac{1}{2}} = \|A\,\underline{u}\|_2 \,\right] \tag{4.30}$$

The A^TA-energy norm of the error $\underline{z}^{(k)}$ is given by

$$\|\underline{z}^{(k)}\|_{A^TA} = \|A\underline{z}^{(k)}\|_2 = \|A\,\underline{u} - A\,\underline{u}^{(k)}\|_2 = \|\underline{b} - A\,\underline{u}^{(k)}\|_2 = \|\underline{r}^{(k)}\|_2,$$

i.e. the Euclidean norm of the residual $\underline{r}^{(k)}$. The resulting convergence test is given by

$$\|\underline{r}^{(k)}\|_2 \leq \varepsilon_R \, \|\underline{r}^{(0)}\|_2 \tag{4.31}$$

and is called *defect (or residual) test*. The defect test is a good test for the theoretical *convergence analysis* of an algorithm and for the comparison of two different algorithms, but it depends on the quality of the initial solution and is therefore not very useful in practice. For a fixed ε_R solutions of basically any absolute accuracy can be constructed by simply varying the initial solution. To guarantee a reasonable solution, ε_R has to be chosen to be very small and the method is not very efficient in practice any more.

In practical numerical tests it is better to check whether the correction

$$\Delta\underline{u}^{(k)} = \underline{u}^{(k)} - \underline{u}^{(k-1)} \tag{4.32}$$

in the k^{th} iteration is "small" in a particular norm. The accuracy ε can then be chosen from the underlying physics. The norm is chosen as the particular weighted norm $\|\cdot\|_W$ that was already used in section 3.2 for the error estimate of the integration scheme:

$$\|\Delta\underline{u}^{(k)}\|_W := \|W\Delta\underline{u}^{(k)}\|_2, \quad \text{with } W := (w_{ii}),\ w_{ii} := \frac{1}{\left|\left(\underline{u}^{(k)}\right)_i\right| + \delta} \tag{4.33}$$

The parameter δ is chosen to be the smallest interesting value for $|(\underline{u}^{(k)})_i|$. The vector $\underline{u}^{(k)}$ in the neutronics problem has components of different magnitude, e.g. the unknowns ϕ_g in the different energy groups $g = 1,..,G$. In this case the parameter δ can be chosen differently for each energy group g.

A convergence test can now be defined by simply asking for

$$\|\Delta\underline{u}^{(k)}\|_W \leq \varepsilon_W \tag{4.34}$$

Because of the component-wise scaling of $\Delta\underline{u}^{(k)}$ in this particular norm it is again a relative error-test, but ε_W corresponds directly with the number of valid digits of the solution $\underline{u}^{(k)}$. Because of

$$\begin{aligned} \|\underline{r}^{(k)}\|_W &= \|W\,\underline{r}^{(k)}\|_2 = \|WAA^{-1}\underline{r}^{(k)}\|_2 = \\ &= \|WAW^{-1}W\Delta\underline{u}^{(k)}\|_2 \leq \|A\|_2\,\|\Delta\underline{u}^{(k)}\|_W \leq \ \|A\|_2\,\varepsilon_W \end{aligned}$$

it can also be guaranteed that the residual $\underline{r}^{(k)}$ is small and that the iteration method is not caught in a *saddle point*.

For the comparison of the efficiency of the new methods with the existing methods in the reactor simulation package PANBOX2 of the Siemens AG Operating Division KWU [3], their convergence test

$$\max_{m=1}^{M} \left| \frac{\left(\phi_1^{(k)}\right)^m}{\left(\phi_1^{(k-1)}\right)^m} - 1 \right| < \varepsilon_P, \tag{4.35}$$

was also implemented, i.e. only the fast neutrons from energy group $g = 1$ are used (compare the numerical results in section 6.4).

Chapter 5

Parallelization

In Bauer [2, ch. 1] parallelized algorithms are assigned to one of four general parallel programming models:

The underlying algorithm of the parallelized block can be classified as either *homogeneous* or *heterogeneous*. Homogeneous parallelization applies the same code to multiple data elements, while heterogeneous parallelization applies multiple codes to multiple data elements. Obviously there is a grey area between these two models and it is possible that both models could occur in a mixed form in an algorithm.
A key-issue in both models is the division of the code into smaller chunks that are then run on multiple processors. The key to making this procedure work is *independence* between the chunks, i.e. one chunk does not modify a variable that another might be simultaneously reading or modifying. Where they can not be avoided, dependencies between chunks result in *communication* between and/or *synchronization* of the processors.

- **Homogeneous parallelization** is possible when the work to be done by the algorithm can be broken into identical (homogeneous), independent subtasks, e.g. dividing a loop with 100 completely independent iterations into four groups of 25 iterations and executing each group separately on one of four processors.

- **Heterogeneous parallelization** is possible when the work to be done by the algorithm is spread over a large number of *different*, independent subtasks, each of which works a discrete portion of the total algorithm.

The second classification that can be made deals with the parallel work distribution and is described in terms of *granularity*. Granularity is categorized broadly as either *fine* or *coarse*. **Fine-grained** distributions spread the total work to be done by the parallelized algorithm over many small subtasks, while **coarse-grained** distributions have fewer subtasks each doing more of the work. The distinction between the two is obviously not unique and becomes only clear at the extremes.

An alternative classification of parallel algorithms by Flynn [13] is based on the presence of single or multiple streams of instructions:

SISD *Single-instruction, single-data* stream. Serial computer.

MISD *Multiple-instruction, single-data* stream. Heterogeneous parallelization, implemented with appropriate safeguards for access to the shared data.

SIMD *Single-instruction, multiple-data* streams. Homogeneous parallelization.

MIMD *Multiple-instruction, multiple-data* streams. Heterogeneous parallelization with entirely independent subtasks.

Almost as important as the choice of the appropriate algorithm is the question what *hardware* is or can be used and what *software* is available for the generation and implementation of parallelism. Basically three hardware-configurations for running algorithms in parallel can be distinguished:

- Workstation Cluster (workstations linked by a connecting network)
- Multiprocessor Computers with *distributed* memory
- Multiprocessor Computers with *shared* memory

On processors that do not share their memory, so-called *message passing platforms*, like MPI or PVM, have to be used for the communication between the processors. On shared-memory machines the operating system usually offers system functions and library routines to spawn subprocesses, called *threads*. At the moment it is easier to transport parallel code onto different architectures, when message passing platforms are used, but the performance of thread-based parallelization is far better and it is not very hard to make the code portable (e.g. by *POSIX-threads*).

In the first section of this chapter the direct parallelization of the particular serial algorithms for the solution of sparse linear equation systems (LESs) that were developed in the previous chapter are described. The algorithms are parallelized by *partitioning* the underlying numerical grid, or in the general case the data, and distributing the arising work on multiple processors (homogeneous parallelization). The whole solution method is started in parallel on the processors with each processor working on a different share of the grid or data (coarse-grained).

In the second section the implementation is described. The hardware was a Silicon Graphics, Inc. (SGI) multiprocessor computer with shared memory. The operating system, IRIX 6.1, supports thread-based parallelization techniques. For the generation and control of the parallel subprocesses *parallel primitives* (see Bauer [2, ch. 3-4]) were used: the **m**-*library*.

5.1 Parallelization of Multi-Grid and Bi-CGStab by grid partitioning

5.1.1 Grid partitioning

The first step in the direct parallelization of methods for the solution of LESs arising in the discretization process of partial differential equations (PDEs) is the partitioning of the grid. The physical domain $\mathbf{\Omega}$ has to be decomposed into subdomains $\mathbf{\Omega}_i$, $i = 1, .., D$, e.g. like in figure 5.1. The subdomain boundaries have to coincide with

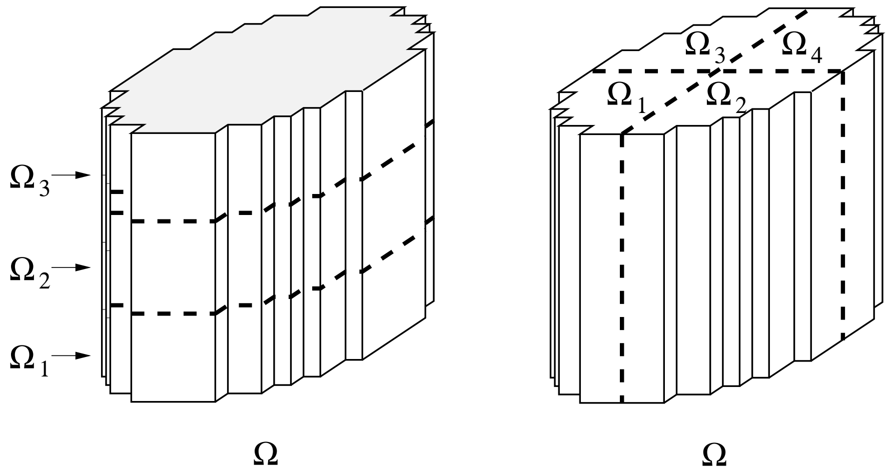


Figure 5.1: Two possible ways of partitioning $\mathbf{\Omega}$

the grid-element boundaries. On a whole hierarchy of grids, this condition has to hold for all the grids. In the case of uniform coarsening (refinement) it is therefore enough that the condition holds on the coarsest grid.

5.1.1.1 Grid partitioning for the NEM-M0 discretization of the multigroup neutron diffusion equations

For the NEM-M0 discretization of the multigroup neutron diffusion equations on a domain $\mathbf{\Omega}$, as presented in figure 5.1, a uniform rectangular grid, as presented in figure 4.1 was used. The partitioning of this grid can for example be carried out as presented in figure 5.1 for the domain $\mathbf{\Omega}$.

The important question is now, how the grid partitioning can be used to parallelize the methods that were developed in the last chapter for the solution of the arising N-dimensional LESs as presented in figure 3.2:

$$A \cdot \underline{u} = \underline{b} \tag{5.1}$$

The partitioning of the data, i.e the matrices and the vectors, is a direct consequence of the grid partitioning. The numbering of the unknowns in the vector $\underline{u}$ is as in figure 3.2. For two energy groups:

$$\underline{u} = (\phi_1^1, \phi_2^1, j_{1xl}^{1-}, j_{2xl}^{1-}, .., j_{1zr}^{1+}, j_{2zr}^{1+}, \;\dots\;, \phi_1^M, \phi_2^M, j_{1xl}^{M-}, j_{2xl}^{M-}, .., j_{1zr}^{M+}, j_{2zr}^{M+})^T$$

Since incoming and outgoing currents are used and not the total current on the box-surface, each component of the vector $\underline{u}$ belongs exactly to one box $m = 1, .., M$ and therefore to one subdomain $\mathbf{\Omega}_i$, $i = 1, .., D$. By appropriate numbering of the boxes it is possible to always obtain the following partitioning of the matrices and vectors

$$A = \begin{pmatrix} A_1 & \mathcal{A}_{12}^{off} & \cdots & \mathcal{A}_{1D}^{off} \\ \mathcal{A}_{21}^{off} & A_2 & & \mathcal{A}_{2D}^{off} \\ \vdots & & \ddots & \vdots \\ \mathcal{A}_{D1}^{off} & \mathcal{A}_{D2}^{off} & \cdots & A_D \end{pmatrix} \quad \text{and} \quad \underline{u} = \begin{pmatrix} \underline{u}_1 \\ \underline{u}_2 \\ \vdots \\ \underline{u}_D \end{pmatrix} \tag{5.2}$$

N_i denotes the number of local components in domain $\mathbf{\Omega}_i$. It follows that

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

The advantage that arises from the particular NEM-M0 discretization is that the vectors do not have to be assembled on the subdomain interfaces, but there is still need for "communication" between the processes for the matrix-vector multiplication and the scalar product.

5.1.2 Coarse-grained homogeneous parallelization of Multi-Grid and Bi-CGStab

Both the Multi-Grid algorithm 4.4 (MGM) and the Bi-CGStab algorithm 4.2 for the solution of an LES 5.1 can be fully described in terms of *vector-vector operations* and *matrix-vector multiplications* together with a small number of *scalar operations*[1]. 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 parallelization**).

The whole algorithm (4.4 for Multi-Grid, 4.2 for Bi-CGStab) is started in parallel on multiple processors (**coarse-grained**). The matrix-vector multiplications and vector-vector operations are distributed on the multiple processors, each working on his part of the grid, or matrix, or vector.

For the basic vector-vector operations, like

- Copying: $\underline{y} = \underline{x} \longrightarrow \underline{y}_i = \underline{x}_i \quad \forall\, i = 1, .., D$
- Initializing: $\underline{x} = c \longrightarrow \underline{x}_i = c \quad \forall\, i = 1, .., D$
- AXPY: $\underline{y} = \underline{y} + a \cdot \underline{x} \longrightarrow \underline{y}_i = \underline{y}_i + a \cdot \underline{x}_i \quad \forall\, i = 1, .., D$
- Scaling: $\underline{x} = a \cdot \underline{x} \longrightarrow \underline{x}_i = a \cdot \underline{x}_i \quad \forall\, i = 1, .., D$

it is clear that the iterations (for each component) are independent. Problems arise in *scalar products* and matrix-vector multiplications, as well as in the application of the Block-SOR method (algorithm 4.1) that is needed for the smoothing and for the preconditioning in the two methods.

5.1.2.1 Communication / Synchronization

In the case of scalar products and matrix-vector multiplications the iterations are not entirely independent, but with some communication between or synchronization of the processes they can also be run in parallel.

The **scalar product**

$$\sigma = \langle \underline{x}, \underline{y} \rangle$$

can be parallelized by *reduction*. Local scalar products

$$\sigma_i := \langle \underline{x}_i, \underline{y}_i \rangle \quad \forall\, i = 1, .., D \tag{5.3}$$

[1] Algorithm A.2 for the solution of the generalized eigenvalue problem in Appendix A can also be fully described in terms of vector-vector operations and matrix-vector multiplications together with a small number of scalar operations and can be parallelized in the same way as MGM and Bi-CGStab

for each subdomain $\mathbf{\Omega}_i$, $i = 1, .., D$ are introduced and the global scalar product σ is calculated from

$$\sigma = \sum_{i=1}^{D} \sigma_i \tag{5.4}$$

To derive that sum on a processor configuration with distributed memory, communication is needed. The local scalar products σ_i are sent from all the *slaves* to the *master*, the sum is evaluated there and then the result is sent back to the slaves. If the slaves share the memory with the master, a synchronization point after the summation replaces the communication.

For the **matrix-vector multiplication**[2]

$$\underline{y} = A \cdot \underline{x} \qquad \text{or} \qquad \begin{pmatrix} \underline{y}_1 \\ \underline{y}_2 \\ \vdots \\ \underline{y}_D \end{pmatrix} = \begin{pmatrix} A_1 & \mathcal{A}_{12}^{off} & \cdots & \mathcal{A}_{1D}^{off} \\ \mathcal{A}_{21}^{off} & A_2 & & \mathcal{A}_{2D}^{off} \\ \vdots & & \ddots & \vdots \\ \mathcal{A}_{D1}^{off} & \mathcal{A}_{D2}^{off} & \cdots & A_D \end{pmatrix} \cdot \begin{pmatrix} \underline{x}_1 \\ \underline{x}_2 \\ \vdots \\ \underline{x}_D \end{pmatrix}$$

communication/synchronization is needed too. The off-diagonal submatrices $\mathcal{A}_{ij}^{off}$ are zero almost everywhere. They have only entries in components that relate to

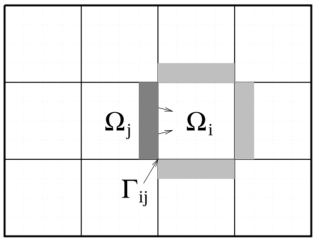


Figure 5.2: Boundary boxes needed for matrix-vector multiplication

boxes along the subdomain boundaries. In the case of the NEM-M0 discretization the entries of $\mathcal{A}_{ij}^{off}$ depend on data from $\mathbf{\Omega}_j$, but $\mathcal{A}_{ij}^{off}$ is defined on the processor that works on $\mathbf{\Omega}_i$. Therefore the according components of $\underline{x}_j$ have to be sent from $\mathbf{\Omega}_j$ to $\mathbf{\Omega}_i$ (see figure 5.2). The components that are needed are the ones that correspond to the outgoing currents of $\mathbf{\Omega}_j$, which are defined on the interface $\mathbf{\Gamma}_{ij}$ between $\mathbf{\Omega}_i$ and $\mathbf{\Omega}_j$, namely (using the notation of section 3.1) the

$$j_{gus}^{m\pm} \quad \text{with} \quad m \in \{m | \, \mathbf{\Omega}^m \subset \mathbf{\Omega}_j \wedge \mathbf{\Gamma}_{us}^m \subset \mathbf{\Gamma}_{ij}\}.$$

[2]Restriction and prolongation of a vector are also matrix-vector multiplications

For each domain $\mathbf{\Omega}_j$, $j \neq i$ a vector $\underline{x}_{ji}^{off} \in \mathbb{R}^{N_j}$ is defined component-wise with $l = 1, .., N_j$ by

$$(\underline{x}_{ji}^{off})_l = \begin{cases} (\underline{x}_j)_l & \text{for } (\underline{x}_j)_l \in \mathbf{\Gamma}_{ij} \\ 0 & \text{else} \end{cases} \tag{5.5}$$

Each process can then evaluate its part of the matrix vector product by

$$\underline{y}_i = A_i \cdot \underline{x}_i \; + \sum_{j \neq i} \mathcal{A}_{ij}^{off} \cdot \underline{x}_{ji}^{off} \quad \forall\, i = 1, .., D \tag{5.6}$$

In the distributed-memory case, the entries of the vector $\underline{x}_{ji}^{off}$, as presented in (5.5), have to be sent from the processor that works on $\mathbf{\Omega}_j$ to the processor that works on $\mathbf{\Omega}_i$. In the shared-memory case a synchronization point is set instead.

5.1.2.2 Modified Block-SOR

The solution of one block in the Block-SOR method always depends on the solution of the previous blocks. For a straightforward parallelization of the Block-SOR method it would be necessary that the processors have to wait for each other. It is therefore better to modify the method.

If the special block-splitting (4.11) for the matrix A from section 4.1.2 is used, the Block-SOR method (algorithm 4.1) can be written in the following implicit matrix form

$$\underline{u}^{(k+1)} := \underline{u}^{(k)} + \omega D^{-1} \left(\underline{b} - L\, \underline{u}^{(k+1)} - R\, \underline{u}^{(k)} \right) \tag{5.7}$$

The same partitioning of the matrix A as in (5.2) is used and the local matrices A_i are split into $A_i = L_i + D_i + R_i$ as in (4.11). The method can now be parallelized as

$$\underline{u}_i^{(k+1)} := \underline{u}_i^{(k)} + \omega D_i^{-1} \left(\underline{b}_i - L_i\, \underline{u}_i^{(k+1)} - R_i\, \underline{u}_i^{(k)} - \sum_{j \neq i} \mathcal{A}_{ij}^{off}\, \underline{u}_{ji}^{off} \right) \quad \forall\, i = 1, .., D \tag{5.8}$$

the vector $\underline{u}_{ji}^{off}$ is defined as in (5.5) and contains the values of the previous local iterate $\underline{u}_j^{(k)}$ of domain $\mathbf{\Omega}_j$. To get it, communication between the processes or a synchronization point is once again needed prior to each iteration.

This method (5.8) exactly defines a *Block-Jacobi* method for the block system defined by the grid partitioning in (5.2). On each individual block the Block-SOR method (algorithm 4.1) is applied.

5.2 Implementation on a multiprocessor computer with shared memory

There are different ways to invoke threads for the parallelization of serial algorithms on a multiprocessor computer with shared memory. Basically they can be divided into three groups:

- Compiler for the analysis and automatic parallelization
- Directives for the (C-, Fortran-, ...) compiler
- Manual generation and control of multiple threads, using system functions and libraries

Within the **IRIX 6.1** operating system on an **SGI** multiprocessor computer the following compilers and libraries are available for the parallelization of C and Fortran code:

For the analysis and automatic parallelization it offers the *Power C Analyzer* (PCA) and the *Power Fortran Accelerator* (PFA) (Bauer [2, ch. 16–21]). However if the data structure is complicated, the analysis and automatic parallelization of the code is bound to fail.

The *IRIS Power C* and the *IRIS Power Fortran* compiler (Bauer [2, ch. 7–15]) have directives to generate and control multiple threads, but the compiler can not deal with very coarse-grained algorithms. If there are dependencies in subroutines that are called in a parallel region, it is impossible to deal with them.

To apply the parallelization strategy of the previous section and parallelize the algorithms of chapter 4, it is therefore absolutely necessary to generate and control the threads manually. The system function to start multiple threads is **sproc()**. To control them, *arenas*, *semaphores*, etc. have to be defined. The handling of these functions requires an excellent knowledge of system-level programming and is definitely non-portable. It is therefore better and easier to use predefined *parallel primitives* (Bauer [2, ch. 3-4]), the **m**-library (table 5.1).

5.2.1 The parallel primitives (C and Fortran)

Parallelization is implemented by spawning a subroutine, which is then executed on multiple threads. Additional routines are provided to coordinate access to shared memory, provide process synchronization, and manage the spawned slave threads.

The primitives have equivalents in both Fortran and C, the **m**-routines. An overview of all the **m**-routines is given in table 5.1.

m_fork($function, arg1, .., arg5$)	.....	start threads
m_kill_procs()	.....	terminate threads
m_set_procs($value$)	.....	set number of threads
m_get_numprocs()	.....	get number of threads
m_get_myid()	.....	get this thread id
m_park_procs()	.....	suspend threads
m_rele_procs()	.....	resume threads
m_next()	.....	global counter
m_sync()	.....	synchronize threads
m_lock()	.....	start critical section
m_unlock()	.....	end critical section

Table 5.1: **M**–library

5.2.1.1 Spawning and terminating parallel processes

The main function for effecting parallelization is **m_fork()**. It uses **sproc()** to generate the threads.

By default the call **m_fork**($function, arg1, .., arg5$) creates $maxproc - 1$ threads of the subroutine $function(arg1, .., arg5)$, where $maxproc$ represents the maximum number of processors available for execution on the system. The calling routine runs on the remaining processor. The number $nproc$ of execution threads can be set to other values with **m_set_procs**($nproc$) in advance of the **m_fork()** call. The calling thread and all those created by **m_fork()** then execute the subroutine $function()$ in parallel. The **m_fork()** call returns to the main program or to the calling routine when all threads have completed execution.

The threads work independently of one another; synchronous execution is not required. Each thread is uniquely identified with a thread id (TID), in the range of 0 to $nproc - 1$, which is queried with **m_get_myid()**. Each inherits the system environment of the master thread.

Processes spawned by a previous **m_fork()** call are terminated with **m_kill_procs()**.

In addition the threads can be suspended during prolonged stretches of serial code to free up system resources by using **m_park_procs()**. The threads are reactivated by **m_rele_procs()**. With **m_get_numprocs()** the number of available threads can be found out and **m_next()** returns a global counter that is available to all threads. However these four functions are not needed.

5.2.1.2 Shared-memory access

The distinction between local memory of one thread and shared memory is made as follows:

The threads share the arguments $arg1, .., arg5$ of $function()$. In addition they also share the global variables, arrays and pointers in C and the COMMON blocks in Fortran. The variables, arrays and pointer that are defined in $function()$ on the other hand are local on each thread. In addition it is possible in Fortran to explicitly specify local COMMON blocks for $function()$ on each thread.

To control the access to the shared memory, the **m**-library offers the functions **m_sync()**, **m_lock()** and **m_unlock()**. The routine pair **m_lock()** and **m_unlock()** define blocks in parallelized 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. This construction is necessary, if a global scalar variable is modified by more than one process at the same time.

The routine **m_sync()** sets a *synchronization point*. It forces each thread to pause in execution and wait until all threads have reached the same point in the code. Once all threads are synchronized, execution resumes. Interthread synchronization isolates blocks within a group of threads so that execution in one does not progress into subsequent blocks until the entire block is executed by all threads. This avoids flow dependence during parallel execution.

5.2.2 Parallel implementation of Multi-Grid and Bi-CGStab

First of all the vectors and matrices that describe the LESs (5.1) that have to be solved, as well as the vectors and matrices that are needed in the solution algorithms, have to be defined as shared.

The matrices and vectors are then partitioned as in (5.2) with D domains. The global indices of the first and the last component of the subvectors $\underline{u}_i$, $i = 1, .., D$ have to be derived. With **m_set_procs**(D) the number of threads is set according to the number of domains D.

The method for the solution of the LES is started in parallel with **m_fork**(**MGM**, A, $\underline{u}, \underline{b}, ...$) or **m_fork(Bi-CGStab**, $A, \underline{u}, \underline{b}, ...$), respectively. Each thread retrieves his TID with **m_get_myid()**. The local vector-vector operations, the local matrix-vector products and the local Block-SOR iterations for the domain i, as defined in section 5.1.2, are executed on the thread with TID $= i - 1$.

Prior to each matrix-vector multiplication and prior to each iteration of the Block-SOR method a synchronization point has to be set with **m_sync()**, to make sure that the currents of neighbouring domains are up to date.

For the calculation of scalar products, local scalar variables σ_i for each thread and one shared scalar variable σ have to be defined. The local scalar products σ_i, $i = 1, .., D$ can then be calculated independently by each thread. The calculation of the global scalar product σ from (5.4) has to be put into a *critical section* though, i.e. it has to be preceded by **m_lock()** and followed by **m_unlock()**. Using C-syntax (5.4) can be implemented as

> **m_lock**();
> σ += σ_i;
> **m_unlock**();

Once the solution method is finished the threads are terminated with **m_kill_procs()**.

Chapter 6

Numerical simulation

The described serial and parallel solution methods, Multi-Grid and preconditioned Bi-CGStab, were finally applied to the linear equation systems arising in the simulation process of realistic reactor problems to test their robustness and efficiency. The mathematical model that was used to calculate the neutron distribution in the reactor was the multigroup diffusion, as described in chapter 2.

There was data for three typical reactor simulation problems available. They are described in the first section of this chapter. The discretization of the multigroup neutron diffusion equations was carried out as described in chapter 3, using the line method with the spatial discretization NEM-M0 and the time discretization TR-BDF2[1], on three grids of different fineness.
At the beginning of the second section the components and parameters that were used for the Multi-Grid method and the Bi-CGStab method are specified. In the following the performance of the serial methods is compared for one time step and for a whole period of the simulation. The asymptotic behaviour is investigated as well as the behaviour of the time step control.
The parallel performance is presented in the third section. The speedup of the two methods on the different grids is calculated and compared to the results reported in the literature. As in the serial case, the parallel methods are also compared for a whole period of the simulation once more.
In the final section the efficiency of the methods, in particular Bi-CGStab, is compared to the efficiency of the established methods used in the existing nuclear reactor simulation code PANBOX2 of the Siemens AG Operating Division KWU [3].

The hardware that was used was an SGI Power Challenge shared memory multipro-

[1] or Implicit backward-Euler for the comparison with PANBOX (see section 6.4)

cessor computer with 12 MIPS–R8000 processors (360 MFLOPS peak each), 1.56 GByte main memory and 4 MByte Cache per processor. The operating system was IRIX 6.1.

6.1 Description of the numerical test problems

It was mentioned in section 2.7 that the complexity of the processes occurring in a nuclear power plant makes it necessary to have a number of interacting modules to describe the various processes. Usually the processes that have an influence on the neutron distribution have to be simulated simultaneously and they have to be coupled to the neutronics simulation.

The numerical results in this chapter were obtained by solving the multigroup diffusion equations (2.26, 2.27) for the neutron distribution in the nuclear reactor, on their own. For the calculation and homogenization of the input data, i.e. the cross sections $\Sigma_{\cdot g}$, the program PANBOX2 (PANBOX) [3] was used. PANBOX was also used to update the cross sections in each time step on the basis of the changes caused by the other processes. The coupling with PANBOX was achieved through a file that contained the cross sections for a number of consecutive time steps.
In PANBOX the cross sections are in fact only available in a homogenized form on a relatively coarse grid formed by the fuel assemblies (see figure 2.2 and remark 2.3) with 18 z-layers. In the following this grid will be referred to as **grid 1** (see figure 6.1). Grids that are obtained by 1 and 2 uniform refinements of grid 1 in each coordinate direction will be called **grid 2** and **grid 3**, respectively (see figure 6.1). For the finer grids (grid 2 and grid 3) it is necessary to interpolate the data.

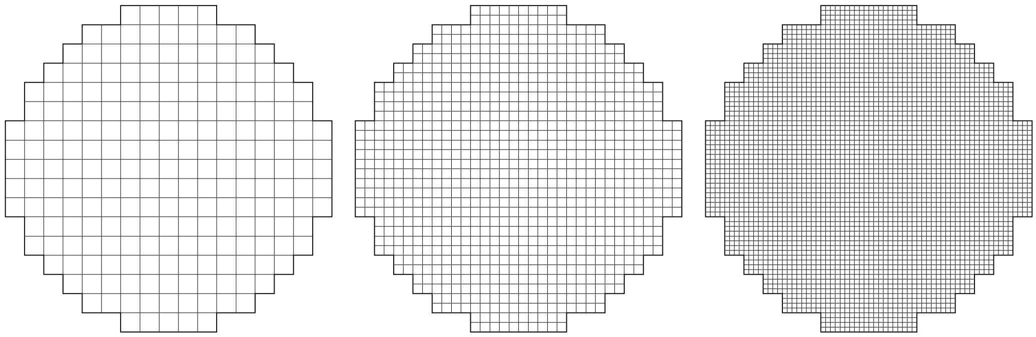


Figure 6.1: Axial cut of grid 1, 2 and 3 for examples 1 and 2

For the numerical tests there was in fact data for three typical reactor simulation problems available. The data for all three problems was given on the grid formed by the fuel assemblies (grid 1) with 18 z-layers (15 for example 3), and considered only two energy groups, i.e. $G = 2$.

6.1.1 Example 1: Boron Dilution

The first example is a Boron dilution case, which may take place as a consequence of a small break loss coolant accident. Such an accident causes an immediate shut-down of the core. The coolant that is boronated to increase the absorption cross section, vaporizes due to the reduced pressure. However the steam that forms in the upper plenum of the reactor has a very low Boron concentration. It is transported to the steam generators where it recondenses. Due to the establishment of natural circulation conditions within the primary coolant system, the subsequent condensate is swept back into the coolant channels of the core and reduces the Boron concentration in the coolant.
The consequence of the Boron dilution is a higher reactivity in the reactor core and subsequently an increase in the fuel temperature. The reactor power eventually stabilizes because of physical feedback mechanisms within the reactor: increases in fuel temperature cause a decrease in coolant density and subsequently a reduction of the core reactivity. Three-dimensional simulation is necessary, as the dilution is not necessarily uniform and produces localized perturbations of the neutronic and thermalhydraulic core parameters.
The radial geometry of the reactor core for this example is shown in figure 6.1 or figure 2.2. In the interior, each box of grid 1 corresponds to a fuel assembly, the boundary boxes correspond to the reflector cells (compare figure 2.2). The width of the boxes is 21.6 cm. The width of the reactor core is therefore 367.2 cm. Axially the reactor core is divided into 18 layers, 16 for the fuel assemblies and 2 for the upper and lower axial reflector. Accordingly the total number of cells (fuel and reflector) is 3978. The hight of the entire reactor core is 427.3 cm.

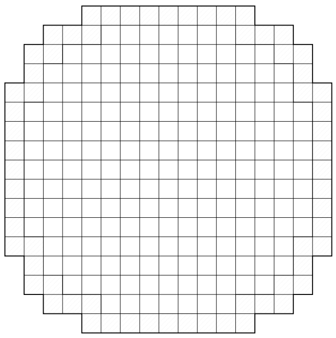
6.1.2 Example 2: Ejection of a control assembly

The second 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) in Finnemann/Galati [12].
The problem that is simulated is the ejection of a control assembly from the core, which may occur as a consequence of the rupture of the drive mechanism casing located on the reactor pressure vessel top. This event can produce significant, well localized perturbations of the neutronic and thermalhydraulic core parameters, without exceeding the safety margins. Hence, a realistic standard reactor situation is

given, that efficiently utilizes the neutronics model.
The multiplication constant $\lambda_0 = 1.0$. The time for the ejection of the control assembly is 0.1 seconds. The core geometry is the same as in example 1.

6.1.3 Example 3: Insertion of a control rod

Finally the third example deals with the insertion of a control rod into a supercritical reactor with a multiplication constant of $\lambda_0 \approx 1.04$. Starting from a configuration where all the control rods are out, a single rod falls into the core and stops at a height of 121.3 cm. The time for the insertion of the control rod is 1 second. The initial flux density distribution of the "fast" neutrons $\phi_1(\vec{x})$ for this example, as derived in Appendix A, is given in figure A.2.
The core geometry differs slightly from the geometry in examples 1 and 2. The axial cut of the grid formed by the fuel assemblies is



The boundary boxes represent the reflector cells again. The width of the fuel assemblies is 23.1 cm, the width of the entire core is 392.7 cm. Axially the core is divided into 15 layers, 13 for the fuel assemblies and 2 for the upper and lower axial reflector again. The height of the core is 451.6 cm. The total number of cells is 3855. However the grid will be referred to as grid 1 once more.

In the following, these examples will be simply referred to as examples 1, 2 and 3.

The discretization of the multigroup neutron diffusion equations (2.26, 2.27) was carried out as presented in chapter 3. The initial conditions for the transient calculations were taken as the solutions of the quasi-stationary problem (2.28). The discretization and solution of this generalized eigenvalue problem together with numerical results is outlined in appendix A. The parameter γ for the time discretization with TR-BDF2 was always chosen to be $\gamma = 1/2$.

6.2 Multi-Grid versus Preconditioned Bi-CGStab

The arising linear equation systems (LESs) in each time step were solved with the two iterative techniques described in chapter 4: Multi-Grid (algorithm 4.4) and Left-preconditioned Bi-CGSTab (algorithm 4.2).

The results of the transient calculations can only be as accurate as the initial solution. To obtain representative results, the quasi-stationary problem has therefore to be solved very accurately, i.e. the tolerance ε for the convergence test of algorithm A.2 has to be very small (compare section A.3). For a fairer comparison of the solution methods, the same initial solutions were taken and the accuracy was checked with the defect test (4.31). The tolerance was $\varepsilon_R = 10^{-6}$.

The two methods were applied to the LESs arising from the discretization of example 1, 2 and 3, on grid 1, 2 and 3 respectively. The number of unknowns N of the LES is about 55,000 on grid 1, about 450,000 on grid 2, and about 3.5 million on grid 3, i.e. a factor 8 from a grid to the next finer one. Finer grids than grid 3 were not possible because of a lack of computer memory. For the Multi-Grid Method (MGM) the coarsest grid was always chosen to be grid 1. It was therefore only possible to run the MGM on grid 2 (where it reduced to a Two-Grid Method) and on grid 3.

The components for the MG algorithm 4.4 were chosen as described in section 4.3.3. The coarse grid system on grid 1 was solved with preconditioned Bi-CGStab, where only few iterations were necessary to get a good coarse grid correction (i.e. defect test with $\varepsilon_R = 0.1$ or 0.5). For the smoothing on the finer grids the Block-SOR method as per algorithm 4.1 was used. The relaxation parameter ω (in Block-SOR) was chosen to be 1.0. The number of smoothing steps was $\nu^{pre} = 3$ for the pre- and $\nu^{post} = 2$ for the post-smoothing. For the "Three-Grid Method" on grid 3, the parameter γ for grid 2 was chosen to be $\gamma_2 = 2$, i.e. the W-cycle was employed.

The Bi-CGStab method was used in its left-preconditioned form, as presented in algorithm 4.2. For the preconditioning, two steps of the Block-SOR method as per algorithm 4.1 were employed. The relaxation parameter ω was $\omega = 1.2$ on grid 1 and $\omega = 1.0$ on grid 2 and 3.

6.2.1 Numerical results: Asymptotical behaviour

The execution times for one time step of each example can be found in table 6.1. In each time step two LESs have to be solved, one for each partial step (trapezoidal and BDF(2)). The numbers in the brackets indicate the total number of iterations needed for the solution of both LESs.

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)

Grid	MG (3/2)	P–Bi-CGStab
1	–	18 s (38)
2	155 s (27)	337 s (73)
3	1818 s (40)	5631 s (148)

Grid	MG (3/2)	P–Bi-CGStab
1	–	20 s (42)
2	254 s (41)	413 s (85)
3	1742 s (35)	7613 s (183)

Table 6.1: Execution time and iterations (in brackets) for one time step dt: example 1, $dt = 0.01$ s (top); example 2, $dt = 0.005$ s (middle); example 3, $dt = 0.01$ s (bottom)

The results of example 3 verify the asymptotical optimality of MGM: When the number of unknowns N grows, the number of iterations (in brackets) does not grow for a fixed tolerance ε_R, it even decreases. The execution time shows the expected $O(N)$ behaviour and only increases by a factor of 7. The performance of the other two examples is similar on grid 3, but on grid 2 it is far better. This is the reason for the bad asymptotical behaviour. The number of iterations increases by a factor of 1.5 from grid 2 to grid 3, and the execution time by a factor of 12. The rate of convergence lies between 0.3 and 0.5 on grid 2 and on grid 3 depending on the example and on the size of the time step.

The preconditioned Bi-CGStab method also shows the behaviour that was expected. When the number of unknowns grows, the number of iterations increases as well. It increases in fact by a factor of about 2 for a fixed tolerance $\varepsilon_R = 10^{-6}$, which is a good result for a preconditioned Krylov-Subspace method applied to a three-dimensional problem. The rate of convergence is 0.7 on grid 2 and 0.85 on grid 3.

A comparison of the two methods turns out in favour of the MGM: The execution times on grid 2 differ by a factor of 2.9 for example 1, by a factor of 2.2 for example 2 and by a factor of 1.6 for example 3. On grid 3 they differ by a factor of 3.9 for example 1, by a factor of 3.1 for example 2 and by a factor of 4.4 for example 3. The finer the grid becomes, the more MGM outperforms preconditioned Bi-CGStab.

For the smaller problem on grid 1 ($N \approx 55000$) it was only possible to apply the preconditioned Bi-CGStab method (rate of convergence: 0.5). MGM was not possible, as the difficult geometry of a reactor makes it hard to introduce coarser grids than grid 1. 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 also be possible to apply MGM on grid 1.
The application of *Algebraic Multi-Grid (AMG)* (see [15], [23], [29], or [30]) on grid 1, would be a possibility that avoids the introduction of coarser geometrical grids. In combination with MGM (i.e. as the Coarse-Grid Solver) AMG could also be used in a *Hybride Multi-Grid Method* for the finer grids.

6.2.2 Numerical results: Transient behaviour

As a comparison of the two methods over a whole period of a reactor simulation and not only for one single time step, 0.3 seconds of example 1 (from t = 500.05 to t = 500.35) were simulated using both methods on grid 2. The tolerance for the defect test was again $\varepsilon_R = 10^{-6}$. The results are presented in table 6.2. The total execution time, including the time needed for the error estimation (see below) and the Input/Output (I/O), is given in the last line.

The initial time step was chosen to be $dt = 0.01$ s, the following time steps were determined automatically with the scheme described in section 3.2.1 for the TR-BDF2 method. To avoid instabilities, only the unknowns ϕ_g^m were used to get an estimate τ of the integration error with formula (3.20).

The magnitude of the neutron flux density ϕ_g^m differs significantly for the two energy groups $g = 1$ and $g = 2$. The parameter δ in (3.22) was therefore chosen differently

t	dt	MG (3/2)
500.060	0.010	154 s (25)
500.070	0.010	164 s (27)
500.077	0.007	181 s (31)
500.090	0.013	200 s (33)
500.100	0.010	176 s (29)
500.108	0.008	135 s (23)
500.122	0.014	155 s (26)
500.150	0.028	166 s (27)
500.167	0.017	174 s (28)
500.200	0.033	182 s (29)
500.225	0.025	165 s (27)
500.250	0.025	174 s (28)
500.267	0.017	162 s (26)
500.300	0.033	175 s (28)
500.325	0.025	165 s (27)
500.350	0.025	173 s (28)

t	dt	P–Bi-CGStab
500.060	0.010	422 s (89)
500.070	0.010	423 s (91)
500.077	0.007	361 s (79)
500.090	0.013	396 s (85)
500.100	0.010	377 s (81)
500.108	0.008	375 s (82)
500.122	0.014	420 s (92)
500.150	0.028	445 s (96)
500.167	0.017	402 s (85)
500.200	0.033	427 s (92)
500.225	0.025	423 s (93)
500.250	0.025	427 s (92)
500.267	0.017	407 s (86)
500.300	0.033	450 s (97)
500.325	0.025	423 s (93)
500.350	0.025	432 s (93)

Table 6.2: Simulation of 0.3 s of example 1 on grid 2 (time (t), time step (dt), execution time and iterations (in brackets))

for each energy group g, namely as

$$\delta_g = 10^{-4} \cdot \max_m(\phi_g^m), \quad g = 1, 2.$$

In figure 6.2 the relative norm of the error estimate ($error = \|\tau\|_2 / \|\phi\|_2$) is plotted for each time step. It can be seen that there is no significant difference between the two methods.

The execution time for one time step is on average 169 seconds for MGM and 413 seconds for preconditioned Bi-CGStab. That is a factor of 2.45. The factor 2.4 for the total execution time is only slightly smaller. In figures 6.3 and 6.4 the iterations and the execution times of the two methods are plotted. The advantage of MGM is obvious.

A comparison of the preconditioned Bi-CGStab method with the method used in PANBOX2 [3] for a transient simulation on grid 1 can be found in section 6.4.

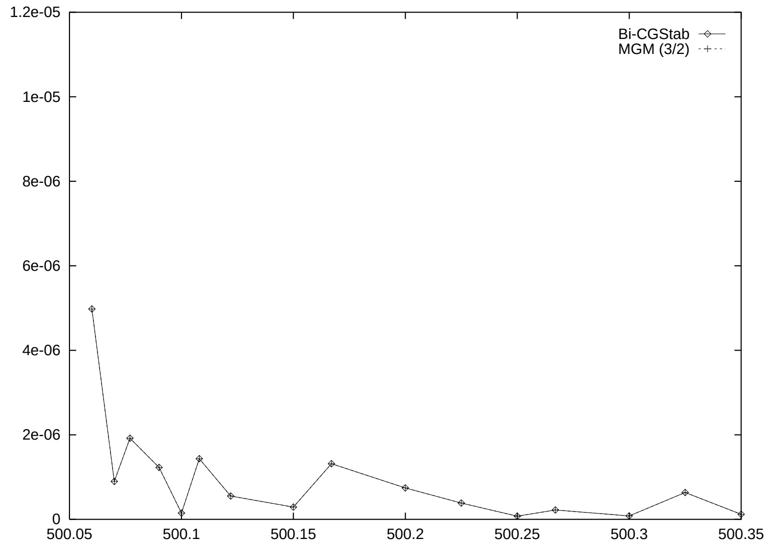


Figure 6.2: Error estimate for each time step (example 1, grid 2)

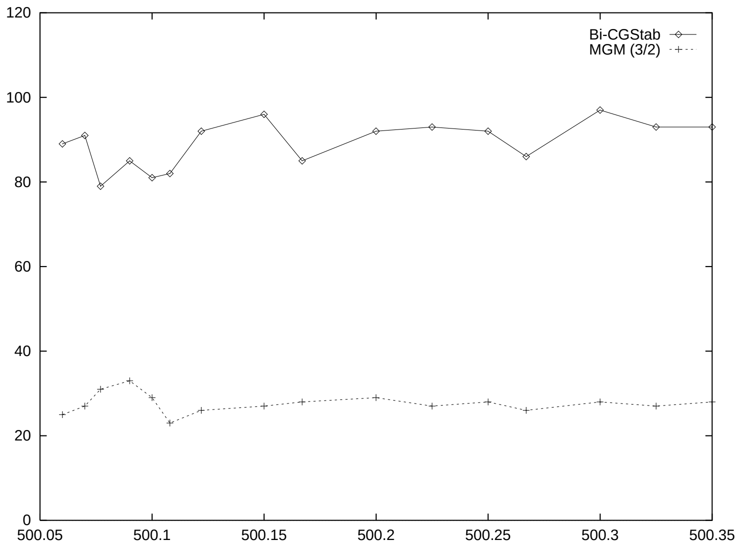


Figure 6.3: Number of iterations for each time step (example 1, grid 2)

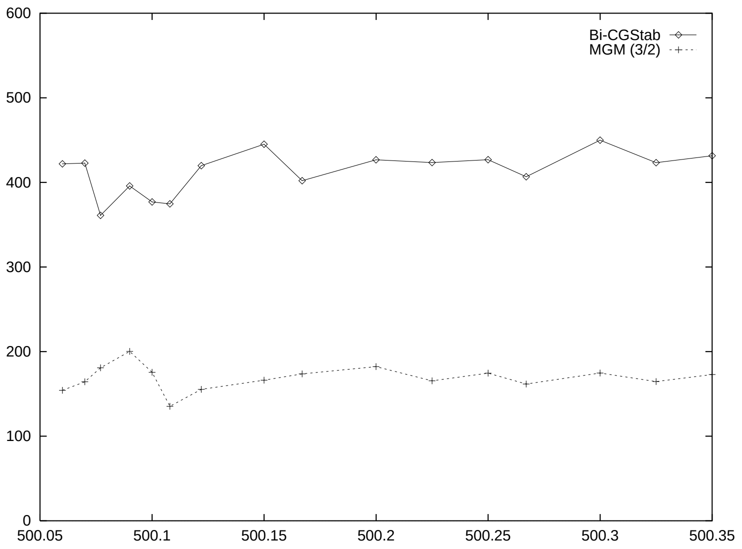


Figure 6.4: Execution time (in seconds) for each time step (example 1, grid 2)

6.3 Parallel results

The parallelization of the two methods was carried out, as described in section 5.1, by partitioning the domain. The implementation on the shared memory machine was based on *threads* (subprocesses). To create and control the threads, **m**-routines were used. The implementation was extensively described in section 5.2.

The parallel algorithms are not equivalent to the serial ones, since the Block-SOR method that is used in both MGM and the preconditioned Bi-CGStab method had to be modified (compare section 5.1.2.2). The consequence of this modification (i.e. the application of Jacobi-type iterations on the subdomain boundaries) were slightly different results. When different partitionings of the domain (i.e. different numbers of processes) were used, it was possible that the numbers of iterations differed as well (compare the results below).

The components and parameters of MGM and preconditioned Bi-CGStab were chosen as in the serial case and are specified at the beginning of section 6.2.

The speedup is calculated from the formula

$$\text{Speedup} = \frac{\text{Serial Execution Time}}{\text{Parallel Execution Time}}.$$

6.3.1 Numerical results: Speedup

The execution times for one time step of example 1 on grid 1 ($N \approx 55000$) and grid 2 ($N \approx 450000$) are given in table 6.3. The speedup graphs are shown in figure 6.5.

Processors	MG (3/2) (grid 2)	Bi-CGStab (grid 2)	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)
3	43.8 s (22)	95.6 s (65)	6.62 s (42)
4	32.3 s (23)	78.0 s (71)	5.09 s (43)
5	26.5 s (23)	62.4 s (70)	3.93 s (43)
6	21.1 s (22)	52.8 s (73)	3.45 s (44)
7	17.9 s (22)	46.2 s (76)	2.78 s (41)
8	16.1 s (23)	38.7 s (74)	2.43 s (41)
9	14.2 s (23)	34.5 s (73)	2.24 s (42)
10	12.9 s (23)	31.5 s (73)	2.07 s (43)
11	11.9 s (23)	25.7 s (70)	2.04 s (43)
12	10.5 s (22)	22.9 s (69)	1.87 s (43)

Table 6.3: One time step ($dt = 0.01$ s) of example 1: Execution time and iterations

The speedup for both methods on grid 2 is superlinear. The Bi-CGStab method for example runs almost 16 times faster on 12 processors than on 1 processor. Even the speedup per iteration is superlinear (14.5 on 12 processors for Bi-CGStab).
The reasons for such unusual behaviour are *cache-effects*: The cache of each processor has a particular size (here: 4 MByte). If the data that is needed for the execution of one loop fits into the cache, the loop can be executed faster than a loop, where the data has to be reloaded into the cache. It is therefore possible and likely that smaller loops in the parallel case can be executed faster than the large loops in the serial case. Nevertheless it is clear that the speedup for both methods on grid 2 is optimal.
The speedup for preconditioned Bi-CGStab on grid 1 is also optimal if less than 10 processors are used. For 10 or more processors the speedup factor is not quite optimal anymore. The maximum speedup that is possible with 12 processors on grid 1 is 10.4.

The larger the LES is, the better the speedup results are. Therefore it is not nec-

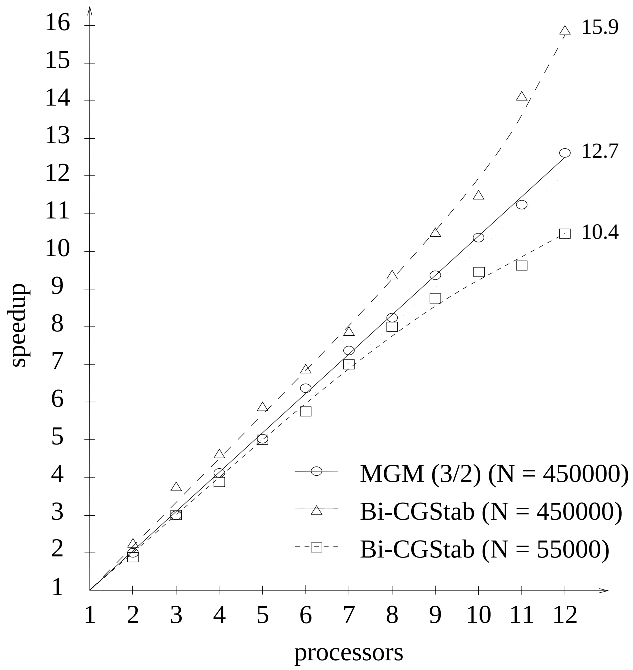


Figure 6.5: One time step ($dt = 0.01$ s) of example 1: Speedup

essary to perform extensive tests on grid 3. To show that this thesis is correct, example 1 was also calculated on grid 3 using 10 processors. The results are given in table 6.4. Because of the larger problem size there are no cache effects on grid 3.

Processors	MG (3/2)	P–Bi-CGStab
1 10	1672 s (36) 161 s (35)	6546 s (169) 651 s (162)
Speedup	10.4	10.1

Table 6.4: Execution time and iterations for one time step ($dt = 0.01$ s) of example 1 on grid 3

There is also no dependence of the speedup behaviour on the data. Examples 2 and 3 were therefore only run on 10 processors (for the results see table 6.5).

Processors	MG (3/2) (grid 2)	Bi-CGStab (grid 2)	Bi-CGStab (grid 1)
1	155 s (27)	337 s (73)	18 s (38)
10	14.1 s (27)	29.2 s (74)	1.8 s (35)
Speedup	11	11.5	10

Processors	MG (3/2) (grid 2)	Bi-CGStab (grid 2)	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)
Speedup	10.3	10.1	9.5

Table 6.5: Execution time and iterations for one time step dt on grid 2: example 2, $dt = 0.005$ s (top); example 3, $dt = 0.01$ s (bottom)

The comparison of the speedup results with the results reported in the literature (e.g. [7], [10], [11], [20], [24], [32], [34], [36]) is difficult, as they are all obtained on different computer systems, for different examples and for different problem sizes. However the results can be perfectly compared to the results in Hammer, Paffrath, et al. [19], as their results were obtained on the exact same computer for example 1 with PANBOX2 [3]. The grids that they used were grid 1 (referred to as *standard*) and a grid with the same axial cut but 72 instead of 18 z-layers (referred to as *large*).

They use a method of higher order from the same family of nodal methods as NEM-M0, called NEM-M2 (developed in Finnemann [8], see also remark 3.2), for the spatial discretization, and a heuristic exponential integration scheme related to the implicit backward-Euler for the time discretization. The arising linear equation systems are solved with a multilevel method that is based on a Red-Black Gauss-Seidel relaxation on the finest grid and Coarse-Mesh Rebalancing (CMR) techniques on the coarser grids.
In [19], different types of parallelization of the method are given, including a machine specific parallelization with threads that is similar to the parallelization used for MGM and Bi-CGStab.

For the standard problem (i.e grid 1) the speedup of their method is 2.7 for 4 proces-

sors, 3.5 for 6 and 3.7 for 8 (see [19, Fig. 7] or also figure 6.10). The corresponding speedup factors of the preconditioned Bi-CGStab on grid 1 (see figure 6.5) are 3.8, 5.7 and 8.0. The results for the standard problem in [19] imply that even for a higher number of processors only a speedup factor of about 4 is possible. Bi-CGStab, on the other hand (compare figure 6.5), allows speedup factors of more than 10 for higher numbers of processors.
The large problem in [19] can not be compared one-to-one with the results on grid 2 in this dissertation, as their problem is only half the size of the problem on grid 2. Their results for the large problem are nevertheless almost optimal regardlessly. The speedup factors are 4.1 for 4 processors, 5.8 for 6 and 7.3 for 8 and therefore only a little bit smaller than the corresponding values for MGM and Bi-CGStab, as presented in figure 6.5.

6.3.2 Numerical results: Transient behaviour

To conclude the presentation of the parallel numerical results, the same 0.3 s of example 1 as in section 6.2.2 (from $t = 500.05$ to $t = 500.35$) are simulated on grid 2. The results are compared with the serial results from section 6.2.2. In figure 6.6 the error estimate of the integration scheme is given for each time step. In figure 6.7 the iterations are given, and in figure 6.8 the execution times.

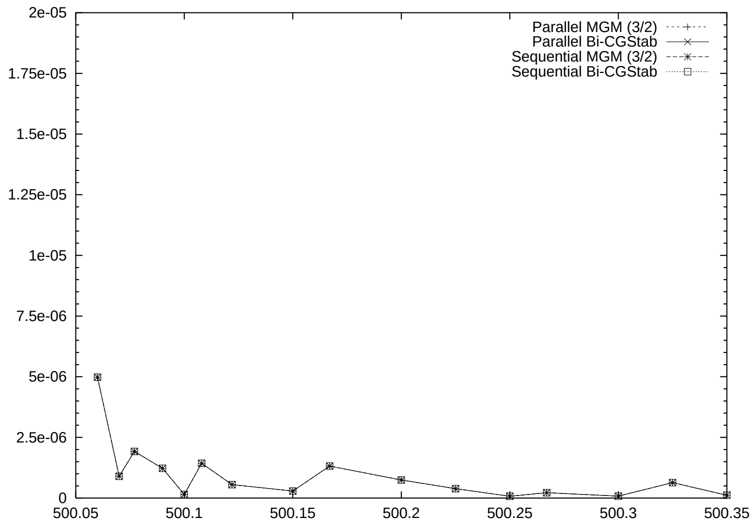


Figure 6.6: Error estimate (example 1, grid 2, parallel: 9 processors)

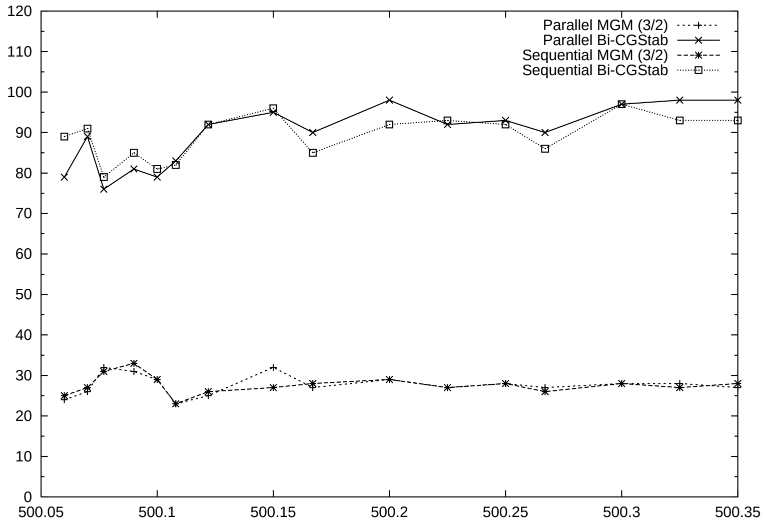


Figure 6.7: Number of iterations (example 1, grid 2, parallel: 9 processors)

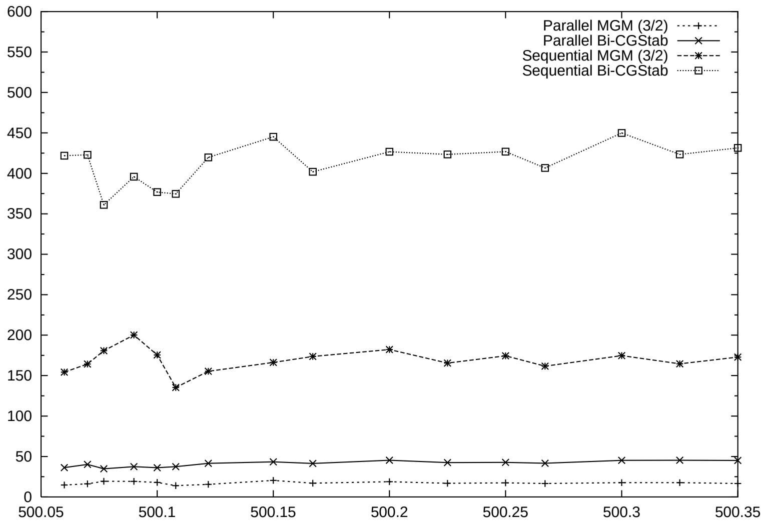


Figure 6.8: Execution time (in seconds) (example 1, grid 2, parallel: 9 processors)

The error estimate in figure 6.6 shows again no significant difference between the methods. The size of the time steps was therefore chosen equally for all methods. The modification of the Block-SOR method in the parallel case resulted in different numbers of iterations in each time step (compare figure 6.7). However the number of iterations in the parallel case is not in general higher than in the serial case, but it can even decrease sometimes. The modification of the Block-SOR method did therefore not effect the convergence of the algorithms significantly, at least for less than 12 subdomains.
The execution times for the parallel methods in figure 6.8 favour the MGM again. With an average execution time of 17.3 seconds for MGM and 41.1 seconds for preconditioned Bi-CGStab, the factor 2.4 in favour of MGM is only slightly smaller than in the serial case (factor: 2.45). The total execution time for the 16 time steps (including the error estimation and the I/O) is 319 seconds for MGM and 691 seconds for Bi-CGStab. MGM therefore supercedes preconditioned Bi-CGStab by a factor of 2.2 over the whole simulation period.

The total execution times for the other partitionings (i.e. numbers of processors) are listed in table 6.6. The corresponding speedup graphs are plotted in figure 6.9.

Processors	P–Bi-CGStab	MG (3/2)
1	6663 s	2763 s
2	3283 s	1492 s
3	2188 s	974 s
4	1601 s	702 s
5	1324 s	586 s
6	1065 s	487 s
7	878 s	411 s
8	753 s	357 s
9	691 s	319 s
10	655 s	296 s
11	542 s	274 s
12	494 s	249 s

Table 6.6: Total execution time for the simulation of 0.3 s of example 1 on grid 2

The total execution times include the error estimation and the I/O, which was executed serially by a “master” process. The speedup is therefore lower than for a

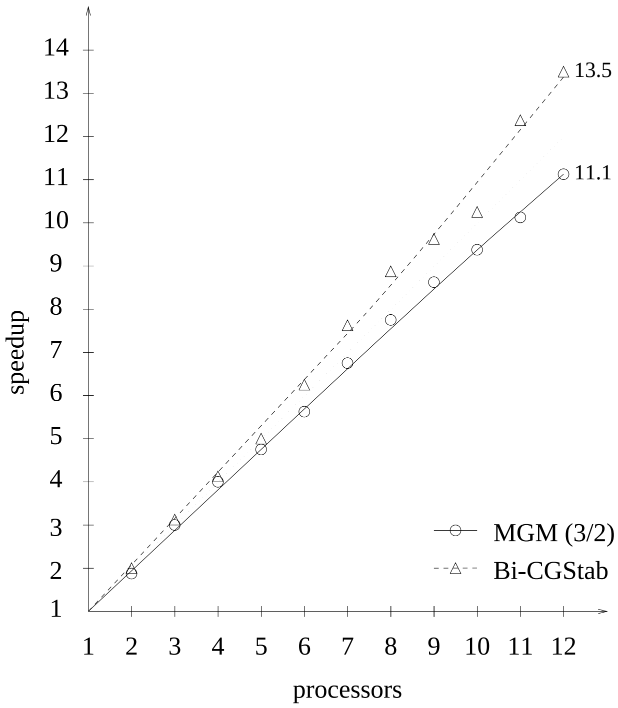


Figure 6.9: Speedup for the simulation of 0.3 s of example 1 on grid 2

single time step. It is nevertheless still superlinear for preconditioned Bi-CGStab and almost linear for MGM. The maximum speedup that was achieved on 12 processors was 11.1 for MGM and 13.5 for Bi-CGStab. In spite of the better speedup of the preconditioned Bi-CGStab method, MGM is still more than 2 times faster, even for 12 processors.

6.3.3 Discussion of the results

The two solution methods MGM, algorithm 4.4, and Left-preconditioned Bi-CGSTab, algorithm 4.2, that were compared in section 6.2 for the serial case were parallelized using machine specific parallelization on a shared memory multiprocessor computer. The components of the two methods are chosen as in the serial case.
The parallel methods are again applied to the three test problems (example 1, 2 and 3) on three different grids (grid 1: $N \approx 55000$, grid 2: $N \approx 450000$ and grid 3: $N \approx 3500000$), the speedup for different partitionings (i.e. different numbers of processors) is determined and the two methods are compared again.

The speedup does not depend on the choice of the example, but it does depend on the problem size. If less than 10 processors are used the speedup for one time step on grid 1 is optimal for preconditioned Bi-CGStab. For more processors the speedup factor reduces slightly, e.g. to 10.4 for 12 processors.
On grid 2 and on grid 3 the speedup for one time step is optimal for preconditioned Bi-CGStab and MGM, even for more than 10 processors. Due to a different iteration count and due to cache effects the speedup is even superlinear, e.g. for example 1 on grid 2 with 12 processors: 12.7 for MGM, and 15.9 (!) for Bi-CGStab.

The modification of the Block-SOR method in the parallel case had no serious effect on the convergence of the two methods. The iteration count differs to the serial case, but it is not in general higher.
A comparison of the execution times shows that even in the parallel case the MGM is more than 2 times faster than preconditioned Bi-CGStab on grid 2 and almost 4 times faster on grid 3. As in the serial case, it would be interesting to introduce coarser grids than grid 1 and investigate the speedup that can be achieved with MGM on grid 1.

Especially for the small (standard) problem on grid 1 using the preconditioned Bi-CGStab method, the speedup results (3.8 for 4 processors, 5.7 for 6, and 8.0 for 8) are better than the results reported in literature. Using a different method but the same hard- and software for the parallelization, Hammer, Paffrath, et al. [19] only achieve a speedup of 2.7 for 4 processors, a speedup of 3.5 for 6 processors, and a speedup of 3.7 for 8 processors.
The speedup results for the larger problems are only a little bit better than the results in [19]. The speedup factors for the "large" problem in [19] are 4.1 for 4 processors, 5.8 for 6, and 7.3 for 8. The corresponding factors on grid 2 are 4.7, 6.9 and 9.4 for Bi-CGStab, and 4.1, 6.3, and 8.3 for MGM, but it has to be noted that the "large" problem in [19] is only half the size of the problem on grid 2.

The speedup factors for the small (standard) problem on grid 1 are far better than the ones reported in literature and the speedup factors for the larger problems are optimal. It can therefore be concluded that the parallelization of the solution methods was very successful.

6.4 Comparison of preconditioned Bi-CGStab to PANBOX2 on the grid formed by the fuel assemblies

The best speedup factor for a solution method is useless if the method in the serial case is not competitive to the method that it is compared to. For the comparison of the speedup results in the previous section, the results achieved with PANBOX (in Hammer, Paffrath, et al. [19]) were used. In its serial version PANBOX was available for numerical tests on the SGI Power Challenge. For the comparison of the serial performances, 10 time steps of the control rod ejection benchmark problem, example 2, on the grid formed by the fuel assemblies (grid 1) were calculated.

In PANBOX the multigroup neutron diffusion equations are usually discretized using NEM-M2 (as in [19]), but they can also be discretized using NEM-M0 (in equivalence to the discretization in this dissertation). For the time discretization a heuristic exponential integration scheme related to the implicit backward-Euler is used. It is a first order integration method. The method for the solution of the arising linear equation systems is a multilevel method that is based on a Red-Black Gauss-Seidel relaxation on the finest grid and Coarse-Mesh Rebalancing (CMR) techniques on the coarser grids. The convergence test in PANBOX was previously described in section 4.4 and is given by equation (4.35). The tolerance ε_P in PANBOX is 10^{-5}.

The pure execution time of the solution method in PANBOX, for 10 time steps of example 2 on grid 1 is 53.5 seconds for NEM-M0 and 125 seconds for NEM-M2 (achieved on the SGI Power Challenge). The execution time for a parallel run of example 2 was not available unfortunately. However with the results from [19], as presented in figure 6.10, we can estimate that the maximum speedup possible with 12 processors is between 4 and 4.5. For 10 time steps of example 2 on grid 1 the best possible parallel execution time of the multilevel method in PANBOX will therefore lie between 12 and 13.5 seconds.

Because of the heuristic exponential ansatz for the time discretization method in PANBOX, the local truncation error is smaller than for the "normal" implicit backward-Euler. However the integration method is of first order and therefore asymptotically less accurate than the second-order TR-BDF2 method that was used throughout this dissertation. For a comparison of the efficiency of preconditioned Bi-CGStab with the efficiency of the multilevel method in PANBOX it is therefore better to use both integration methods: Implicit backward-Euler (solution of one LES per time step) and TR-BDF2 (solution of two LESs per time step). For a fairer comparison it is also necessary to use the same convergence test (4.35) with the same tolerance $\varepsilon_P = 10^{-5}$ as in PANBOX.

The results obtained with the Bi-CGStab method, preconditioned with two steps of Block-SOR ($\omega = 1.2$), for the same 10 time steps of example 2 on grid 1 are given in table 6.7. They are compared to the results obtained with the NEM-M0 version of PANBOX. The parallel performance of PANBOX is estimated from the speedup results in [19] as presented in figure 6.10.

Processors	PANBOX (NEM-M0)	Bi-CGSt. (Im.Eul.)	Bi-CGSt. (TR-BDF2)
1	53.5 s	115 s	175 s
2	*∼ 29 s*	53 s	75 s
4	*∼ 19 s*	28 s	41 s
6	*∼ 15 s*	19 s	28 s
8	*∼ 14 s*	15 s	22 s
10	*∼ 12.5 – 14 s*	13 s	19 s
12	*∼ 12 – 13.5 s*	11 s	16 s

Table 6.7: Pure execution time of the solution method for 10 time steps of ex.2

If the implicit backward-Euler method is used for the time discretization, Bi-CGStab needs 115 seconds. If TR-BDF2 is used, it needs 175 seconds. The serial version of Bi-CGStab is about 2 times slower than PANBOX for implicit backward-Euler, and about 3 times slower for TR-BDF2. If the described time discretization methods are used for a fixed time step size, Bi-CGStab is unable to compete with the multilevel method in PANBOX in the serial case.
Without further investigations it is hard to say, whether the multilevel method in PANBOX is faster than preconditioned Bi-CGStab, or whether the special problem-dependent ansatz for the time discretization in PANBOX makes the difference. Differential equation systems can generally be solved faster when a special problem-dependent ansatz is used in the integration method.

The factors 2 and 3 in the serial case are acceptable though, as the parallel results are far better than the ones that are possible with PANBOX. Figure 6.11 shows how Bi-CGStab catches up with the method used in PANBOX, when the number of processors is increased. On 12 processors Bi-CGStab needs 11 and 16 seconds respectively to solve the 10 time steps of example 2. For the implicit backward-Euler

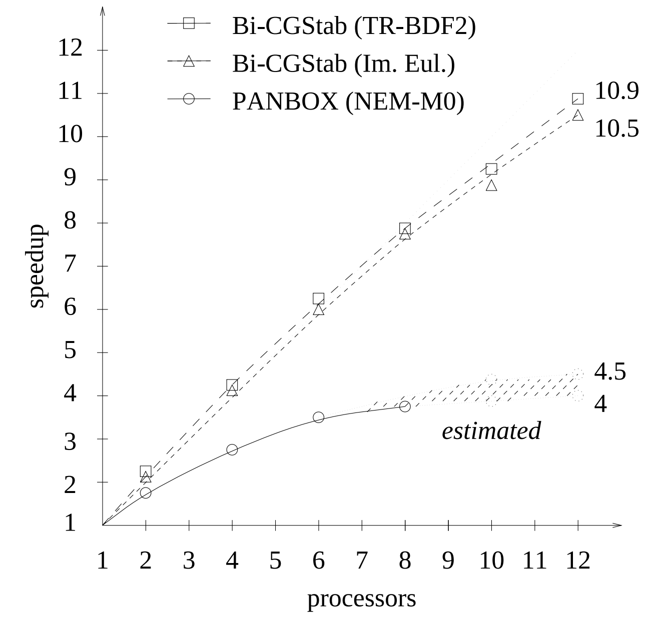


Figure 6.10: Speedup for 10 time steps of example 2 (PANBOX: from [19, Fig. 7])

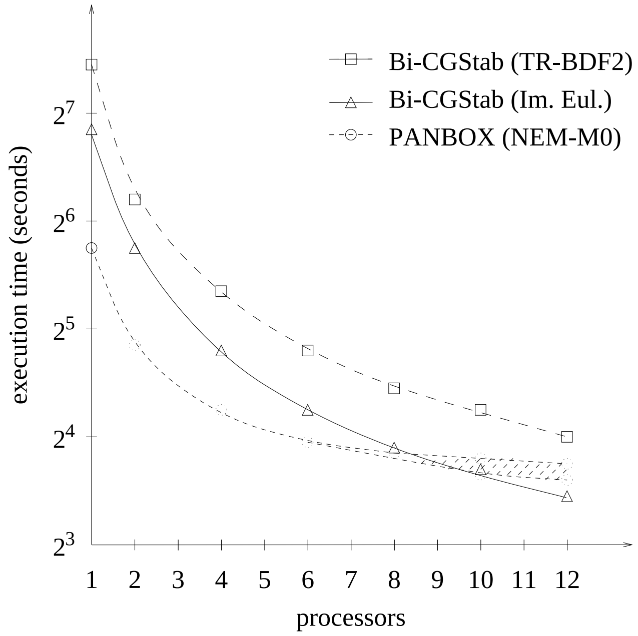


Figure 6.11: Execution time for 10 time steps of example 2 (PANBOX estimated)

version, the parallel results on 12 processors are therefore already about 10–20 % better than the estimated parallel execution times of PANBOX (compare figure 6.11). Figures 6.10 and 6.11 moreover allow the conclusion that for more than 12 processors even the TR-BDF2 version will supercede the estimated results of PANBOX.

Unfortunately it is not possible to compare the results achieved with MGM to the results achieved with the multilevel method in PANBOX. The question, whether MGM is competitive to the multilevel method in PANBOX on grid 1 or not, remains unanswered and has still to be investigated.
MGM applied to the equations arising in the TR-BDF2 case on grid 2 needs 1250 seconds for the solution of the 10 time steps of example 2 on 1 processor and 96 seconds on 12 processors.
A direct comparison of these results with any of the results obtained with PANBOX is not possible. The higher-order NEM-M2 version of PANBOX on the coarser grid 1 needs for example 125 seconds to execute the 10 time steps of example 2 on 1 processor (pure execution time of the multilevel method). On the basis of the speedup results and the serial execution times of Multi-Grid and PANBOX (NEM-M2), it is possible to speculate that for a higher number of available processors the Multi-Grid method on grid 2 will become more efficient than PANBOX (NEM-M2) on grid 1.

Finally it can be concluded that the parallel results with MGM and Bi-CGStab are very promising. On the grid formed by the fuel assemblies it is possible to obtain better execution times with Bi-CGStab than with other parallel simulation methods that are currently used, like PANBOX2 [3]. On the finer grids MGM performs very well: It is more than twice as good as Bi-CGStab, it allows an optimal parallelization, even for a large number of processors, and the performance is almost asymptotically optimal for an increasing number of unknowns ($O(N)$).

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.
A full coupling of the neutronic module with the thermalhydraulic module (e.g. in PANBOX) would also be very interesting. In the moment the neutronic module is calculated on its own and the data is taken from a file, but to study the influence of feedback effects on the convergence behaviour of the solution methods, a full coupling is necessary.

Appendix A

Solution of the stationary problem

To find an initial condition for the transient calculation, a quasi-stationary calculation has first to be performed. It has already been mentioned in section 2.6 that in the stationary case the *generalized eigenvalue problem* (2.28) has to be solved. In particular, the dominant eigenvalue λ_0 (i.e. $|\lambda_0| > |\lambda|$ for all eigenvalues $\lambda \neq \lambda_0$) and the corresponding eigenfunction have to be found.

Since the solution of the stationary problem will be used as the initial solution for the transient problem, the spatial discretization has to be chosen as in the transient case, i.e. NEM-M0. In the first section the discretization is therefore described only briefly. The result is a generalized linear eigenvalue problem.
To find the dominant eigenvalue and the corresponding eigenfunction for a generalized linear eigenvalue problem a variant of the *Inverse Iteration* proposed by Schmid [33] is used. The method is called *Inverse Correction.* A comparison with the inverse iteration, especially for the neutronics problem, can be found in Schmid [33, ch. 7]. The numerical results obtained with this method for example 3 from chapter 6 are given in the final section of the appendix.

A.1 Discretization using NEM-M0

The discretization of partial differential equations (PDEs) with the NEM-M0 method has already been described in full detail in section 3.1. Therefore the discretization of the stationary system (2.28) will only be discussed briefly. As in the transient case the equation system (2.28) is transformed into a system of PDEs of first order

$$\left.\begin{aligned} div\ \vec{j}_g(\vec{x}) + \Big(\Sigma_{ag}(\vec{x}) + \sum_{g'} \Sigma_{gg'}(\vec{x})\Big)\phi_g(\vec{x})\ - & \\ \sum_{g'} \Sigma_{g'g}(\vec{x})\phi_{g'}(\vec{x}) \ = \ \frac{1}{\lambda}\Big(\sum_j \bar{\chi}_g^j \sum_{g'} \nu\Sigma_{fg'}^j(\vec{x})\phi_{g'}(\vec{x})\Big) & \end{aligned}\right\} g = 1...G \quad \text{(A.1)}$$

$$\vec{j}_g(\vec{x}) + D_g(\vec{x})\, grad\ \phi_g(\vec{x}) \ = \ \vec{0} \qquad g = 1...G \quad \text{(A.2)}$$

The unknowns are the neutron flux ϕ_g and the current density $\vec{j}_g$ in each energy group g as well as the eigenvalue λ. The description of all the coefficient functions and parameters of the system can again be found in chapter 2. The boundary conditions are as in the transient case: (3.4), (3.5) and (3.6). All the functions depend on the time t (compare chapter 2). Since a quasi-stationary situation with $t = 0$ is assumed, the dependency on t can be omitted.

By following exactly the same steps as in the discretization process of the transient system, the following discrete linear equation system is obtained

$$\left.\begin{aligned} \left(\Sigma_{ag}^m + \sum_{g'=1}^{G} \Sigma_{gg'}^m + \sum_{u=x,y,z} \frac{2c_{1gu}^m}{a_u^m}\right)\phi_g^m \ - \sum_{g'=1}^{G} \Sigma_{g'g}^m\, \phi_{g'}^m\ - & \\ \sum_{u=x,y,z} \frac{4c_{1gu}^m}{a_u^m}\left(j_{gur}^{m-} + j_{gul}^{m+}\right) \ = \ \frac{1}{\lambda}\left(\sum_{j=1}^{J} \bar{\chi}_g^j \sum_{g'=1}^{G} \nu\Sigma_{fg'}^{jm}\, \phi_{g'}^m\right) & \end{aligned}\right\} g = 1...G \quad \text{(A.3)}$$

$$\left.\begin{aligned} j_{gur}^{m+} - c_{1gu}^m\, \phi_g^m + c_{2gu}^m\, j_{gur}^{m-} + c_{3gu}^m\, j_{gul}^{m+} &= 0 \\ j_{gul}^{m-} - c_{1gu}^m\, \phi_g^m + c_{3gu}^m\, j_{gur}^{m-} + c_{2gu}^m\, j_{gul}^{m+} &= 0 \end{aligned}\right\}\left\{\begin{aligned} g &= 1...G \\ u &= x, y, z \end{aligned}\right. \quad \text{(A.4)}$$

for the boxes $m = 1...M$.
It is a system of $N = M\cdot G\cdot(1+2Dim)$ equations for the $M\cdot G\cdot(1+4Dim)$ unknowns ϕ_g^m and $j_{gus}^{m\pm}$. The $\Sigma_{\cdot g}^m$ are again the box-homogenized cross sections (compare section 3.1). The coefficients $c_{\cdot gu}^m$ are defined as in the transient case. Using the interface condition (3.5), i.e.

$$j_{gur}^{m_1-} = j_{gul}^{m_2-}, \qquad j_{gul}^{m_2+} = j_{gur}^{m_1+} \qquad \text{(A.5)}$$

with two adjacent boxes m_1 and m_2, and the vacuum boundary condition (3.6), the incoming currents $j_{gur}^{m_1-}$ and $j_{gul}^{m_2+}$ can be substituted with the outgoing currents $j_{gul}^{m_2-}$ and $j_{gur}^{m_1+}$ from the neighbouring box.

The resulting generalized linear eigenvalue problem can be written in matrix form:

$$A \cdot \underline{u} = \rho \; F \cdot \underline{u} \qquad \left(\rho := \frac{1}{\lambda}\right) \tag{A.6}$$

In the case of two energy groups, i.e. $G = 2$:

$$\underline{u} = (\phi_1^1, \phi_2^1, j_{1xl}^{1-}, j_{2xl}^{1-}, .., j_{1zr}^{1+}, j_{2zr}^{1+}, \; \dots \;, \phi_1^M, \phi_2^M, j_{1xl}^{M-}, j_{2xl}^{M-}, .., j_{1zr}^{M+}, j_{2zr}^{M+})^T$$

A has the same pattern as in the transient case (see figure 3.2). F has the following pattern

$$F := \left(\begin{array}{ccc} \underbrace{\begin{array}{cccc} * \; * & & & 0 \\ * \; * & & & \\ & 0 & & \\ & & \ddots & \\ 0 & & & 0 \end{array}}_{F_{11}} & & \text{\Large 0} \\ & \bullet \; \bullet \; \bullet & \\ \text{\Large 0} & & \underbrace{\begin{array}{cccc} * \; * & & & 0 \\ * \; * & & & \\ & 0 & & \\ & & \ddots & \\ 0 & & & 0 \end{array}}_{F_{MM}} \end{array} \right) \tag{A.7}$$

and is therefore *singular*.

A.2 Solution of generalized linear eigenvalue problems

A.2.1 The Inverse Iteration

A standard method to find the dominant eigenvalue and the corresponding eigenvector for such a generalized linear eigenvalue problem is the *inverse iteration* as presented in algorithm A.1. It is an iterative method, where in each iteration a linear equation system (LES)

$$(A - \tilde{\rho}_i F) \, \underline{u}_{i+1} = F \underline{u}_i \tag{A.8}$$

has to be solved. Since the matrices of the special problem that was developed in the previous section are large and sparse, iterative methods (denoted by $\approx$ in the algorithm) have to be applied to solve these LESs. In the i^{th} iteration the approximations for the eigenvalue and the eigenvector are given by $(\rho_i, \underline{u}_i)$, $\tilde{\rho}_i$ is a *shift* that has to be chosen appropriately in each iteration.

Given: $\underline{u}_0$, $\tilde{\rho}_0$
$\underline{\text{for}}$ $i = 0, 1, ...$ $\underline{\text{until}}$ convergence $\underline{\text{do}}$

$$\underline{u}_{i+1} \approx (A - \tilde{\rho}_i F)^{-1} F \underline{u}_i;$$
$$\underline{u}_{i+1} = \frac{\underline{u}_{i+1}}{\|\underline{u}_{i+1}\|};$$
$$\rho_{i+1} := \frac{\langle \underline{u}_{i+1}, A\underline{u}_{i+1} \rangle}{\langle \underline{u}_{i+1}, F\underline{u}_{i+1} \rangle};$$

Algorithm A.1 : Inverse Iteration

To get a good convergence of the inverse iteration, the shift $\tilde{\rho}_i$ should be chosen close to the desired eigenvalue, but this choice results in a high condition number $\kappa(A - \tilde{\rho}_i F)$. The effect of a high condition number on the iterative solution of the LES (A.8) is outlined briefly in the introduction to chapter 4. For the convergence of the *outer* (inverse) *iteration* an increasing number of *inner iterations*, i.e. iterations for the solution of (A.8), are needed. Therefore a careful choice of the shift and of the accuracy of the inner iteration is important.

A.2.2 The Inverse Correction

A modification of the inverse iteration that avoids this problem has been proposed by Schmid [33]. The method is called *inverse correction*, see algorithm A.2.

Given: $\underline{u}_0$, $\tilde{\rho}_0$, ρ_0
$\underline{\text{for}}$ $i = 0, 1, ...$ $\underline{\text{until}}$ convergence $\underline{\text{do}}$

$$\underline{d}_i := (A - \rho_i F)\, \underline{u}_i;$$
$$\underline{v}_i \approx (A - \tilde{\rho}_i F)^{-1} \underline{d}_i;$$
$$\underline{u}_{i+1} := \underline{u}_i - \underline{v}_i;$$
$$\rho_{i+1} := \frac{\langle \underline{u}_{i+1}, A\underline{u}_{i+1} \rangle}{\langle \underline{u}_{i+1}, F\underline{u}_{i+1} \rangle};$$

Algorithm A.2 : Inverse Correction

In the inverse correction only a correction $\underline{v}_i$ for the previous approximation $\underline{u}_i$ is calculated. The advantage of this method is that only few valid digits for the correction $\underline{v}_i$ are sufficient to improve the approximation $\underline{u}_i$. The choice of the shift $\tilde{\rho}_i$ is not so critical either, e.g. $\tilde{\rho}_i := \sigma_i \, \rho_i$ with $\sigma_i := 0.95$ or 0.995.

To find out whether the approximation $\underline{u}_i$ obtained in the ith iteration of algorithm A.2 (or algorithm A.1 respectively) is accurate enough or not, the convergence tests described in section 4.4 are used again.

A comparison of the two methods together with an analysis and numerical tests can be found in Schmid [33, ch. 7]. The results of the analysis and the numerical tests in Schmid [33] show that in the case of large, sparse generalized linear eigenvalue problems the method of the inverse correction (algorithm A.2) is better than the inverse iteration (algorithm A.1). Therefore only the inverse correction was used to solve the stationary problem.

The main step in algorithm A.2 is the solution of a LES

$$(A - \rho_i F) \, \underline{u}_i = \underline{v}_i \tag{A.9}$$

in each iteration i. Since these LESs have the same structure as the LESs that arise in the transient case, the same iterative methods can be applied. The methods that were used here are described in chapter 4 and 5. The numerical results are presented in the next section.

A.3 Numerical results

The generalized linear eigenvalue problem, arising in the discretization process of the quasi-stationary multigroup diffusion equations, was solved with the method of the inverse correction (algorithm A.2). An extensive presentation of numerical results obtained with this method for realistic 3D reactor simulations and a comparison to the results obtained with the inverse iteration can be found in Schmid [33, ch. 7]. For the solution of the LESs in [33], Multi-Grid and Block-ILU preconditioned GMRES are used.

The solutions obtained with the inverse correction for the examples 1, 2 and 3, defined in section 6.1, differ only slightly to the solutions obtained with PANBOX2 of the Siemens AG Operating Division KWU [3]. The multiplication constants (or eigenvalues) obtained for example 3 on grid 1 and 2 are compared to the results of PANBOX on grid 1 in table A.1.

	Grid 1	Grid 2	PANBOX(NEM-M0)	PANBOX(NEM-M2)
λ_0	1.039857	1.039462	1.039858	1.039849

Table A.1: Multiplication constant λ_0 for example 3

The difference between the two multiplication constants for the same spatial discretization NEM-M0 on grid 1 is only 10^{-6}. Because of the inaccuracy of the data on grid 2, which is interpolated from the homogenized data on grid 1, the eigenvalue obtained on grid 2 is not as accurate (difference: $3.9 \cdot 10^{-4}$).
As the solution of the eigenvalue problem is not unique, the eigenvectors can only be compared by using the angle between the vectors. The difference between the two vectors is therefore calculated from

$$\delta = \frac{\langle \underline{x}, \underline{y} \rangle}{\|x\|_2 \ \|y\|_2} - 1$$

For the same spatial discretization NEM-M0 on grid 1, the difference δ between the eigenvector obtained from PANBOX and from Bi-CGStab was in fact $\delta = 3.5 \cdot 10^{-9}$.

Figure A.2 shows the calculated flux density distribution of the “fast” neutrons $\phi_1(\vec{x})$ in the reactor. The distribution is between $3 \cdot 10^{10}$ $[\frac{\text{neutrons}}{\text{cm}^2 \cdot \text{s}}]$ at the boundaries **(pink)** and $9 \cdot 10^{14}$ $[\frac{\text{neutrons}}{\text{cm}^2 \cdot \text{s}}]$ in the interior **(red)**.

It was mentioned in the previous section that the main step in the inverse correction is the solution of a LES (A.9) in each iteration i. The LESs are almost singular, but it is not necessary to solve them very exactly. A tolerance of $\varepsilon_R = 10^{-2}$ for the defect test (4.31) is enough to get a good correction. The shift $\tilde{\rho}_i$ is chosen as suggested in section A.2.2, namely $\tilde{\rho}_i := \sigma_i \, \rho_i$ with $\sigma_i := 0.95$ for $i = 1, 2$ and $\sigma_i := 0.995$ for $i \geq 3$.

The performance of the different solution methods from chapters 4 and 5, applied to these LESs within the inverse correction method, is given in table A.2. The components and parameter for Multi-Grid (MGM, algorithm 4.4) and preconditioned Bi-CGStab are chosen as in chapter 6 (at the beginning of section 6.2). There is no good initial guess for the solution of the LES (A.9) in each iteration i available, therefore it makes sense to also use Full-MGM (algorithm 4.5) on grid 2. The components are chosen as for MGM. The number of nested MG-iterations on grid 1 is 2.

The convergence test was chosen to be the defect test (4.31) with $\varepsilon_R = 10^{-8}$. The number of “outer” iterations i of the inverse correction algorithm A.2 that were

	Grid 2			Grid 1
Processors	Full-MGM	MG (3/2)	Bi-CGStab	Bi-CGStab
1	175.4 s (24)	461.3 s (72)	846.8 s (174)	48.0 s (104)
2	87.2 s (24)	231.2 s (73)	398.5 s (177)	21.3 s (102)
3	62.7 s (25)	156.8 s (73)	294.7 s (176)	14.6 s (98)
4	43.2 s (25)	109.4 s (73)	182.1 s (172)	10.8 s (97)
5	36.6 s (27)	83.6 s (73)	159.5 s (185)	9.3 s (98)
6	28.4 s (25)	70.1 s (73)	127.3 s (178)	8.2 s (96)
7	25.8 s (27)	60.4 s (74)	111.3 s (179)	6.7 s (95)
8	22.7 s (27)	52.9 s (73)	93.5 s (174)	6.3 s (98)
9	20.0 s (26)	48.3 s (74)	85.7 s (177)	5.7 s (98)
10	19.1 s (27)	43.4 s (74)	79.1 s (181)	5.1 s (95)
11	17.2 s (24)	42.1 s (73)	76.1 s (174)	4.7 s (96)
12	14.7 s (23)	36.8 s (72)	65.2 s (170)	4.3 s (91)

Table A.2: Execution time and "inner" iterations for the quasi-stationary calculations of example 3 ($\varepsilon_R = 10^{-8}$)

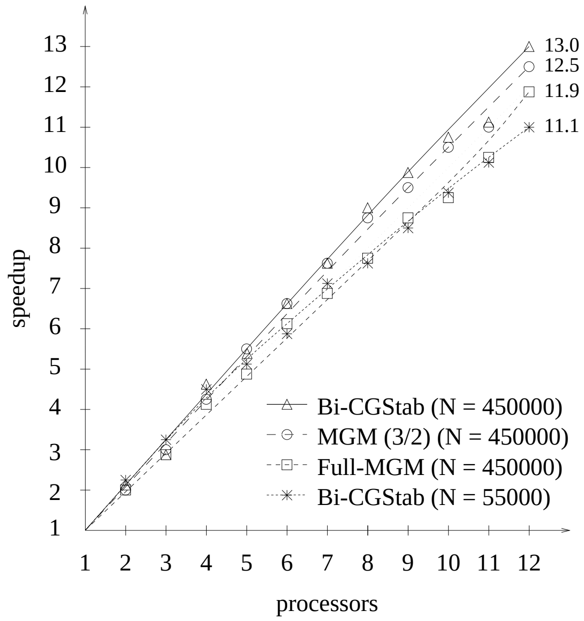


Figure A.1: Speedup for the quasi-stationary calculations of example 3 needed, was 9 for all different methods. The number of "inner" iterations, i.e. the sum of all iterations needed for the solution of the 9 LESs, is given in table A.2 together with the execution times.

The results for preconditioned Bi-CGStab and MGM are similar to the results in the transient case (compare chapter 6). MGM is almost two times faster than preconditioned Bi-CGStab once again. The introduction of the nested iteration in Full-MGM even increases the efficiency of MGM by a factor 2.5. Full-MGM is therefore almost 5 times faster than preconditioned Bi-CGStab.

The speedup is finally presented in figure A.1. On grid 2 it is optimal for all methods once again. The speedup for BiCGStab on grid 1 starts to loose its optimality for more than 10 processors as in the transient case.

The results confirm the proposition that parallel Full-MGM is a perfect match for the solution of the LESs arising in the inverse correction method.

Bibliography

[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 Trans. Computer Aided Des. **4**(4), 436–451 (1985).

[2] B. E. Bauer, *Practical Parallel Programming.* Academic Press, Inc., San Diego (1992).

[3] R. Böer, R. Böhm, H. Finnemann and R. Müller, *The coupled neutronics nd thermal hydraulics code system PANBOX for PWR safety analysis.* Atomkernenergie/Kerntechnik **57**(1), 49–54 (1992).

[4] J. J. Duderstadt and L. J. Hamilton, *Nuclear Reactor Analysis.* J. Wiley & Sons, New York (1976).

[5] D. Emmendorfer and K. H. Höcker, *Theorie der Kernreaktoren, Band 1: Der stationäre Reaktor, 2. Aufl.* Bibliographisches Institut, Mannheim (1982).

[6] D. Emmendorfer and K. H. Höcker, *Theorie der Kernreaktoren, Band 2: Der instationäre Reaktor.* Bibliographisches Institut, Mannheim (1993).

[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. HPCN '95 Milan, May 3–5 1995, Springer Verlag, Heidelberg, 318–328 (1995).

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

[9] H. Finnemann and W. Grundlach, *Space-time kinetics code IQSBOX for PWR and BWR. Part I: Description of physical and thermo-hydraulic models.* Atomkernenergie/Kerntechnik **37**(3), 176–182 (1981).

[10] H. Finnemann and J. Volkert, *Parallel Mulitgrid Algorithms Implemented on Memory-Coupled Multiprocessors.* Nuclear Science and Engineering: **100**, 226–236 (1988).

[11] H. Finnemann, J. Brehm, E. Michel and J. Volkert, *Mulitgrid Solution of diffusion equations on distributed memory multiprocessor systems.* Atomkernenergie/Kerntechnik **52**(3), 169–174 (1988).

[12] H. Finnemann and A. Galati, *NEACRP 3–D LWR Core Transient Benchmark,* Final Specifications. NEACRP-L-335, Revision 1 (1992).

[13] M. J. Flynn, *Very High Speed Computing Systems.* Proceedings IEEE, **54**, 1901–1909 (1966).

[14] G. H. Golup and C. F. Van Loan, *Matrix Computations (Second Edition).* The Johns Hopkins University Press, Baltimore, London (1989).

[15] T. Grauschopf, M. Griebel and H. Regler, *Additive Multilevel-Preconditioners based on Bilinear Interpolation, Matrix Dependent Geometric Coarsening and Algebraic Multi-Grid for Second Order Elliptic PDEs.*
SFB-Bericht Nr.342/02/96 A (1996).

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

[17] W. Hackbusch, *Iterative Lösung großer schwachbesetzter Gleichungssysteme.* Teubner, Stuttgart (1991).

[18] E. Hairer and G. Wanner, *Solving Ordinary Differential Equations II: Stiff and Differntial-Algebraic Problems.* Springer Verlag, Berlin, Heidelberg (1991).

[19] 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, September 16–20 1996.

[20] Han Gyu 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, September 16–20 1996, J42–J51 (1996).

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

[22] M. Jung and U. Langer, *Skriptum zur Vorlesung FEM (Eine Einführung für Ingenieursstudenten).* TU Chemnitz (Faculty of Mathematics) and Johannes Kepler University Linz (Institute for Mathematics), Chemnitz and Linz (1993).

[23] F. Kickinger, *Algebraic Multigrid for Discrete Elliptic Second-Order Problems.* Institutsbericht Nr. 513, University of Linz, Institute for Mathematics (1995).

[24] Yong Hee Kim and Nam Zin Cho, *Parallel Solution of the Neutron Diffusion Equation with the Domain Decomposition Method on a Transputer Network.* Nuclear Science and Engineering: **114**, 252–270 (1993).

[25] A. Knoll and R. Müller, *Coupling RELAP5 and PANBOX2: A Three-Dimensional Space-Time Kinetics Application with RELAP5.* International Conference on New Trends in Nuclear System Thermalhydraulics, Pisa, May 30 – June 2 1994.

[26] U. Langer, *Skriptum zur Vorlesung NUMERIK II (RWA).* Johannes Kepler University Linz, Institute for Mathematics, Linz (1996).

[27] U. Langer, *Skriptum zur Vorlesung NUMERIK III (AWA, ARWA).* Johannes Kepler University Linz, Institute for Mathematics, Linz (1997).

[28] U. Langer, *Skriptum zur Vorlesung MULTIGRID - METHODEN.* Johannes Kepler University Linz, Institute for Mathematics, Linz (1996).

[29] A. Reusken, *A Multigrid Method Based on Incomplete Gaussian Elimination.* Eindhoven University of Technology, Department for Mathematicas and Computer Science, RANA 95-13 (1995).

[30] J. Ruge and K. Stüben, *Algebraic Multigrid (AMG).* Multigrid Methods (St. McCormick, ed.), Frontiers in Applied Mathematics, Vol. 5, SIAM, Philadelphis (1986).

[31] Y. Saad, *Iterative Methods for Sparse Linear Systems.* PWS Publishing Company, Boston (1996).

[32] H. N. Sarsour, P. J. Turinsky and C. S. Henkel, *Parallel Solution of the Few-Group Neutron Diffusion Equations.* Proceedings of the 5$^{\text{th}}$ SIAM Conference on parallel processing for scientific computing, Houston, March 25–27 1991, SIAM, Philadelphia (1992).

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

[34] M. Uematsu, M. Tsuiki and K. Kubota, *Parallel Processing of Boiling Reactor Core Analysis.* Journal of Nuclear Science and Technology, **32[6]**, 495–506 (June 1995).

[35] H. A. Van der Vorst, *Bi-CGSTAB: A more smoothly converging variant of Bi-CG for the solution of nonsymmetric linear systems.* SIAM J.Sci.Stat.Comput. **13**, 631–644 (1992).

[36] J.-L. Vaudescal, *Résolution Numérique des Équations de la Diffusion Neutronique Multigroupe.* Doctor Thesis, University of Paris IX, Paris (1993).

[37] E. L. Wachspress, *Iterative solution of elliptic systems and applications to the neutron diffusion equations of reactor physics.* Prentice-Hall, Inc., Englewood cliffs, N. J. (1966).

[38] F. Wagner, *Zeitdiskretisierung der Mehrgruppen-Neutronen-Diffusionsgleichungen für die objektorientierte, numerische Simulation.* AMNS, Bd. 12, Wißner, Augsburg (1996).

[39] P. Wesseling, *An introduction to multigrid methods.* John Wiley & Sons, Chichester, New York, Brisbane, Toronto, Singapore (1992).