# Beiträge zur Beschleunigung numerischer Simulationen von elektroquasistatischen Feldverteilungen

Christian Friedrich Richter

geboren am 27.06.1986 in Geldern

Von der Fakultät für Elektrotechnik, Informationstechnik
und Medientechnik der Bergischen Universität Wuppertal
genehmigte

Dissertation

zur Erlangung des akademischen Grades eines
Doktors der Ingenieurswissenschaften (Dr.-Ing.)

| | |
|---|---|
| Erstgutachter: | Prof. Dr. rer. nat. Markus Clemens |
| Zweitgutachter: | Prof. Dr. rer. nat. Sebastian Schöps |
| Vorlage der Arbeit: | 23. Oktober 2018 |
| Mündlichen Prüfung: | 23. Januar 2019 |

Die Dissertation kann wie folgt zitiert werden:

urn:nbn:de:hbz:468-20190329-142324-6
[http://nbn-resolving.de/urn/resolver.pl?urn=urn%3Anbn%3Ade%3Ahbz%3A468-20190329-142324-6]

# Inhaltsverzeichnis

**Inhaltsverzeichnis** **iii**

**Akronyme** **vii**

**Symbolverzeichnis** **xi**

**1 Einleitung** **1**

**2 Numerische Berechnung elektroquasistatischer Felder** **7**
- 2.1 Elektroquasistatische Felder . . . 7
- 2.2 Mikrovaristoren als Material mit nichtlinearer Leitfähigkeit . 11
- 2.3 Die Finite Elemente Methode in der Elektroquasistatik . . . 13
- 2.4 Zeitintegratoren . . . 18
  - 2.4.1 Implizite Zeitintegration . . . 23
  - 2.4.2 Explizite Zeitintegration . . . 25
- 2.5 Zusammenfassung . . . 29

**3 Beschleunigung numerischer Simulationen durch Grafikprozessoren** **31**
- 3.1 Aufbau eines Grafikprozessors . . . 33
- 3.2 Durchführung allgemeiner Berechnungen mittels einer GPU . 36
- 3.3 Zusammenfassung . . . 38

**4 Iterative Lösung großer dünnbesetzter linearer Gleichungssysteme** **39**
- 4.1 Die Methode der konjugierten Gradienten . . . 39
- 4.2 AMG als Vorkonditionierer . . . 42
  - 4.2.1 Smoothed-Aggregation Algebraic Multigrid . . . 45
  - 4.2.2 Bibliotheken . . . 50
- 4.3 Zusammenfassung . . . 51

**5 Einsatz von Grafikkarten-Beschleunigung in der Feldsimulation 53**
5.1 Beschleunigung des linearen Gleichungssystemlösers . . . . . 53
5.1.1 Matrix-Speicherformate . . . . . . . . . . . . . . . . . 55
5.1.2 Vorkonditionierer auf GPU . . . . . . . . . . . . . . . . 59
5.1.3 Nutzung mehrerer Grafikkarten . . . . . . . . . . . . 63
5.2 Massiv-parallele Assemblierung . . . . . . . . . . . . . . . . 74
5.2.1 Assemblierung auf GPUs . . . . . . . . . . . . . . . . 76
5.2.2 Kombination von Assemblierung und SpMV . . . . . 77
5.3 Zusammenfassung . . . . . . . . . . . . . . . . . . . . . . . . 79

**6 Beschleunigung der EQS-Simulation durch Adaption der Zeitintegrationsverfahren 81**
6.1 Beschleunigung des impliziten Zeitintegrationsverfahrens . . 82
6.2 Beschleunigung des expliziten Zeitintegrationsverfahrens . . 85
6.2.1 Ansätze zur Stufenwahl . . . . . . . . . . . . . . . . . 89
6.2.2 Startwertschätzung mittels „Subspace Projection Extrapolation“ -Methode . . . . . . . . . . . . . . . . . 93
6.2.3 Startwertgenerierung mittels Verfahren zur Modellordnungsreduktion . . . . . . . . . . . . . . . . . . . . . . 95
6.3 Zusammenfassung . . . . . . . . . . . . . . . . . . . . . . . . 98

**7 Validierung anhand von Anwendungsbeispielen 99**
7.1 Verwendete Rechensysteme . . . . . . . . . . . . . . . . . . . 99
7.2 Definition der Modelle . . . . . . . . . . . . . . . . . . . . . 100
7.2.1 Konventioneller Höchstspannungsisolator . . . . . . . 101
7.2.2 Hochspannungsdurchführung . . . . . . . . . . . . . . 104
7.2.3 Höchstspannungsisolatoren mit Mikrovaristoren . . . 106
7.2.4 Höchstspannungsisolatoren mit Wassertropfen . . . . 109
7.2.5 Höchstspannungsisolator in erhöhter Auflösung . . . 112
7.3 Vergleich des Multi-GPU-PCG-Lösers für LGS . . . . . . . . 113
7.3.1 Konvergenzverhalten . . . . . . . . . . . . . . . . . . . 113
7.3.2 Speicherbedarf . . . . . . . . . . . . . . . . . . . . . . 117
7.3.3 Rechengeschwindigkeit . . . . . . . . . . . . . . . . . . 121
7.3.4 Mixed-Precision-Ansatz . . . . . . . . . . . . . . . . . 126
7.4 Validierung der GPU-basierten Assemblierung . . . . . . . . 128
7.5 Untersuchung der Zeitintegratoren anhand der Anwendungsbeispiele . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 132
7.5.1 Beschleunigte Simulationen für Beispielmodelle mit Ansatzfunktionen erster Ordnung . . . . . . . . . . . 135

7.5.2 Beschleunigte Simulationen für Beispielmodelle mit Ansatzfunktionen zweiter Ordnung . . . . . . . . . . . 144
7.5.3 Zeitintegrationsfehler . . . . . . . . . . . . . . . . . . . . 152
7.6 Zusammenfassung . . . . . . . . . . . . . . . . . . . . . . . . 162

**8 Zusammenfassung 163**

**9 Anhang 169**
9.1 Pseudocodes der verwendeten Funktionen . . . . . . . . . . . 169

**Literaturverzeichnis 175**

**Eigene Veröffentlichungen 189**

# Akronyme

| | |
|---|---|
| AInv | Approximate Inverse |
| ALU | Algorithmical Logical Unit |
| AMG | algebraisches Mehrgitterverfahren |
| BDF | Backward-Differentiation Formulas |
| BLAS | Basic Linear Algebra Subprograms |
| CAD | Computer-Aided Design |
| CFL | Courant-Friedrich-Lewy |
| CG | Conjugate Gradients |
| COO | Coordinate list |
| CPU | Central Processing Unit |
| CSR | Compressed Sparse Row |
| CUDA | Compute Unified Device Architecture |
| DIA | Diagonal |
| DIRK | Diagonal Implicit Runge-Kutta |
| DP | Double Precision |
| EQS | Elektroquasistatik |
| FD | Finite Differenzen |
| FEM | Finite Elemente Methode |

| | |
|---|---|
| FIT | Finite Integrationstechnik |
| GFK | Glasfaserverstärkter Kunststoff |
| GMG | geometrisches Mehrgitterverfahren |
| GPGPU | General Purpose Graphics Porcessing Unit |
| GPU | Graphics Processing Unit |
| HYB | Hybrid |
| hypre | High Performance Preconditioners |
| IC | Incomplete Choleski |
| IEEE | Institute of Electrical and Electronics Engineers |
| ILP | Instruction Level Parallism |
| IRK | Implicit Runge-Kutta |
| LGS | lineares Gleichungssystem |
| MEQSICO | Magneto-/Electro-Quasistatic Simulation Code |
| MGS | Modified Gram-Schmidt |
| MOR | Model Order Reduction |
| MPI | Message Passing Interface |
| MRHS | Multiple Right-Hand-Side |
| PCG | Preconditioned Conjugate Gradients |
| PCIe | Peripheral Component Interconnect Express |
| PETSc | Portable Extensible Toolkit for Scientific Computing |
| POD | Proper Orthogonal Decomposition |
| RHS | Right Hand Side |
| RKC | Runge-Kutta-Chebychev |

| | |
|---|---|
| SA | Smoothed Aggregation |
| SDIRK | Singly Diagonal Implicit Runge-Kutta |
| SIMT | Single Instruction Multiple Threads |
| SM | Streaming Multiprocessor |
| SPE | Subspace Projection Extrapolation |
| SpMV | Sparse Matrix Vector Multiplication |
| SSOR | Symmetric Successive Over-Relaxing |
| SVD | Singular Value Decomposition |
| TBB | Thread Building Blocks |
| ZnO | Zinkoxid |

# Symbolverzeichnis

| | |
|---|---|
| $H$ | Gröbster AMG-Level |
| $K$ | Anzahl der CG-Iterationen zur Lösung des LGS |
| $M$ | Anzahl der Zeitschritte bis zum letzten Zeitpunkt |
| $R$ | Anzahl der Snapshots |
| $S$ | Speicherbedarf in Byte |
| $T$ | Letzter Zeitpunkt der Rechnung |
| $V$ | betrachteter Raum in $\Omega$ |
| $W$ | Anzahl der verwendeten linkssinglären Vektoren |
| $\Delta t_{CFL}$ | Maximaler stabiler Zeitschritt nach CFL |
| $\Delta t$ | diskreter Zeitschritt |
| $\Gamma$ | Rand des Gesamtraums $\Omega$ |
| $\Omega$ | gesamter Rechenraum |
| $\epsilon$ | Toleranzwert |
| $\kappa$ | spezifische Leitfähigkeit in $\frac{S}{m} = \frac{A}{V\,m}$ |
| $\lambda$ | Eigenwert |
| $\mathbf{Agg}$ | Aggregatmatrix |
| $\mathbf{A}$ | Systemmatrix des linearen Gleichungssystems |
| $\mathbf{B}$ | Vorkonditionierer |
| $\mathbf{C}$ | Systemmatrix ohne schwache Verbindungen |

| | |
|---|---|
| $\mathbf{D}$ | Hauptdiagonalmatrix |
| $\mathbf{I}$ | Einheitsmatrix |
| $\mathbf{J}$ | Jacobimatrix |
| $\mathbf{K}_{\kappa}$ | diskretisierte Leitwertmatrix in $\frac{A}{V\,m^3}$ |
| $\mathbf{K}_{\kappa,D}$ | Leitwertmatrix für Dirichletwerte in $\frac{A}{V\,m^3}$ |
| $\mathbf{K}_{\kappa,tot}$ | Leitwertmatrix für alle FEM-Knoten in $\frac{A}{V\,m^3}$ |
| $\mathbf{M}_{\varepsilon}$ | diskretisierte Permittivitätsmatrix in $\frac{A\,s}{V\,m^3}$ |
| $\mathbf{M}_{\varepsilon,D}$ | Permittivitätsmatrix für Dirichletwerte in $\frac{A\,s}{V\,m^3}$ |
| $\mathbf{M}_{\varepsilon,tot}$ | Permittivitätsmatrix für alle FEM-Knoten in $\frac{A\,s}{V\,m^3}$ |
| $\mathbf{P}$ | Prolongationsmatrix |
| $\mathbf{Q}_e$ | FEM-Indexmatrix |
| $\mathbf{Q}$ | Orthonormalvektorenmatrix |
| $\mathbf{R}$ | Restriktionsmatrix |
| $\mathbf{S}$ | Glättungsmatrix |
| $\mathbf{T}$ | ungeglättete Interpolationsmatrix |
| $\mathbf{U}$ | Matrix der rechtssingulären Vektoren |
| $\mathbf{V}$ | Matrix der linkssingulären Vektoren |
| $\mathbf{X}$ | Snapshotmatrix |
| $\mathbf{\Sigma}$ | Singulärwertmatrix |
| $\mu_0$ | Permeabilität im Vakuum in $\frac{V\,s}{A\,m}$ |
| $\rho$ | Ladungsdichte in $\frac{A\,s}{m^3}$ |
| $\sigma$ | Spektralradius |
| $\tau$ | Zeitkonstante in $s$ |
| $\theta$ | Skalierungsfaktor, $0 \leq \theta \leq 1$ |

| | |
|---|---|
| $\varepsilon_0$ | Permittivität im Vakuum in $\frac{A\,s}{V\,m}$ |
| $\varepsilon_r$ | relative Permittivität |
| $\varepsilon$ | Permittivität in $\frac{A\,s}{V\,m}$ |
| $\varkappa$ | Konditionszahl |
| $\varphi$ | Elektrisches Potential in $V$ |
| $\mathbf{b}$ | Rechte-Seite-Vektor des linearen Gleichungssystems |
| $\mathbf{e}$ | Fehlervektor |
| $\mathbf{f}$ | Diskreter Vektor der Ableitung $\frac{d\mathbf{u}}{dt}$ |
| $\mathbf{f}$ | Diskreter Vektor der Ableitung $\frac{d\mathbf{u}(t)}{dt}$ in $\frac{V}{s}$ |
| $\mathbf{k}$ | Kandidatenvektor |
| $\mathbf{q}$ | Orthonormalvektor |
| $\mathbf{r}$ | Residuumsvektor |
| $\mathbf{u}_0$ | Diskreter Anfangswertvektor in $V$ |
| $\mathbf{u}_D$ | Dirichletwertvektor in $V$ |
| $\mathbf{u}$ | Diskreter Potentialvektor in $V$ |
| $\vec{A}$ | Fläche in $\Omega$ |
| $\vec{B}$ | Magnetische Flussdichte in $T = \frac{V\,s}{m^2}$ |
| $\vec{D}$ | Elektrische Flussdichte in $\frac{A\,s}{m^2}$ |
| $\vec{E}$ | Elektrische Feldstärke in $\frac{V}{m}$ |
| $\vec{H}$ | Magnetische Feldstärke in $\frac{A}{m}$ |
| $\vec{J}$ | Elektrische Stromdichtein $\frac{A}{m^2}$ |
| $\vec{M}$ | Magnetisierung in $\frac{V\,s}{A\,m}$ |
| $\vec{P}$ | Polarisation in $\frac{A\,s}{V\,m}$ |
| $\vec{s}$ | Strecke in $\Omega$ |

| | |
|---|---|
| $\vec{x}$ | Punkt in $\Omega$ |
| $c$ | Lichtgeschwindigkeit in in $\frac{m}{s}$ |
| $e$ | Lokaler Integrationsfehler |
| $f$ | Frequenz in $s^{-1}$ |
| $h_{FEM}$ | Minimale Kantengröße des FE-Gitters |
| $h$ | Laufvariable des AMG-Levels |
| $i$ | für Matrizen: Zeilenindex |
| $i$ | bei Zeitintegration: Laufvariable des Zeitschritts |
| $j$ | für Matrizen: Spaltenindex |
| $j$ | bei Zeitintegration: Laufvariable der Stufe |
| $k$ | Laufvariable der CG-Iteration |
| $l$ | Laufvariable der AMG-Iteration |
| $n_D$ | Anzahl der Dirichletknoten |
| $n_E$ | Anzahl der FEM-Elemente |
| $n_{GPU}$ | Anzahl der verwendeten GPUs |
| $n_{tot}$ | Gesamtanzahl aller Knoten |
| $nnz$ | Nicht-Null Einträge der Systemmatrix |
| $n$ | Anzahl der Unbekannten |
| $r$ | Laufvariable der Modellordungsreduktion |
| $s_{GPU}$ | Anzahl der GPU-Streams |
| $s$ | Stufenanzahl bei der Zeitintegration |
| $t_{ref}$ | Kopierdauer eines Vektor zwischen zwei GPUs in $s$ |
| $v$ | Testfunktion |

# Kapitel 1

# Einleitung

Eine typische Entwicklungsaufgabe im Bereich der Hochspannungstechnik ist das Design von Isolatoren. Hier ist das Erreichen eines Entwurfes, welcher auf möglichst kleinem Bauraum Entladungsfreiheit garantiert, wesentliches Kriterium. Neben Langstabisolatoren sind auch Durchführungen, Muffen oder Kabelendverschlüsse als mögliche Applikationen zu nennen. Diese Betriebsmittel der elektrischen Energieübertragungstechnik zielen darauf, das elektrische Feld so zu formen, dass eine möglichst homogene Feldverteilung erreicht wird und keine Feldüberhöhungen entstehen, welche durch Entladungen den Isoliergegenstand beschädigen können und letztendlich zu einem Durchschlag führen. Zur Steuerung des elektrischen Feldes werden anwendungsabhängig unterschiedliche Ansätze und Materialien zur Feldsteuerung eingesetzt. Klassische Ansätze sind die kapazitive, refraktive oder auch geometrische Feldsteuerung [1]. Ein anderer Ansatz ist die feldstärkenabhängige Steuerung durch Feldsteuerelemente mit nichtlinearer Leitfähigkeit. Diese sind in der Lage, ihre Leitfähigkeit lokal feldstärkeabhängig zu ändern. Verglichen mit traditionellen Ansätzen stellt diese Vorgehensweise eine vielversprechende Alternative dar, da durch die dynamische Steuerung kleinere Bauformen realisiert werden können. Ein Beispiel für solche Feldsteuermaterialien sind sogenannte Mikrovaristoren. Hierbei handelt es sich um ein Produkt, welches auf Zinkoxidkeramikvaristoren basiert, wie sie in Hochspannungsableitern eingesetzt werden [2]. Das zerstoßene Mikrogranulat kann in eine Silikonmatrix oder einen Lack eingebracht werden und so leicht an verschiedene Formen angepasst werden.

Durch die numerische Berechnung der Feldverteilung kann das gesamte Feld visualisiert und Optimierungen vorgenommen werden. Die Optimierung von beispielsweise Langstabisolatoren, welche diese Materialien verwenden, stellt jedoch eine neue Problemklasse dar. Nun müssen nicht nur kapazitive, sondern auch resistive Effekte berücksichtigt werden. Dies macht die numerische Simulation einer elektroquasistatischen Feldmodellierung, anstelle einer elektrostatischen Feldmodellierung erforderlich. Das elektroquasistatische Problem wird im Raum und über der Zeit gelöst, was die Simulationszeit erheblich steigert.

Zusätzlich muss die feldstärkeabhängige Leitfähigkeit berücksichtigt werden, welche in jedem Zeitschritt ortsabhängig neu ermittelt werden muss. Bei impliziten Zeitintegrationsverfahren ist zusätzlich eine sukzessive Linearisierung notwendig, da die Leitfähigkeit von der zu errechnenden Feldverteilung abhängt. Dies führt, insbesondere für größere Modelle mit mehreren Millionen Unbekannten zu erheblichen Rechenzeiten, welche im Einzelfall mehrere Wochen pro Modell betragen können [3][B1]. Unter diesen Voraussetzungen ist eine Optimierung, welche mit der Berechnung einer höheren Anzahl an Modellvariationen einhergeht, sehr zeitaufwendig.

Das wesentliche Hindernis ist daher die Rechendauer, die zur Lösung einer Modellkonfiguration benötigt wird. Diese Arbeit zielt auf einen Geschwindigkeitsgewinn in der Simulation bei gleichbleibender Genauigkeit, der durch die Benutzung neuartiger Verfahren und deren Optimierung auf parallele Rechnerarchitekturen gewonnen werden kann. Zum einen werden dazu technisch bedingte Geschwindigkeitsgewinne genutzt, wie sie nach dem Moore'schen Gesetz zu erwarten sind [5]. Zum anderen wird durch methodische Entwicklung ein weiterer Geschwindigkeitsgewinn erreicht, wie er etwa für die Geschichte der linearen Gleichungssystemlöser in Abb. 1.1 dargestellt ist. Als Referenz dient das Modell eines Langstabisolators, welches zu Beginn der Bearbeitungsphase in 2012 mit Ansatzfunktionen erster Ordnung 105 Stunden zur Lösung benötigte. Mit Ansatzfunktionen zweiter Ordnung war das Problem in akzeptabler Zeit nicht lösbar.

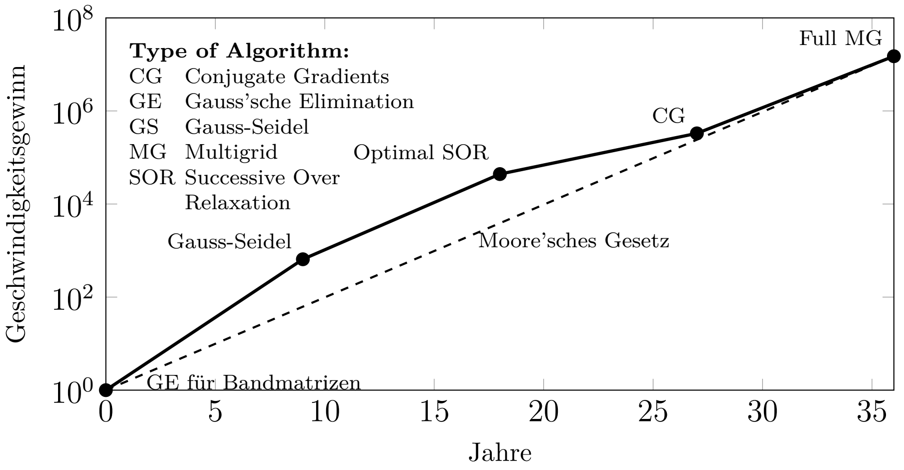


Abbildung 1.1: Moore'sches Gesetz für Löser von linearen Gleichungssystemen (eigene Darstellung nach [4]).

Die leitenden Forschungsfragen dieser Arbeit sind:

- Wie kann der technische Fortschritt im Bereich hochparalleler Rechensysteme für die Simulation von Problemen der Elektroquasistatik (EQS) genutzt werden?
- Welche Veränderung für Algorithmen ergeben sich aus der Verwendung hochparalleler Rechensysteme und wie beeinflussen diese die Verfahren und Ansätze zur Lösung von elektroquasistatischen Problemen? Sind durch diese Änderungen andere Ansätze vielversprechend?
- Kann mit den erreichten Fortschritten die Simulationszeit unter eine Stunde gesenkt werden, damit eine ausreichende Anzahl an Modellvarianten mit mehr als einer Millionen Freiheitsgraden in akzeptablen Zeiten zur Verfügung stehen, um eine numerische Geometrieoptimierung durchführen zu können?

## Gliederung der Arbeit

Die vorliegende Arbeit schlägt verschiedene Möglichkeiten vor, die Simulation zu beschleunigen. Die Einzelprozesse werden im Hinblick auf ihren Beitrag zur Gesamtrechenzeit bewertet und optimiert. Der Fokus liegt auf der Beschleunigung mit Grafikprozessoren (engl.: Graphics Processing Unit (GPU)), welche über mehrere hundert bis tausend Prozessoren verfügen, weshalb eine gute Parallelisierbarkeit der Prozesse wichtig ist.

Nach der Einleitung werden in Kapitel 2 die elektrotechnischen Grundlagen skizziert, auf denen die mathematischen Modelle aufbauen. Die elektroquasistatische Potentialgleichung wird aus den Maxwell'schen Gleichungen hergeleitet, die anschließend im Raum mit der Finite Elemente Methode (FEM) und in der Zeit mit impliziten und expliziten Zeitintegratoren diskretisiert wird.

Technisch soll eine Beschleunigung der Simulationsrechenzeit durch massive Parallelisierung unter Nutzung von Grafikprozessoren erreicht werden. Daher wird in Kapitel 3 der Aufbau einer GPU, basierend auf dem Compute Unified Device Architecture (CUDA) Architekturmodell, hinsichtlich Programmiermodell, Speichermodell und Rechenmodell dargestellt.

In Kapitel 4 werden Methoden zur iterativen Lösung großer, dünnbesetzter linearer Gleichungssysteme eingeführt. Als Vorkonditionierer wird ein algebraisches Mehrgitterverfahren (AMG) basierend auf Smoothed Aggregation (SA) vorgeschlagen.

Aufbauend auf diesen Grundlagen werden in Kapitel 5 und 6 eigene Ansätze zur Beschleunigung entwickelt. In jedem Zeitschritt sind mehrere, aus der räumlichen Diskretisierung resultierende lineare Gleichungssysteme zu lösen. Dieser Lösungsvorgang stellt den größten Rechenaufwand dar. Daher wird als der lineare Gleichungssystemlöser, welcher auf der Methode der konjugierten Gradienten (engl.: Conjugate Gradients (CG)) basiert, mittels GPUs beschleunigt. Um Probleme, welche den Speicherplatz einer GPU übersteigen, rechnen zu können und einen Geschwindigkeitsgewinn zu erreichen, wird ein CG-Verfahren entwickelt, welches mehrere GPUs parallel nutzt. Hier werden Vektor- und Matrix-Klassen, Kommunikationsroutinen und ein

CG-Verfahren mit AMG-Vorkonditionierer für mehrere GPUs entwickelt und die Rechenzeit für den Lösungsvorgang reduziert.

Die Assemblierung der FEM-Matrix mit feldabhängigen Leitwerten ist in ihrer Struktur bereits ein paralleler Prozess. Hier wird eine Implementierung für die GPU vorgestellt, welche die technischen Neuerungen moderner GPUs ausnutzt.

Nach der Beschleunigung mit GPUs verschiebt sich der Zeitaufwand, welcher auf einzelne Komponenten eines Zeitschritts entfällt. Daher werden neben Verfahren der impliziten Zeitintegration auch Methoden der expliziten Zeitintegration für die Elektroquasistatik adaptiert, welche vorher zeitlich unrentabel waren, nun aber einen weiteren Geschwindigkeitsgewinn erreichen. Durch den geänderten Zeitintegrationsansatz können dann Möglichkeiten der Startwertgenerierung für den linearen Gleichungssystemlöser effektiv genutzt werden, welche die Rechenzeit weiter reduzieren.

In Kapitel 7 werden eine Reihe von Beispielanwendungen der Hochspannungstechnik simuliert um die vorgestellten Ansätze zu verifizieren.

Die Arbeit schließt mit einer Zusammenfassung (Kap. 8).

# Kapitel 2

# Numerische Berechnung elektroquasistatischer Felder

## 2.1 Elektroquasistatische Felder

Die Grundlage zur Beschreibung elektromagnetischer Felder stellen die sogenannten Maxwell'schen Gleichungen dar [6]. Diese bilden makroskopisch alle Zusammenhänge der Elektromagnetischen Feldtheorie ab. Allgemein werden sie in differentieller oder integraler Form angegeben. In integraler Form lauten sie [7]:

$$\iint\limits_{\partial V} \vec{D} \cdot d\vec{A} = \iiint\limits_{V} \rho \, dV \tag{2.1a}$$

$$\oint\limits_{\partial A} \vec{E} \cdot d\vec{s} = -\iint\limits_{A} \frac{\partial \vec{B}}{\partial t} \cdot d\vec{A} \tag{2.1b}$$

$$\iint\limits_{\partial \Omega} \vec{B} \cdot d\vec{A} = 0 \tag{2.1c}$$

$$\oint\limits_{\partial A} \vec{H} \cdot d\vec{s} = \iint\limits_{A} \left( \vec{J} + \frac{\partial \vec{D}}{\partial t} \right) \cdot d\vec{A} \tag{2.1d}$$

Hierbei sind, als elektrische Vektorgrößen, $\vec{D}$ die elektrische Flussdichte und $\vec{E}$ die elektrische Feldstärke. $\vec{J}$ ist die elektrische Stromdichte und $\rho$ – als einzige ungerichtete Größe – die Ladungsdichte. Das Magnetfeld wird

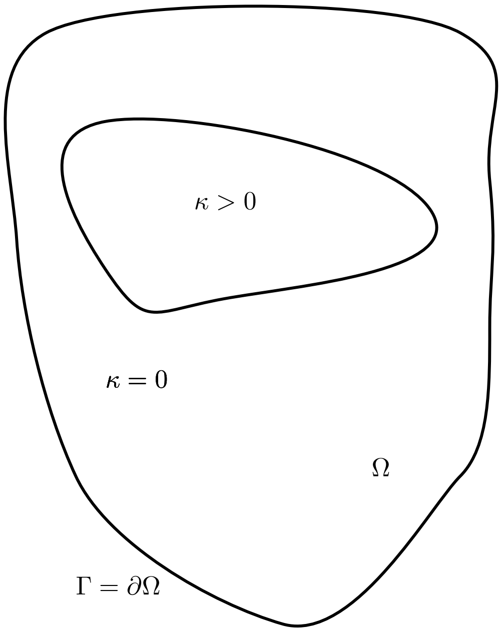


Abbildung 2.1: Beispielraum mit leitfähigem ($\kappa > 0$) und nicht leitfähigem ($\kappa = 0$) Gebiet.

durch die magnetische Flussdichte $\vec{B}$ und die magnetische Feldstärke $\vec{H}$ dargestellt. Dabei sind alle Funktionen orts- und zeitabhängig.

Der Ort ist ein Punkt $\vec{x} \in \mathbb{R}^3$ im betrachteten Raum $V$ im Gesamtraum $\Omega$. Der Rand des Rechenraumes $V$ wird durch die Fläche $\partial V$ gegeben. Der Rand des Gesamtraumes wird mit $\partial\Omega = \Gamma$ bezeichnet. Auf ihm sind zur vollständigen Definition des Problems zusätzlich Randbedingungen vorzugeben. $\vec{A}$ bezeichnet eine Fläche im Raum $\Omega$. Analog zum Raum bezeichnet $\partial\vec{A}$ den Rand der Fläche $\vec{A}$. Dies ist in Abb. 2.1 dargestellt.

(2.1a) wird das „Gauß'sche Gesetz“ genannt. Es setzt die elektrische Flussdichte in Beziehung zur elektrischen Ladung. Als Induktionsgesetz oder Faraday'sches Gesetz wird (2.1b) bezeichnet. Dieses stellt eine Beziehung zwischen elektrischer Feldstärke und der zeitlichen Änderung der magnetischen Flussdichte her. (2.1c) bezeichnet man als „Gauß'sche Gesetz für Magnetfelder“. Es stellt die Nichtexistenz magnetischer Ladungen dar. Abschließend setzt (2.1d) die magnetische Feldstärke in Beziehung zu Ladungsbewegungen. Dies bezeichnet man als Durchflutungsgesetz oder Ampère'sches Gesetz.

Die Maxwell'schen Gleichungen lassen sich auch in differentieller Form darstellen [8]:

$$\operatorname{div}\vec{D} = \rho \tag{2.2a}$$

$$\operatorname{rot}\vec{E} = -\frac{\partial\vec{B}}{\partial t} \tag{2.2b}$$

$$\operatorname{div}\vec{B} = 0 \tag{2.2c}$$

$$\operatorname{rot}\vec{H} = \vec{J} + \frac{\partial\vec{D}}{\partial t} \tag{2.2d}$$

Die elektrischen und magnetischen Felder sind über die Materialgesetze

$$\vec{D} = \varepsilon_0\vec{E} + \vec{P}, \tag{2.3a}$$

$$\vec{J} = \kappa\vec{E}, \tag{2.3b}$$

$$\vec{B} = \mu_0\vec{H} + \vec{M}, \tag{2.3c}$$

miteinander verbunden.

Dabei bezeichnet in (2.3a) $\varepsilon_0$ die Permittivität im Vakuum und $\vec{P}$ die materialabhängige Polarisation. Der Leitwert $\kappa$ gibt in (2.3b) die spezifische elektrische Leitfähigkeit eines Materials an. Die magnetischen Größen $\vec{B}$ und $\vec{H}$ werden in (2.3c) durch die Permeabilität $\mu$ verbunden. Dies setzt sich analog zur Permittivität aus der Permeabilität im Vakuum $\mu_0$ und der Magnetisierung $\vec{M}$ zusammen. Hierbei ist es möglich, Materialien mit linearen und nichtlinearen Eigenschaften zu betrachten. Als ein Anwendungsbeispiel für Materialien mit nichtlinearer Leitfähigkeit werden in Kap. 2.2 Mikrovaristoren vorgestellt, welche auch Gegenstand aktueller Materialforschung sind.

Die Elektroquasistatik bezeichnet ein Modell, welches in einem Bereich Anwendung findet, in welchem ein elektrisches Feld in der Zeit veränderlich, im Raum zu einem Zeitpunkt jedoch näherungsweise konstant ist. Es findet also keine wellenförmige Ausbreitung im Raum und keine Abstrahleffekte statt [9, 10]. Dies kann dargestellt werden durch die Maxwell-Gleichungen [6] unter folgenden Voraussetzungen:

Damit das quasistatische Modell verwendet werden kann, muss die Zeitkonstante

$$\tau = \frac{1}{2\pi f} \ll \frac{l}{c} \tag{2.4}$$

sein. $l$ ist dabei das Ausmaß des Rechenraums, $c$ die Lichtgeschwindigkeit und $f$ die Frequenz der anregenden Wechselspannung [11].

In dieser Arbeit werden nur elektroquasistatische Probleme betrachtet, d.h., die im System gespeicherte elektrische Energie ist sehr viel größer als die gespeicherte magnetische Energie [12]:

$$\int_0^B \vec{H} \cdot d\vec{B} \ll \int_0^D \vec{E} \cdot d\vec{D}. \tag{2.5}$$

Damit kann $\frac{\partial \vec{B}}{\partial t} \approx 0$ angenommen werden, was bei einsetzen in das Induktionsgesetz ein rotationsfreies elektrische Feld zur Folge hat. Ansatz ist das Ampère'sche Durchflutungsgesetz (2.2d). Durch Anwendung des Divergenz-Operators ergibt sich

$$\operatorname{div}\,(\operatorname{rot}\vec{H}) = \operatorname{div}\,\vec{J} + \operatorname{div}\frac{\partial}{\partial t}\vec{D}. \tag{2.6}$$

Mit div rot $(\cdot) = 0$, welches unter der Voraussetzung eines zusammenziehbaren Gebietes gilt, folgt die Kontinuitätsgleichung

$$0 = \operatorname{div}\,\vec{J} + \operatorname{div}\frac{\partial}{\partial t}\vec{D}. \tag{2.7}$$

Das elektrische Feld $\vec{E}$ kann nun durch das elektrische Potential $\varphi$ mit

$$\vec{E} = -\operatorname{grad}(\varphi) \tag{2.8}$$

als Gradient einer skalaren Größe dargestellt werden.

Durch einsetzen der Gleichungen (2.3a), (2.3b) und (2.8) in (2.7) und Auflösen nach $\varphi$ ergibt die Ausgangsgleichung für elektroquasistatische Probleme:

$$\operatorname{div}\,(\kappa \operatorname{grad}(\varphi)) + \operatorname{div}\left(\varepsilon \operatorname{grad}\left(\frac{\partial \varphi}{\partial t}\right)\right) = 0 \tag{2.9}$$

zuzüglich Rand- und Anfangsbedingungen. Mit der Gleichung (2.9) werden kapazitive und resistive Effekte abgebildet.

Zur numerischen Lösung ist das Problem in Raum und Zeit zu diskretisieren. Die Diskretisierung im Raum kann durch verschiedene Methoden durchgeführt werden, wie z. B. die Methode der Finite Differenzen (FD) [14], die Finite Integrationstechnik (FIT) [15] oder die FEM [16]. Für die Diskretisierung der Zeit werden numerische Zeitintegratoren verwendet. Hierbei ist zwischen impliziten und expliziten Integratoren zu unterscheiden. Beide Typen werden im Kap. 2.4 dargestellt.

## 2.2 Mikrovaristoren als Material mit nichtlinearer Leitfähigkeit

Gegenstand aktueller Forschung in der Hochspannungstechnik ist die Verwendung von Zinkoxid (ZnO)-Varistormaterialien. Diese verändern ihre Leitfähigkeit in Abhängigkeit von der lokalen Feldstärke. Breite Anwendung finden Varistoren in Metalloxid-Überspannungsableitern, welche in Hochspannungsnetzen verwendet werden, um Überspannungen gegen Erde abzuleiten [2, 1].

Ein anderer Ansatz ist die Nutzung von Mikrovaristoren. Im Unterschied zu Überspannungsableitern werden diese verwendet, um das elektrische Feld zu steuern. Dadurch ist ein Stromfluss, wie er bei einer Überspannung bei einem Überspannungsableiter vorkommt, nicht vorgesehen. Bei der Herstellung von Mikrovaristoren werden kleine dotierte ZnO-Partikel mit einer Korngrößen von 5 bis 130 $\mu m$ in eine Silikonmatrix oder einen Lack eingebettet. Abhängig von der anliegenden elektrischen Feldstärke verändert sich auch hier bei jedem Korn die elektrische Leitfähigkeit. Hierbei liegt der Schaltpunkt eines einzelnen Korns bei etwa 3 bis 3.5 V [17, 18, 19]. Dies macht diese vielfältig und weitgehend formunabhängig einsetzbar. Das elektrische Verhalten kann dabei neben dem Material vor allem durch die Menge der in die Basis eingebrachten Mikrovaristoren beeinflusst und reproduzierbar hergestellt werden [20]. Die Nutzung ist daher auf verschiedenen Anwendungen der Hochspannungstechnik und der elektrischen Maschinen erforscht worden, wie etwa Kabelendverschlüssen, Muffen, Hochspannungs-

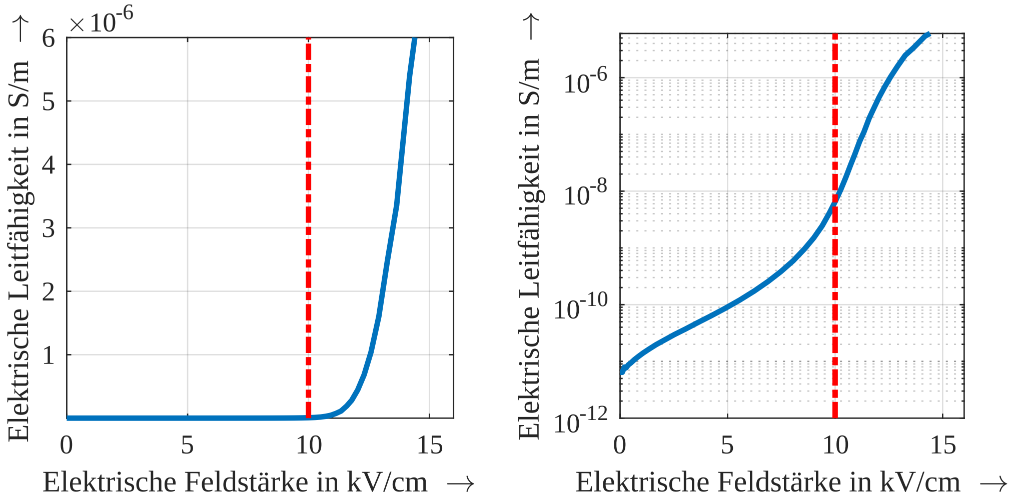


Abbildung 2.2: Leitwert-Kurve von Mikrovaristor-Material des Typs Wacker SLM 79049 [26] in linearer und logarithmischer Skalierung

durchführungen, Transformatoren, Wickelstäben in elektrischen Maschinen und Langstabisolatoren an Hochspannungsleitungen [21, 22, 23, 24, 25].

Abb. 2.2 zeigt beispielhaft die Leitfähigkeit von Mikrovaristoren des Typs Wacker SLM 79049 [26]. Es ist erkennbar, dass das Material bis zu einem sogenannten „Schaltpunkt“ von $|\vec{E}| = 10\,\frac{\mathrm{kV}}{\mathrm{cm}}$ eine relativ niedrige Leitfähigkeit aufweist, welche dann stark ansteigt.

Es ergibt sich also kein konstanter, sondern ein veränderlicher Leitwert. Dieser passt seine Leitfähigkeit in Abhängigkeit des Betrages der elektrischen Feldstärke $|\vec{E}|$ an. Damit verändert sich der Leitwert lokal abhängig in nichtlinearer Weise, was in einem Mikrovaristorblock abhängig von der angelegten Spannung den leitfähigen Bereich ausdehnt und zusammenzieht und das elektrische Feld steuert.

Die nichtlineare Materialabhängigkeit wird in der elektroquasistatischen Gleichung (2.9) berücksichtigt, indem der spezifische Leitwert $\kappa$ wie folgt definiert wird:

$$\kappa = \kappa\Big(\vec{x}, |\vec{E}|\Big) = \kappa\Big(\vec{x}, |\operatorname{grad}(\varphi)|\Big). \tag{2.10}$$

Im Folgenden wird der Leitwert mit $\kappa(\varphi)$ bezeichnet.

Daraus ergibt sich die elektroquasistatische Gleichung unter Berücksichtigung von nichtlinearen Leitfähigkeiten zu

$$\mathrm{div}\,(\kappa(\varphi)\;\mathrm{grad}\,(\varphi)) + \;\mathrm{div}\left(\varepsilon\;\mathrm{grad}\left(\frac{\partial\varphi}{\partial t}\right)\right) = 0. \tag{2.11}$$

Für die Lösung ist eine solche Materialkurve in jedem Orts- und Zeitpunkt auszuwerten. Am besten werden hierzu monotonie-erhaltende Splines genutzt. In der Praxis werden aber oft die einfacheren kubischen Splines [27] verwendet und im Nachhinein auf ihre Monotonie überprüft .

Die sich ergebende Aufgabe besteht nun in der Lösung dieser nichtlinearen parabolischen partiellen Differentialgleichung. Der Linienmethode (engl.: „Method of Lines“) folgend, ist dazu zuerst eine Diskretisierung im Raum erforderlich. Anschließend erfolgt die Lösung im Zeitbereich zu diskreten Zeitpunkten mittels einer Zeitintegrationsmethode [28].

Dies ermöglicht es, das zeittransiente Potential computergestützt zu simulieren und basierend auf den Ergebnissen zum Beispiel Optimierungsaufgaben zu lösen [3, 29],[B2, B3].

## 2.3 Die Finite Elemente Methode in der Elektroquasistatik

Die FEM hat sich als eine der dominierenden Methoden zur räumlichen Diskretisierung für niederfrequente Feldsimulationen etabliert [16]. Vorteile sind die gute Anwendbarkeit für verschiedene, auch komplexe Geometrien und die einfache Verallgemeinerung auf hohe Ordnungen.

Die Methode wurde 1943 von Richard Courant eingeführt, um Randwertprobleme basierend auf partiellen Differentialgleichungen zu lösen [30]. In den 1950er Jahren wurde unabhängig davon eine mehr anwendungsbezogene Methode entwickelt, welche eine Anzahl von Punktmassen durch masselose Sprungfedern verband. Diese wurde in den 1960er - 1970er Dekaden formalisiert und zu einer allgemein anwendbaren Methode für partielle Differentialgleichungen weiterentwickelt [30].

Die Methode der Finiten Elemente hat eine weite Verbreitung in unterschiedlichsten Fachdisziplinen gefunden. Daher sei hinsichtlich der zugrunde liegenden Methodik an dieser Stelle nur auf einen kleinen Ausschnitt relevanter Literatur verwiesen, welche eine Einführung in die Thematik gibt [31, 32, 33, 34, 35, 36, 37, 38].

Im Folgenden soll die Methode kurz anhand der elektroquasistatischen Gleichung (2.9) dargestellt werden. Ausgehend von der Finite-Elemente-Formulierung nach Galerkin [39] wird das Problem zuerst als schwache Form geschrieben. Hierzu wird (2.9) mit einer Testfunktion $v$ gewichtet und über den Raum $\Omega$ integriert:

$$\begin{gathered}\int\limits_{\Omega} v \cdot \operatorname{div}\left(\kappa(\varphi) \operatorname{grad} \varphi\right) d\Omega + \int\limits_{\Omega} v \cdot \operatorname{div}\left(\varepsilon \operatorname{grad} \frac{\partial \varphi}{\partial t}\right) d\Omega = 0 \\ \forall v \in H_0^1(\Omega) := \{v \in L^2(\Omega), v' \in L^2(\Omega), v_{|\partial\Omega} = 0\}.\end{gathered} \tag{2.12}$$

Dabei ist $H_0^1(\Omega)$ der Sobolevraum, und $L^2(\Omega)$ der Raum aller quadratintegrablen Funktionen.

Mit der ersten Green'schen Identität erhält man

$$\begin{aligned}&\int\limits_{\Omega} \operatorname{grad} v \cdot \left(\kappa(\varphi) \operatorname{grad} \varphi\right) d\Omega + \int\limits_{\Omega} \operatorname{grad} v \cdot \left(\varepsilon \operatorname{grad} \frac{\partial \varphi}{\partial t}\right) d\Omega = \\ &\qquad \int\limits_{\partial\Omega} \left(\kappa(\varphi) \frac{\partial \varphi}{\partial \vec{n}} \cdot v + \varepsilon \frac{\partial}{\partial \vec{n}} \left(\frac{\partial \varphi}{\partial t}\right) \cdot v\right) dS,\end{aligned} \tag{2.13}$$

mit dem Normalenvektor $\vec{n}$ zum Rand des Rechengebietes $\Omega$. Die rechte Seite ist dabei aufgrund der homogenen Dirichletrandbedingungen gleich Null [8]. Es ergibt sich das Variationsproblem: Finde $\varphi \in H_0^1(\Omega)$, sodass (2.13) für alle $v \in H^1(\Omega)$ gilt.

Für eine diskrete Variationsformulierung werden $v(\vec{x})$ und $\varphi(\vec{x}, t)$ durch eine endliche Anzahl Knotenbasisfunktionen $N_j$ approximiert, für welche

$$N_j(\vec{x}_i) = \begin{cases} 1, & i = j \\ 0, & i \neq j \end{cases}, \qquad i, j = 1, ..., n_{tot} \tag{2.14}$$

gilt, mit $n_{tot}$ als der Anzahl der Knoten des 3D-FEM-Gitters und $\vec{x}_i$ als dem Ort des Gitterknotens.

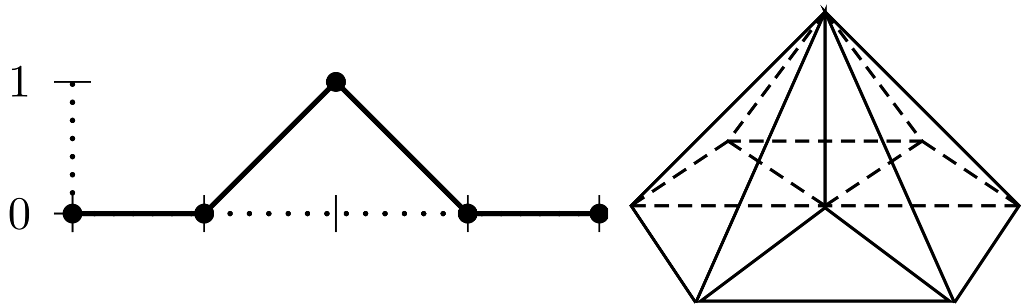


Abbildung 2.3: Darstellung der Ansatzfunktion in einer 1D- und einer 2D-Diskretisierung

Das elektrische Potential schreibt sich somit als

$$\varphi(\vec{x},t) \approx \sum_{j=1}^{n_{tot}} u_j(t) N_j(\vec{x}), \tag{2.15}$$

mit den Freiheitsgraden $u_j$, die im Fall niedrigster Ordnung dem elektrischen Potential im Knoten $j$ entsprechen (vgl. Abb. 2.3) und $u_j = 0$ am Dirichlet-Rand ist. Wir führen hierzu die Vektoren $\mathbf{u} \in \mathbb{R}^n$ mit $n$ Freiheitsgraden und $\mathbf{u}_D \in \mathbb{R}^{n_D}$ mit $n_D$ Freiheitsgraden ein. $n_D$ gibt die Anzahl an Dirichlet-Knoten an. Die Anzahl aller Knoten eines diskretisierten Problems $n_{tot}$ ergibt sich aus $n_{tot} = n + n_D$. Hierbei geben bei elektroquasistatischen Berechnungen Dirichlet-Ränder klassischerweise das Potential spannungsführender Teile – inhomogener Dirichlet-Rand – oder geerdeter Teile – homogener Dirichlet-Rand – an. Homogene Neumann-Ränder modellieren den Übergang in den freien Raum außerhalb des Rechengebietes. Einsetzen von (2.15) in (2.13) ergibt $i = 1, ..., n_{tot}$ Gleichungen

$$\begin{aligned} \sum_{j=1}^{n_{tot}} \Bigg( u_j \int_{\Omega} \text{grad } N_i \cdot \kappa(\cdot) \text{ grad}\, N_j \, d\Omega \, + \\ \frac{\partial}{\partial t} u_j \int_{\Omega} \text{grad } N_i \cdot \varepsilon \text{ grad}\, N_j \, d\Omega \Bigg) \; = \; 0. \end{aligned} \tag{2.16}$$

Dies führt auf ein (nichtlineares) algebraisches Gleichungssystem in Matrixform. Die Matrizen werden elementweise aufgebaut („assembliert“). Die

Koeffizienten der Leitwertmatrix $\mathbf{K}_{\kappa,tot} \in \mathbb{R}^{n_{tot}\times n_{tot}}$ und Permittivitätsmatrix $\mathbf{M}_{\varepsilon,tot} \in \mathbb{R}^{n_{tot}\times n_{tot}}$ sind elementweise definiert mit

$$[\mathbf{K}_{\kappa,tot}]_{i,j}(\cdot) = \int\limits_{\Omega} \text{grad } N_i \cdot \kappa(\cdot) \text{ grad}\, N_j \, d\Omega; \tag{2.17a}$$

$$[\mathbf{M}_{\varepsilon,tot}]_{i,j} = \int\limits_{\Omega} \text{grad } N_i \cdot \varepsilon \text{ grad}\, N_j \, d\Omega. \tag{2.17b}$$

Die so berechneten Matrizen sind symmetrisch und positiv definit. Die Unterteilung des Raums in Elemente kann dabei durch Elemente mit verschiedenen Formen, wie (gekrümmten) Tetraedern oder Hexaedern, erfolgen. Für Tetraeder wird ein Element mit Ansatzfunktionen erster Ordnung durch Freiheitsgrade auf den vier Ecken definiert. Die Matrix $\mathbf{M}_{\varepsilon,tot}$ wird in der Praxis elementweise aufgebaut. Das lässt sich durch die Summe über die Elemente

$$\mathbf{M}_{\varepsilon,tot} = \sum_{e=1}^{n_E} \mathbf{Q}_e{}^T \mathbf{M}_{\varepsilon,e} \mathbf{Q}_e \tag{2.18}$$

mit $n_E$ der Anzahl aller Elemente darstellen, wobei $\mathbf{M}_{\varepsilon,e} \in \mathbb{R}^{4\times 4}$ der lokale Beitrag ist und $\mathbf{Q}_e \in \mathbb{R}^{4\times n_{tot}}$ die Inzidenzmatrix, die globale und lokale Nummerierung aufeinander abbildet. Die Assemblierung skaliert linear mit der Anzahl der Elemente. Der Rechenaufwand je Element ist konstant, wie (2.16) zu entnehmen ist. Folglich ergibt sich eine Rechenkomplexität für die Assemblierung von $\mathcal{O}(n_E) = \mathcal{O}(n)$.

Um dies nun für beliebig geformte Tetraeder unterschiedlicher Größe durchführen zu können, werden die Elemente auf ein Einheitstetraeder transformiert [33]. Die Koeffizienten für Transformation und Integration aus (2.17) sind bekannt [40] und können im Simulationsprogramm hinterlegt werden.

Im Falle von nichthomogenen Dirichlet-Randbedingungen können die entsprechenden Koeffizienten separiert und als eigene Matrix auf die rechte Seite von (2.16) gebracht werden. Durch Aufspalten der Integrale und Multiplikation mit den vorgegebenen Randwerten ergibt sich

$$\mathbf{M}_\varepsilon \frac{d\mathbf{u}(t)}{dt} + \mathbf{M}_{\varepsilon,D} \frac{d\mathbf{u}_D(t)}{dt} + \mathbf{K}_\kappa \Big(\mathbf{u}(t)\Big)\ \mathbf{u}(t) + \mathbf{K}_{\kappa,D}\mathbf{u}_D(t) = 0. \tag{2.19}$$

Die Matrix $\mathbf{M}_{\varepsilon,tot}$ wird in eine Matrix $\mathbf{M}_\varepsilon \in \mathbb{R}^{n\times n}$ für die freien Knoten und eine Matrix $\mathbf{M}_{\varepsilon,D} \in \mathbb{R}^{n\times n_D}$ für die Dirchlet-Knoten geteilt. Analog wird die Matrix $\mathbf{K}_{\kappa,tot}$ in $\mathbf{K}_\kappa \in \mathbb{R}^{n\times n}$ für die freien Knoten und eine Matrix $\mathbf{K}_{\kappa,D} \in \mathbb{R}^{n\times n_D}$ für die Dirchlet-Knoten geteilt. Bei dieser Formulierung wird davon ausgegangen, dass keine nichtlinear leitfähigen Materialen in den Dirichlet-Knoten des Gitters vorhanden sind. Daher wird $\mathbf{K}_{\kappa,D} = const$ angenommen. Daraus ergibt sich $\mathbf{b}(t) \in \mathbb{R}^n$ zu

$$\mathbf{b}(t) = -\mathbf{M}_{\varepsilon,D}\frac{d}{dt}\mathbf{u}_D(t) - \mathbf{K}_{\kappa,D}\mathbf{u}_D(t) \tag{2.20}$$

und die diskretisierte elektroquasistatische Gleichung mit Beachtung der Randbedingungen als Anregungsterm auf der rechten Seite zu

$$\mathbf{M}_\varepsilon \frac{d\mathbf{u}(t)}{dt} + \mathbf{K}_\kappa\Big(\mathbf{u}(t)\Big)\mathbf{u}(t) = \mathbf{b}(t). \tag{2.21}$$

Der Anregungsterm $\mathbf{b}(t)$ ist zu jedem Zeitpunkt $t$ aus den Randwerten bestimmbar. $\mathbf{M}_{\varepsilon,D}$ und $\mathbf{K}_{\kappa,D}$ sind dabei – unter der Voraussetzungen, dass keine nichtlinearen Materialien an Dirichlet-Knoten grenzen – nur einmal aufzustellen. Bei Dirichlet-Rändern werden zumeist gleiche Werte auf ganzen Flächen vorgegeben, etwa ein Spannungswert für komplette spannungsführende Teile. Damit gilt für die Anzahl der vorgegebenen Dirichlet-Werte $n_D$, dass diese $n_D \ll n$ sind.

Unter Beachtung der Tatsache, dass geerdete Teile mit einem Potential von 0 V keinen Beitrag zu $\mathbf{b}(t)$ liefern, ergibt sich für ein einphasiges Modell $\mathbf{M}_{\varepsilon,D} \in \mathbb{R}^{n\times 1}$ und $\mathbf{K}_{\kappa,D} \in \mathbb{R}^{n\times 1}$ und für ein dreiphasiges Modell $\mathbf{M}_{\varepsilon,D} \in \mathbb{R}^{n\times 3}$ und $\mathbf{K}_{\kappa,D} \in \mathbb{R}^{n\times 3}$.

Im Simulationscode Magneto-/Electro-Quasistatic Simulation Code (MEQSICO) [10] wird die Diskretisierung im Raum mittels der Methode der Finiten Elemente durchgeführt. Die Umsetzung in MEQSICO ist in Abb. 2.4 dargestellt. Der Vollständigkeit halber wird hier die Zeitdiskretisierung mit betrachtet, welche im folgenden Kap. 2.4 behandelt wird.

```
function MEQSICO(Modell,Endzeitpunkt)
    Diskretisierung des Models mittels FEM.
    Festlegung der Freiheitsgrade und Dirichlet-Ränder.
    Integration der schwachen Formulierung.
    Einarbeitung der Randbedingungen.
    while Endzeitpunkt noch nicht erreicht do
        Berechne Zeitschritt $t_i \to t_{i+1}$ {
        Berechnung der rechten Seite.
        Aufstellen der linearisierten Systemmatrix.    ▷ für implizite Zeitintegrationsverfahren
        Lösung des linearen Gleichungssystems.
        Nichtlineare Korrektur.    ▷ für implizite Zeitintegrationsverfahren
        Berechnen der Lösung $\mathbf{u}_{i+1}$.
        }
    end while
    Speichern und Auswerten der Ergebnisse (Postprocessing).
end function
```

Abbildung 2.4: Algorithmus für einen Simulationsdurchlauf.

Mit [29] wurden Ansatzfunktionen zweiter Ordnung in MEQSICO integriert. Sie definieren ein Tetraeder wie Ansatzfunktionen erster Ordnung durch vier Freiheitsgrade auf den vier Ecken des Elementes. Zusätzlich werden jedoch noch weitere sechs Freiheitsgrade auf den sechs Kanten des Tetraederelements betrachtet. Somit ist die Dimension der lokalen Matrizen eines Elementes mit einer Ansatzfunktion zweiter Ordnung (2.17) $\mathbf{K}_{\kappa,e}$ , $\mathbf{M}_{\varepsilon,e} \in \mathbb{R}^{10\times10}$.

## 2.4 Zeitintegratoren

Zur Lösung eines elektroquasistatischen Problems (2.9) ist nicht nur eine Diskretisierung im Raum erforderlich, wie dies in Kap. 2.3 beschrieben ist. Zusätzlich ist die Diskretisierung in der Zeit von nöten. Die sich ergebende Differentialgleichung kann bei gegebenen Anfangs- und Randbedingung gelöst werden. Hierzu wird ein Anfangswert $\mathbf{u}_0$ bzw. $\varphi_0$ im zu berechnenden Gebiet $\Omega$ festgelegt. Als Randbedingungen werden Dirichlet- und Neumann-Randbedingungen festgelegt. Die Dirichlet-Randbedingungen sind im Vektor

$\mathbf{u}_D(t)$ enthalten. $\mathbf{u}_D(t)$ ändert sich über der Zeit bei zeitveränderlichen Potentialen, wie z. B. bei der Anregung mit einer sinusförmigen Spannung. Diese Randwerte bilden den Right Hand Side (RHS)-Vektor $\mathbf{b}(t)$ (vgl. Kap. 2.3). Das Aufstellen von $\mathbf{b}(t)$ in jedem Zeitschritt stellt daher eine Aufgabe von geringem Rechenaufwand dar, wie in Kap. 2.3 dargestellt.

Zur Zeitintegration können nun verschiedene Verfahren angewandt werden, um zu einer Lösung $\mathbf{u}(t)$ zu gelangen. Eine einfache Methode hierfür ist die explizite einstufige Zeitintegration nach Euler. In ihrer einfachsten Form wird ausgehend von einer Lösung zum Zeitpunkt $t$ mit Hilfe einer finiten Differenz

$$\mathbf{f}(t, \mathbf{u}) = \frac{d\mathbf{u}(t)}{dt} \approx \frac{\mathbf{u}(t + \Delta t) - \mathbf{u}(t)}{\Delta t} \tag{2.22}$$

die Lösung zum neuen Zeitpunkt ermittelt. Hieraus folgt, dass sich die Lösung über der Zeit auch als Folge darstellen lässt, für welche von einer Ausgangslösung $\mathbf{u}_i$ die Lösung $\mathbf{u}_{i+1}$ mit

$$\mathbf{u}_{i+1} = \mathbf{u}_i + \Delta t\, \mathbf{f}(t_i, \mathbf{u}_i) \qquad i = 1, ..., M \tag{2.23}$$

ermittelt wird. Hierbei wird

$$\mathbf{f}(t_i, \mathbf{u}_i) = \mathbf{M}_\varepsilon{}^{-1}\Big(\mathbf{b} - \mathbf{K}_\kappa(\mathbf{u}_i)\mathbf{u}_i\Big) , \tag{2.24}$$

mit $0 < t_1 < t_2 < .. < t_M = T$ und $\mathbf{u}_{i+1} \approx \mathbf{u}(t_i + \Delta t)$ verwendet.

Die Konvergenzgeschwindigkeit der Lösung lässt sich durch Integrationsverfahren höherer Ordnung verbessern. Einen umfassenden Einblick in die Lösung dieser Probleme gibt [13]. Ein Problem der expliziten Verfahren ist, dass sie instabil werden können [41].

Der sogenannte Stabilitätsbereich gibt den Bereich an, in welchem ein Verfahren für ein System gewöhnlicher Differentialgleichungen in der Zeit der

$$\mathbf{M}_\varepsilon\, \mathbf{u}' + \mathbf{K}_\kappa(\mathbf{u})\, \mathbf{u} = \mathbf{b} , \tag{2.25}$$

stabil bleibt, also die numerische Lösung nicht aufschwingt, wobei $\mathbf{K}_\kappa$ symmetrisch positiv definit ist. Durch Umstellen der Ausgangsfunktion (2.22)

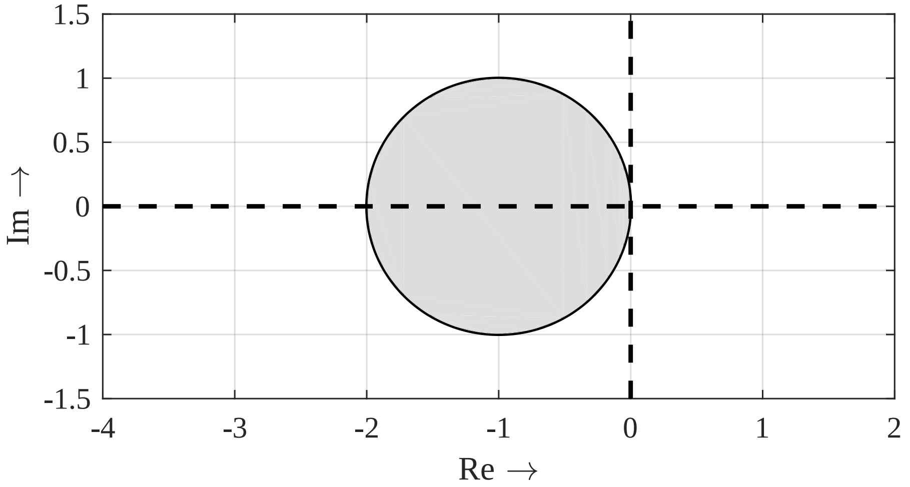


Abbildung 2.5: Stabilitätsgebiet des expliziten Euler-Verfahrens. Stabiler Bereich in Grau.

erhält man $\mathbf{u}' = \mathbf{J}\mathbf{u}$, mit der Jacobi-Matrix $\mathbf{J}$. Diese wird mittels Eigenwertzerlegung in Diagonalform gebracht. Da hier nun die einzelnen Werte entkoppelt sind, wird im Folgenden die skalare Testgleichung betrachtet:

$$y_j' = \lambda_j y_j, \quad \lambda_j \in \mathbb{C} \quad j = 1, ..., n \; . \tag{2.26}$$

Mit dem Eigenwert $\lambda$ erhält man die sogenannte Dahlquist'sche Testfunktion [42]. Dies lässt sich mittels Runge-Kutta-Verfahren als

$$y_i = R(z)^i y_0 \tag{2.27}$$

darstellen, wobei $R(z)$ – die Stabilitätsfunktion der Methode – eine rationale Funktion und $z = \Delta t \cdot \lambda$ ist. Für das explizite Euler-Verfahren ergibt sich

$$y_i = (1 + z)y_{i-1} = (1 + z)^i y_0. \tag{2.28}$$

Durch die Bedingung $|R(z)| \leqslant 1$ bleibt $y_i$ beschränkt und definiert damit das Stabilitätsgebiet des expliziten Eulerverfahrens – einen Kreis mit Radius 1 um den Wert -1. Der größte Wert für $z$ darf also 0, der kleinste -2 sein.

Abb. 2.6 zeigt das Verhalten bei drei verschiedenen Werten von $z$. Dabei wird das Verhalten in der komplexen Ebene sowie das Verhalten des Realteils über den Iterationsschritten aufgezeigt. Bei $|z| < -2$ zeigt sich ein

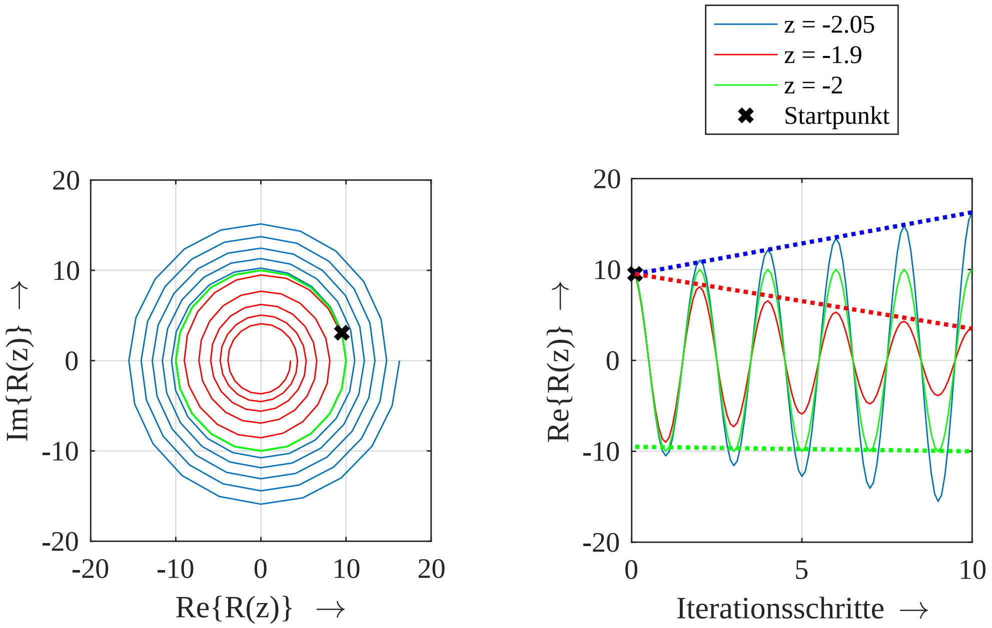


Abbildung 2.6: Stabilitätsverhalten des expliziten Euler-Verfahrens für verschiedene $z$.

zusammenlaufender Kreis in der komplexen Ebene, beziehungsweise eine gedämpfte Schwingung über den Iterationsschritten. Beim $|z| = 2$ Bleibt die Kreisgröße in der komplexen Ebene konstant. Bei $|z| > -2$ zeigt sich, wie das Problem mit zunehmender Iterationszahl divergiert. Der Kreis in der komplexen Ebene weitet sich aus. Die Schwingung des Realteils über den Iterationen verhält sich aufschwingend.

Die Eigenwerte $\lambda_j$ raumdiskretisierter parabolischer Probleme sind bei dem in Kap. 2.3 verwendeten Ritz-Galerkin-Ansatz aufgrund der reellwertigen symmetrischen Matrix reell und wachsen proportional zur Kantenlänge entlang der negativen reellen Achse der komplexen Ebene mit $1/(\Delta x)$ [42]. Dies führt für das explizte Euler-Verfahren zur Courant-Friedrich-Lewy (CFL)-Bedingung $\Delta t_{CFL} \leqslant C(\Delta x)^2$ [42], welche erfüllt sein muss, damit es stabil bleibt [41]. Dies kann bei hochauflösenden Modellen ($\Delta x \ll 1$) zu einer sehr kleinen Zeitschrittweite und damit zu einer sehr hohen Anzahl an Zeitschritten führen. Probleme, welche numerisch instabil sind, und

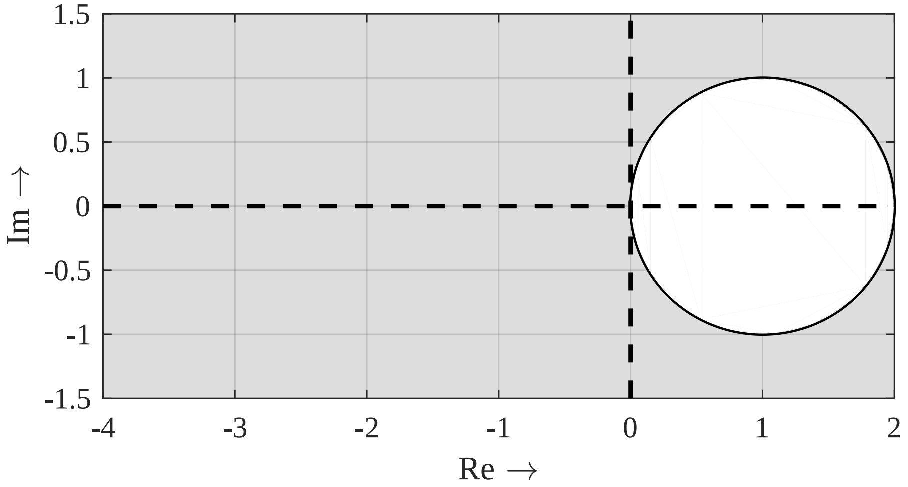


Abbildung 2.7: Stabilitätsgebiet des impliziten Euler-Verfahrens. Stabiler Bereich in Grau.

nur durch extrem kleine Zeitschrittweiten zu lösen sind, werden als „steife Probleme“ bezeichnet [43].

Eine Alternative zu expliziten Verfahren stellen implizite Zeitintegrations-Verfahren, z. B. das implizite Euler-Verfahren, dar. Hier wird ausgehend von der zu errechnenden Lösung $\mathbf{u}_{i+1}$ ein linksseitiger Differenzenquotient aufgestellt. Die Funktion wird im Unterschied zu den expliziten Methoden nicht am Ausgangszeitpunkt $t$, sondern am zu errechnenden Zeitpunkt $t+\Delta t$ ausgewertet. Bei der Lösung von nichtlinearen Problemen ist hier allerdings zusätzlich ein nichtlineares algebraisches Gleichungssystem pro Zeitschritt zu lösen. Hierfür sind Linearisierungsmethoden erforderlich, die iterativ auf einer wiederholten Lösung von algebraischen Gleichungssystemen basieren, was mit zusätzlichem Rechenaufwand verbunden ist.

Dieser Ansatz wurde von Curtis und Hirschfelder [44] 1952 eingeführt und ist insbesondere seit den Arbeiten von Gear [45] als Backward-Differentiation Formulas (BDF) bekannt [43]. Bei den BDF handelt es sich um Mehrschritt-Verfahren, wobei das implizite Euler-Verfahren als Sonderform nur den aktuellen und den zu errechnenden Schritt betrachtet, daher ein Ein-Schritt-

Verfahren ist. Es ergibt sich für das implizite Euler-Verfahren (bzw. die BDF-Methode erster Ordnung)

$$\mathbf{u}_{i+1} = \mathbf{u}_i + \Delta t\,\mathbf{f}(t + \Delta t, \mathbf{u}_{i+1}), \tag{2.29}$$

wobei

$$\mathbf{f}(t + \Delta t, \mathbf{u}_{i+1}) = \mathbf{M}_\varepsilon{}^{-1}\Big(\mathbf{b} - \mathbf{K}_\kappa(\mathbf{u}_{i+1})\mathbf{u}_{i+1}\Big),$$

mit dem Vektor $\mathbf{u}_i$, $0 < t_1 < t_2 < .. < t_m = T$ und $\mathbf{u}_{i+1} \approx \mathbf{u}(t_i + \Delta t)$.

Da sich (2.29) nicht explizit nach $\mathbf{u}_{i+1}$ auflösen lässt, ist hier nun in jedem Zeitschritt ein nichtlineares Gleichungssystem zu lösen. Beinhaltet das Problem nun zusätzlich noch nichtlineare Materialien, wie etwa ein Material mit nichtlinearer Leitfähigkeit $\kappa(\mathbf{u})$, ist zusätzlich eine nichtlineares Lösungsverfahren anzuwenden. In MEQSICO wird hierzu ein Newton-Raphson-Verfahren [46] verwendet.

Bei dem impliziten Euler-Verfahren handelt es sich um ein sogenanntes A-stabiles Verfahren. Diese Bezeichnung wurde durch Dahlquist [47] eingeführt und bezeichnet ein Zeitintegrationsverfahren dessen Stabilitätsfunktion die gesamte reell negative komplexe Halbebene $\mathbb{C}^-$ einnimmt. Ein solches Verfahren wird unabhängig von der gewählten Zeitschrittweite nicht instabil, was ein großer Vorteil hinsichtlich der Robustheit ist. Eine Erweiterung der A-Stabilität stellt die L-Stabilität dar. Ein L-stabiles Verfahren ist A-stabil und strebt weiterhin für

$$\lim_{z \to \infty} R(z) = 0. \tag{2.30}$$

L-stabile Verfahren sind nicht nur schrittweitenunabhängig stabil, sondern dämpfen zusätzlich transiente Komponenten effizient [43]. Sowohl beim impliziten Eulerverfahren, als auch bei den um folgenden Betrachteten impliziten Zeitintegrationsverfahren SDIRK3(2), handelt es sich um L-stabile Verfahren [43, 48].

### 2.4.1 Implizite Zeitintegration

Mit einem $s$-stufigen Runge-Kutta-Verfahren kann ein Anfangswertproblem – wie in (2.25) – auch für Lösungen höherer Ordnung gelöst werden. Dabei gibt $s$ die Anzahl der Stufen (engl.: „Stages“) an, welche in jedem Zeitschritt errechnet werden müssen. Zur Ermittlung der Lösung des nächsten Zeitschrittes muss

$$\mathbf{u}_{i+1} = \mathbf{u}_i + \Delta t \sum_{j=1}^{s} b_j \mathbf{k}_j \tag{2.31}$$

mit

$$\mathbf{k}_j = \mathbf{f}\left( t_i + c_j \Delta t \ , \ \mathbf{u}_i + \Delta t \sum_{l=1}^{j-1} a_{jl} \mathbf{k}_l \right) \tag{2.32}$$

errechnet werden [49].

Die Koeffizienten $a_{jl}$, $b_j$, und $c_j$ lassen sich im „Butcher-Tableau“

$$\begin{array}{c|c} \mathbf{c} & \mathbf{A}_B \\ \hline & \mathbf{b}_B^T \end{array} = \begin{array}{c|cccc} c_1 & a_{11} & a_{12} & \dots & a_{1s} \\ c_2 & a_{21} & a_{22} & \dots & a_{2s} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ c_s & a_{s1} & a_{s2} & \dots & a_{ss} \\ \hline & b_1 & b_2 & \dots & b_s \end{array}$$

darstellen.

Ist die obere Dreiecksmatrix von $\mathbf{A}_B$ besetzt, so handelt es sich um ein Implicit Runge-Kutta (IRK)-Verfahren. Da durch die Besetzung der oberen Dreiecksmatrix Rückkopplung von den höheren an die niedrigeren Stufen vorhanden sind, ist in jedem Zeitschritt ein nichtlineares Gleichungssystem der Größe $(n \cdot s) \times (n \cdot s)$ bei $s$ Stufen und $n$ Freiheitsgraden des Problems zu lösen. Um dies sequentiell lösen zu können, haben sogenannte Diagonal Implicit Runge-Kutta (DIRK)-Verfahren die Eigenschaft, keine Einträge in der oberen Dreiecksmatrix zu besitzen [43]. Das Butcher-Tableau ändert sich zu

$$\begin{array}{c|c} \mathbf{c} & \mathbf{A}_B \\ \hline & \mathbf{b}_B^T \end{array} = \begin{array}{c|cccc} c_1 & a_{11} & 0 & \dots & 0 \\ c_2 & a_{21} & a_{22} & \dots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ c_s & a_{s1} & a_{s2} & \dots & a_{ss} \\ \hline & b_1 & b_2 & \dots & b_s \end{array} .$$

Hierdurch können die einzelnen $s$ Stufen hintereinander berechnet werden. Als weiteren Vorteil gegenüber den DIRK-Verfahren hat die Klasse der Singly Diagonal Implicit Runge-Kutta (SDIRK)-Verfahren den selben Eintrag auf der Hauptdiagonalen, sodass Faktorisierung bzw. Präkonditionierungsoperatoren der Systemmatrix $s$-fach wiederholt verwendet werden können [43]. Im Butcher-Tableau haben SDIRK-Verfahren die Struktur

$$\begin{array}{c|c} \mathbf{c} & \mathbf{A}_B \\ \hline & \mathbf{b}_B^T \end{array} = \begin{array}{c|cccc} c_1 & a_d & 0 & \dots & 0 \\ c_2 & a_{21} & a_d & \dots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ c_s & a_{s1} & a_{s2} & \dots & a_d \\ \hline & b_1 & b_2 & \dots & b_s \end{array} .$$

Bei dem in dieser Arbeit verwendeten impliziten Integrationsverfahren SDIRK3(2) [48] handelt es sich um ein 4-stufiges Zeitintegrationsverfahren. Dabei liefert das Verfahren nach s=2 bereits eine Lösung zweiter Ordnung und nach s=4 eine Lösung dritter Ordnung. Hierdurch kann sofort ein Fehlerschätzer über die Beziehung $\mathbf{e} \leq |\hat{\mathbf{u}}(t) - \mathbf{u}(t)|$ konstruiert werden, wobei $\mathbf{e}$ den Vektor der lokalen Integrationsfehler angibt, $\hat{\mathbf{u}}(t)$ die Lösung zweiter Ordnung aus dem SDIRK3(2)-Verfahren und $\mathbf{u}(t)$ die Lösung dritter Ordnung [48].

### 2.4.2 Explizite Zeitintegration

Explizite Verfahren haben den Vorteil, dass die nichtlineare, von $\mathbf{u}(t)$ abhängige, Leitwertmatrix am Startpunkt $\mathbf{u}_t$ und nicht am zu errechnenden Zeitpunkt $\mathbf{u}_{t+1}$ ermittelt wird. Es entfällt die Newton-Raphson- oder Fixpunkt-Iteration zur Lösung des nichtlinearen algebraischen Problems. Dass diese Verfahren bei Zeitschrittweite größer $\Delta t_{CFL}$ außerhalb ihres Stabilitätsbereiches operieren, stellte einen wesentlichen Nachteil dar. Häufig ist der stabile Zeitschritt so klein, dass die Nutzung ineffizient gegenüber impliziten Verfahren wird. Einen Überblick über explizite Runge-Kutta-Verfahren gibt [50].

Einen Kompromiss zwischen der expliziten und der impliziten Zeitintegration sollen die sogenannten stabilisierten expliziten Runge-Kutta Methoden erreichen. Mit ihnen sollen die Vorteile der expliziten Zeitintegration, das Umgehen der Lösung einer Vielzahl von Gleichungssystemen mit integrierter

Newton-Raphson-Iteration, mit der Möglichkeit zu größeren Zeitschritten verbunden werden. Explizite stabilisierte Runge-Kutta Methoden zielen darauf, das Stabilitätsgebiet entlang der negativen reellen Achse zu erweitern und so die Zeitschrittweite, welche bei klassischen expliziten Zeitintegrationsmethoden beschränkt ist, zu erweitern. Dabei sind diese Methoden so aufgebaut, dass zuerst Stabilitätspolynome konstruiert werden, welche einen möglichst langen Streifen entlang der negativen reellen Achse abdecken. Tschebyscheff Polynome bieten sich hier aufgrund ihres rekursiven Aufbaus an [42]. Anschließend wird aus diesem Stabilitätspolynom eine numerische Zeitintegrationsmethode konstruiert. Nach der Grundidee, welche erstmals in [51, 52, 53] vorgestellt wurde, wird eine Anzahl an expliziten Methoden zu einem langen stabilen Zeitschritt kombiniert [42].

Einer der wichtigsten Vertreter ist das Runge-Kutta-Chebychev (RKC)-Verfahren [54]. Es erhöht die Anzahl an Stufen dynamisch in Abhängigkeit von der Steifigkeit des Problems, welche in Bezug zum betragsmäßig größten Eigenwert und der Zeitschrittweite steht. Zur Berechnung sind jedoch mindestens zwei Stufen erforderlich [55]. Dabei erfolgt die Berechnung rekursiv unter Wiederverwendung der Vektoren, sodass der Speicherplatzbedarf unabhängig von der Anzahl der Stufen ist. Ein Zeitschritt berechnet sich dabei für $s$ Stufen wie folgt:

$$\mathbf{y}_0 = \mathbf{u}_i, \tag{2.33a}$$

$$\mathbf{y}_1 = \mathbf{y}_0 + \tilde{\mu}_1 \, \Delta t \mathbf{f}(t_i \, , \, \mathbf{u}_i), \tag{2.33b}$$

$$\begin{aligned} \mathbf{y}_j = &(1 - \mu_j - \nu_j)\mathbf{y}_0 + \mu_j \mathbf{y}_{j-1} + \nu_j \mathbf{y}_{j-2} + \\ &\tilde{\mu}_j \, \Delta t \, \mathbf{f}(t_i + \Delta t c_{j-1} \, , \, \mathbf{y}_{j-1}) + \tilde{\gamma}_j \Delta t \, \mathbf{f}(t_i \, , \, \mathbf{u}_i), \qquad \text{für } j = 2, .., s \end{aligned} \tag{2.33c}$$

$$\mathbf{u}_{i+1} = \mathbf{y}_s s. \tag{2.33d}$$

Alle Koeffizienten sind dabei für $s \geq 2$ in analytischer Form verfügbar und wie folgt definiert [56]:

$T_j$ ist das Tschebyscheff-Polynom erster Art vom Grad $j$. Dann gilt

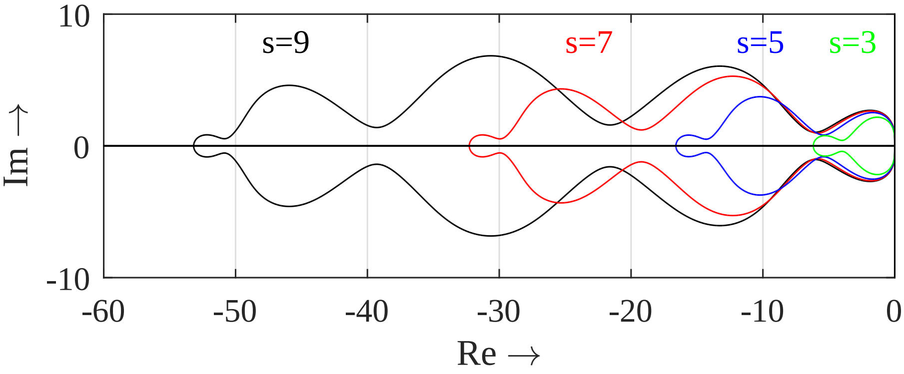


Abbildung 2.8: Rand des Stabilitätsgebiets des RKC-Verfahrens für verschiedene Stufenanzahlen.

$$\varepsilon_{RKC} = \frac{2}{13}, \quad w_0 = 1 + \frac{\varepsilon_{RKC}}{s^2}, \quad w_1 = \frac{T_s'(w_0)}{T_s''(w_0)},$$
$$b_j = \frac{T_j''(w_0)}{(T_j'(w_0))^2} \quad (2 \le j \le s), \quad b_0 = b_2, \quad b_1 = b_2,$$

und

$$\tilde{\mu}_1 = b_1 w_1, \quad \mu_j = \frac{2b_j w_0}{b_{j-1}}, \quad \nu_j = \frac{-b_j}{b_{j-2}}, \quad \tilde{\mu}_j = \frac{2b_j w_1}{b_{j-1}},$$
$$\tilde{\gamma}_j = -(1 - b_{j-1}T_{j-1}(w_0))\tilde{\mu}_j \quad \text{für } (2 \le j \le s).$$

Die Stützstellen $c_j$, welche in (2.33c) für $\mathbf{f}(t_i + \Delta t c_{j-1}\,, \mathbf{y}_{j-1})$ benötigt werden, ergeben sich aus

$$c_j = \frac{T_s{}'(w_0)}{T_s''(w_0)} \frac{T_j''(w_0)}{T_j'(w_0)} \approx \frac{j^2-1}{s^2-1} \text{ für } (2 \le j \le s-1), c_1 = \frac{c_2}{T_2'(w_0)} \approx \frac{c_2}{4}, c_s = 1.$$

Aus (2.33) ist der rekursive Aufbau des Verfahrens ersichtlich. Die einzelnen Koeffizienten können in gleicher Bezeichnung [56] entnommen werden. Das Verfahren wurde hinsichtlich Konvergenz und Stabilität in [57] analysiert.

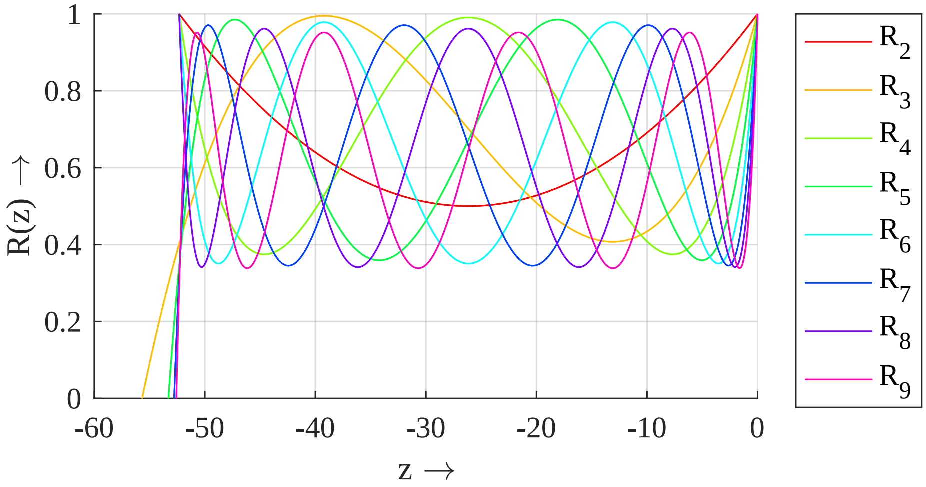


Abbildung 2.9: Rekursive Stabilitätspolynome des RKC-Verfahrens für die Stufenzahl s=9.

In Abb. 2.8 ist das Stabilitätsgebiet für das RKC-Verfahren mit drei, fünf, sieben und neuen Stufen dargestellt. Man erkennt die Erweiterung des Stabilitätsbereiches entlang der negativen reellen Achse mit steigender Stufenanzahl dessen Ränder sich mit $\beta(s) = 0,653s^2$ approximieren lassen [58]. Aus Darstellungsgründen ist der stabile Bereich nicht eingefärbt. Er befindet sich jedoch analog zu Abb. 2.5, dem expliziten Euler-Verfahren, innerhalb der jeweiligen Randmarkierung.

Das rekursive Stabilitätspolynom ist gegeben durch

$$\begin{aligned} R_0(z) =& 1, \\ R_1(z) =& 1 + \tilde{\mu}_1 z, \\ R_j(z) =& (1 - \mu_j - \nu_r) + \mu_j \, R_{j-1}(z) + \\ & \nu_j \, R_{j-2}(z) + \tilde{\mu}_j \, R_{j-1}(z) \, z - \tilde{\gamma}_j z \qquad (2 \leq j \leq s). \end{aligned} \tag{2.34}$$

Die Koeffizienten von (2.33) gelten analog.

Abb. 2.9 zeigt die rekursiv aufgebauten Polynome $R_j(z)$ mit $j = 2, ..., 9$ aus (2.34) für eine Stufenzahl von $s = 9$.

Eine Implementation wurde in [58] vorgestellt. Für die Validierung der errechneten Ergebnisse und die Steuerung der Schrittweite ist ein Fehlerschätzer für das RKC-Verfahren erforderlich. Dieser wird in [56] basierend auf den Erkenntnissen aus [57] gegeben durch

$$\mathbf{e}_{i+1} = \frac{1}{15}[12(\mathbf{u}_i - \mathbf{u}_{i+1}) + 6\Delta t(\mathbf{f}(t_i, \mathbf{u}_i) + \mathbf{f}(t_{i+1}, \mathbf{u}_{i+1}))] \tag{2.35}$$

und in gleicher Form in MEQSICO übernommen. Ferner wird in [56] ein Ansatz zur A-Priori-Schätzung der Anzahl der notwendigen Stufen gegeben, welcher in Kap. 6.2 näher betrachtet wird.

## 2.5 Zusammenfassung

In diesem Kapitel wurden die grundlegenden Konzepte des Ansatzes der Elektroquasistatik, der Modellierung und der numerischen Berechnung dargestellt. Hierzu wurde aus den Maxwell'schen Gleichungen die differentielle Gleichung für die Elektroquasistatik hergeleitet.

Ferner wurden Mikrovaristoren als Möglichkeit zur Feldsteuerung mittels nichtlinearer feldstärkeabhängiger Leitfähigkeit vorgestellt. Durch das Material wird in die elektroquasistatische Gleichung ein feldstärke- und damit potentialabhängiger Leitwert $\kappa(\varphi)$ eingebracht.

Um die nichtlineare parabolische partielle Differentialgleichung zu lösen, ist eine Diskretisierung im Raum und in der Zeit erforderlich. Als Methode zur Raumdiskretisierung wird die FEM betrachtet und hieraus die Berechnung der raumdiskretisierten Matrizen für Leitwert und Permittivität abgeleitet.

Abschließend werden Möglichkeiten der Zeitintegration erläutert. Ausgehend von einem expliziten Euler-Verfahren, welches bei „steifen“ Problemen sehr kleine Zeitschrittweiten erfordert um stabil zu bleiben, werden im Gegensatz dazu implizite Zeitintegrationsverfahren vorgestellt. Dem Vorteil großer Zeitschrittweiten steht allerdings die Notwendigkeit eines Verfahrens zur Lösung der nichtlinearen Gleichung gegenüber, wie etwa dem Newton-Raphson-Verfahren. Als Ansatz, um akzeptable Schrittweitengröße ohne die Notwendigkeit eines nichtlinearen Iterationsverfahrens zu erreichen, wird abschließend das RKC-Verfahren als stabilisiertes explizites Zeitintegrationsverfahren dargestellt.

# Kapitel 3

# Beschleunigung numerischer Simulationen durch Grafikprozessoren

Im Zuge der Parallelisierung von Berechnungen sind auch einfache Anwendungscomputer heutzutage mit mehreren Rechenprozessoren ausgestattet. Bereits in den 1980er Jahren wurden mehrere Rechenknoten mittels dem Message Passing Interface (MPI)-Standard zu Rechennetzwerken verknüpft [59]. Hierbei werden die benötigten Daten über die Rechenknoten verteilt oder dupliziert, um den Kommunikationsaufwand zwischen den Knoten zu verringern. Mit dem Aufkommen von Mehrkernprozessoren wurden Parallelisierungsstandards geschaffen, in welchen mehrere Rechenprozesse, die auf mehreren Prozessoren parallel ausgeführt werden, auf einen gemeinsamen Arbeitsspeicher zugreifen. Beispiele hierfür sind OpenMP [60] und Intel Thread Building Blocks (TBB) [61].

Im Zuge einer weiteren Prozessparallelisierung rückten GPUs in den Fokus aktueller Forschung. Durch ihre Aufgabe als Bilddarstellungsinstrument sind GPUs mit einer großen Anzahl eher leistungsschwacher Algorithmical Logical Units (ALUs) ausgestattet. In den Anfängen der Berechnung allgemeiner Probleme auf GPUs wurden Programmiersprachen zur Grafikdarstellung wie OpenGL oder DirectX [62, 63] so verändert genutzt, sodass stattdessen allgemeine Berechnungen durchgeführt wurden [64]. Dies musste innerhalb der Grafik-Programmierschnittstelle stattfinden. Daher mussten allgemeine Probleme in einer Form dargestellt werden, dass diese für die GPU wie eine Rendering-Operation erschienen. Um das Jahr 2000 waren GPUs darauf ausgerichtet, für jeden Pixel eine Farbe zu berechnen [65]. Dies geschah in den sogenannten Pixel Shadern, den oben genannten ALUs. Neben der Position auf dem Bildschirm konnten mit den Pixel Shadern weitere Informationen zu einer Ausgabefarbe berechnet werden. Da diese weiteren Informationen durch den Programmierer kontrolliert werden konnten, war es möglich, den Wert für die Ausgangsfarbe auch als Ergebnis einer allgemeinen Berechnung aufzufassen, welche nichts mit einer tatsächlichen Farbe zu tun hatte. So konnten die Eingaben in Farbwerte transformiert, die Berechnung ausgeführt und die Ausgabefarbwerte zurück konvertiert werden [64].

Dies unterlag jedoch einer Reihe von Restriktionen, welche den praktischen Nutzten dieser Technik limitierten, wie etwa der Notwendigkeit einer zusammenhängenden Datenstruktur, der Unsicherheit bei der Arbeit mit Gleitkommazahlen und der Notwendigkeit in einer „Shading-Language“ zu arbeiten, welche im allgemeinen deutliche Unterschiede zu den klassischen Programmiersprachen wie C, C++ oder Fortran aufweist [66]. Hiervon ausgehend wurden Architekturen und Programmiersprachen entwickelt, welche die Ausführung allgemeinen parallelen Codes auf GPUs ermöglichten. Als erste Grafikkarte dieser Art ist die im November 2006 vorgestellte Nvidia GeForce 8800 GTX zu nennen, welche erstmals Komponenten enthielt, deren einziger Zweck die Lösung von allgemeinen Rechenaufgaben auf GPUs war [64].

Wesentliche Änderungen waren die Einführung einer „unified shader pipeline“, mit welcher alle ALUs von einem einzelnen Programm gesteuert werden konnten und der Fähigkeit, Gleitkomma-Operationen nach Institute of Electrical and Electronics Engineers (IEEE)-Standard durchführen zu können, welche für allgemeine Rechenoperationen unerlässlich ist. Ferner wurden Speicher mit Lese- und Schreibzugriff, ein gemeinsamer globaler Speicher und ein steuerbarer Cache für die einzelnen Multiprozessoren geschaffen [64].

Als Programmiersprachen sind das vom Hersteller Nvidia gestellte, proprietäre CUDA C [67] als auch der offene Standard OpenCL [68] der Khronos

Group zu nennen. Diese Arbeit stützt sich auf CUDA C, weshalb nur Karten des Herstellers Nvidia betrachtet werden. Die Sprache basiert auf dem Sprachstandard C, welcher um einige Schlüsselworte erweitert wurde. Dies ermöglicht die einfache Implementation in Programme, welche auf C oder – wie in dieser Arbeit – C++ beruhen.

Der Bedarf an Rechensystemen zur Berechnung hochparallelisierbarer Probleme, z. B. im Bereich der Künstlichen Intelligenz, hat zu einer schnellen Entwicklung neuer Systeme und rapiden Steigerung der Spezifikations- und Leistungsdaten von General Purpose Graphics Porcessing Units (GPGPUs) geführt. GPGPUs sind GPUs, welche für die Berechnung von allgemeinen, nicht grafisch darstellenden Problemen optimiert sind. So haben sich die Leistungsdaten der auf allgemeine Berechnung optimierten Nvidia Tesla-Serie in der letzten Dekade deutlich gesteigert. Der in 2008 veröffentlichte Nvidia GT200 Chipsatz, war der erste Chipsatz der Nvidia Tesla Serie, welcher zur Berechnung in doppeltgenauer (engl: Double Precision (DP)) Zahlendarstellung fähig war. Er erreichte mit 240 Streamingprozessoren eine DP-Rechenleistung von 78 GFLOPS [69]. Der in 2018 veröffentlichte Chipsatz der Volta-Architektur GV100, welcher in den Gleichnamigen Nvidia Tesla Modellen zum Einsatz kommt kann eine deutlich gesteigerte Leistung vorweisen. Er erreicht eine theoretische DP-Rechenleistung von 7000 GFLOPS und nutzt hierzu 5120 Streamingprozessoren [70].

## 3.1 Aufbau eines Grafikprozessors

Ein auf der CUDA Architektur beruhender Grafikprozessor besteht aus einer Anzahl an Mehrkernrecheneinheiten, Streaming Multiprocessors (SMs) genannt. Ein SM besteht aus bis zu mehreren hundert Rechenkernen (engl.: Cores). Tab. 3.1 stellt die Spezifikationswerte der in dieser Arbeit verwendeten und der im August 2018 leistungsstärksten marktverfügbaren GPGPU, der Nvidia Tesla V100, dar. Die oben genannten Anzahl an Rechenkernen pro SM ergibt sich aus der Division $\frac{Cores}{SM}$.

Im Programmiermodell wird die Parallelität durch Ausführungsstränge, sogenannte „Threads“, gesteuert, analog zur parallelen Programmierung

| Modell | Nvidia Tesla K20m | Nvidia Tesla K80 | Nvidia Tesla V100 |
|---|---|---|---|
| Architektur | Kepler | Kepler | Volta |
| GPU | GK110 | GK210 | GP100 |
| CUDA Cores | 2496 | 2x2880 | 5120 |
| SMs | 15 | 2x15 | 60 |
| DP-Rechenleistung | 1,175 TFLOPS | 2,91 TFLOPS | 7 TFLOPS |
| Takt | 706 MHz | 560 MHz - 875 MHz | 1370 MHz |
| Globaler Speicher | 5 GB | 2x 12GB | 16 GB |
| Speicherbandbreite | 208 GB | 2x 240 GB/s | 900 GB/s |

Tabelle 3.1: Leistungsdaten verschiedener in dieser Arbeit verwendeter (Tesla K20m, K80) und aktueller (Tesla V100) GPUs [71, 72, 70].

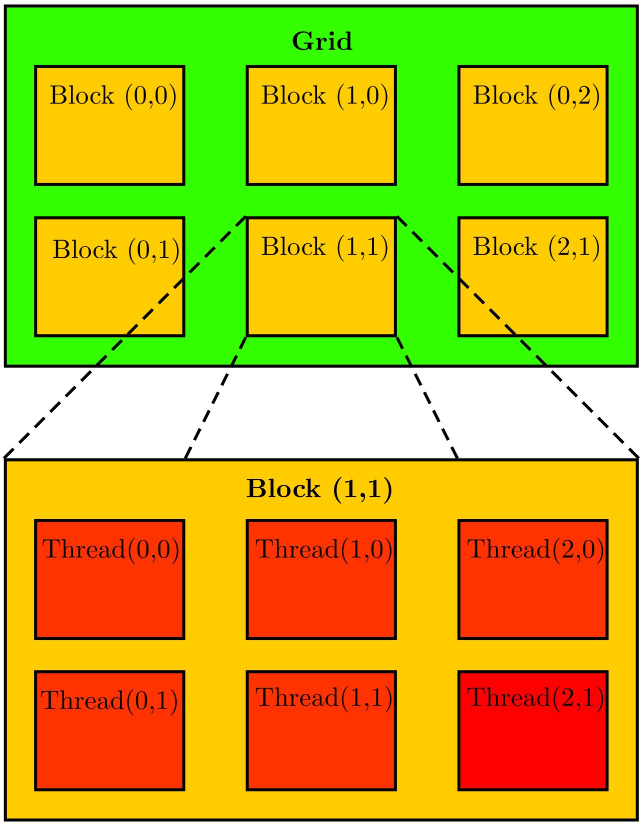


Abbildung 3.1: Logischer Aufbau des CUDA-Programmiermodells unterteilt in Blocks und Threads (eigene Darstellung nach [67]).

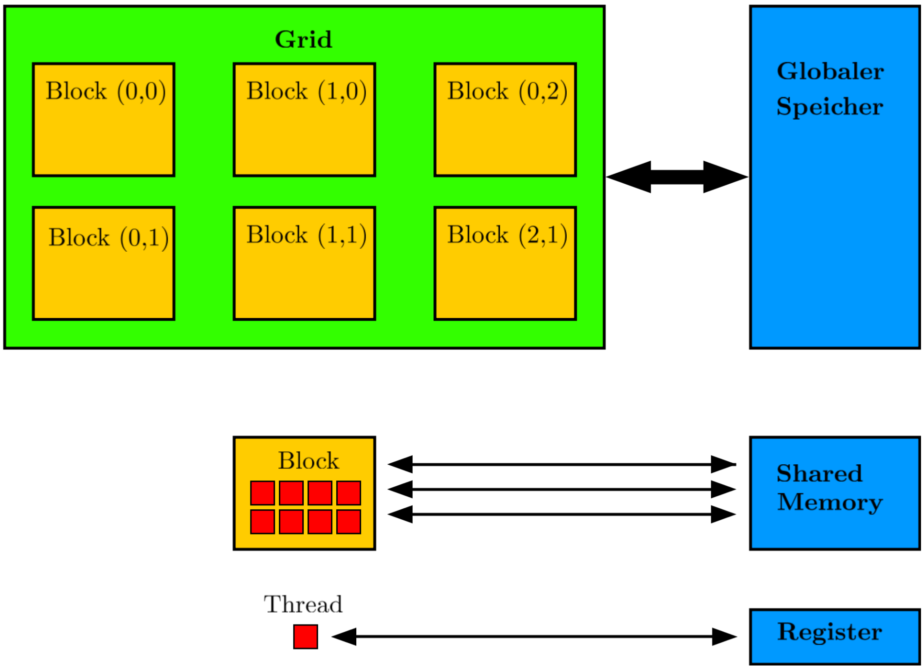


Abbildung 3.2: Logischer Aufbau der Speicherhierarchie eines CUDA-Grafik-Prozessors (eigene Darstellung nach [67]).

für eine Mehrkern-Central Processing Unit (CPU). Threads sind Instanzen, welche parallel zueinander ausgeführt werden können. Eine Anzahl von Threads wird in einem Block zusammengefasst. Beim Aufruf einer Routine wird durch Angabe der Threads und Blocks der Umfang der Parallelität durch den Programmierer vorgegeben. Die einzelnen Blocks werden anschließend in der Ausführung den SMs zugewiesen. Ein Core führt dabei die Anweisungen für einen Thread aus (vgl. Abb. 3.1).

Das Speichermodell lehnt sich an dieses Konstrukt an. Jeder SM verfügt über eine große Anzahl an Registern, welche bei der Programmkompilierung, abhängig vom lokalen Speicherbedarf eines Threads, auf die einzelnen Cores verteilt werden. Auf Ebene der Blocks verfügt jeder Block über einen „Shared Memory“, welcher durch den Programmierer beeinflusst werden kann und auf den die kooperierenden Threads eines Blocks zugreifen können. Weiterhin beinhaltet jeder SM nicht beeinflussbaren L1-Speicher. Diese zeichnen sich durch eine sehr hohe Bandbreite und geringe Latenzzeiten aus. Alle SMs sind an einen globalen Speicher angeschlossen, auf welchen jeder Thread zugreifen kann. Dieser ist üblicherweise wesentlich größer und bietet z. B. für eine Nvidia Tesla K20m 5 GB Speicherplatz. Mit einer theoretischen Übertragungsbandbreite von 200 GB/s bei der Tesla K20 ist dieser jedoch wesentlich langsamer als die oben genannten On-Chip-Memories. Die Beziehung der einzelnen Speicherarten zu Threads, Blocks und Gesamt-GPU sind in Abb. 3.2 dargestellt.

## 3.2 Durchführung allgemeiner Berechnungen mittels eines Grafikprozessor

Der SM verwaltet die Steuerung der parallelen Threads, wie etwa die Ressourcenzuweisung oder die Ausführungsplanung. Er ist darauf optimiert, möglichst keine Verzögerungen im Rechenfluss zu schaffen. Um dies umzusetzen, wird ein Single Instruction Multiple Threads (SIMT)-Modell genutzt [67], welches darauf zielt, die gleiche Anweisung für unterschiedliche Eingabewerte durchzuführen. Hierzu werden die Threads in Gruppen, sogenannte „Warps“, unterteilt [64].

Ein Warp besteht aus 32 Threads, welche Instruktionen parallel ausführen. Hieraus folgt, dass möglichst kein Unterschied bei den zeitgleichen Instruktionen der Threads eines Warp auftreten darf, um eine optimale Nutzung zu ermöglichen. Ein Beispiel für unterschiedliche Instruktionen ist eine if-Abfrage, welche von den einzelnen Threads unterschiedlich beantwortet wird. Dies führt dazu, dass ein Teil der Threads wartet, während ein anderer Teil die Abfrage abarbeitet [67].

Insbesondere bei sehr lokalen Problemen, bei welchen alle notwendigen Daten im On-Chip-Memory und den Registern der Threads vorhanden sind, ergibt sich hieraus der begrenzende Faktor für die Rechengeschwindigkeit. Diese Probleme erhalten ihre Limitierung aus den Latenzen und der Anzahl der Instruktionen, welche zu bewältigen sind. Hier spricht man von sogenannten „instruction-limited kernels“ [64]. Dies ist für die Assemblierung der FEM-Matrizen relevant, welche in Kap. 5.2.2 betrachtet wird.

Dem gegenüber stehen die sogenannten „bandwidth-limited kernels“, also auf der GPU ausgeführte Berechnungsabschnitte, welche in ihrer Geschwindigkeit durch die oben genannte Übertragungsbandbreite limitiert werden. Dies wird dadurch verursacht, dass mit im globalen Speicher gespeicherten Daten nur wenige aufeinander folgende Berechnungen durchgeführt werden. Bei einem Großteil der numerischen Verfahren in der Elektrotechnik trifft diese Problemklasse zu. Beispielhaft seien hier die Finite-Difference-Time-Domain (FDTD)-Methode [14] oder Iterationsverfahren, wie das CG-Verfahren [73], genannt. Letzteres wird in Kap. 4.1 betrachtet.

Um die Bandbreite möglichst voll ausschöpfen zu können, sind bestimmte Zugriffsmuster erforderlich. Ein Zugriff kann „coalesced“, also zusammenhängend, oder „non-coalesced“, nicht zusammenhängend, erfolgen. Grundsätzlich werden Daten aus dem globalen Speicher in „Chunks“ von 128 Byte (Standard) oder 32 Byte (optional) übertragen [67]. Dies sind also deutlich mehr als eine einfachgenaue (engl.: single-percison – 32 Bit bzw. 4 Byte) oder doppeltgenaue (engl.: double-percision – 64 Bit bzw. 8 Byte) Gleitkommazahl. Hieraus werden die durch den Warp nutzbaren Zahlen entnommen und die übrigen Daten verworfen. Beispiel für einen hohen Grad zusammenhängender Daten ist ein Skalarprodukt. Hier werden die zusammenhängenden Werte des ersten und zweiten Vektors geladen und anschließend multipliziert.

Ein vereinfachtes Beispiel: Es soll das Skalarprodukt zweier doppeltgenauer Vektoren $v_1$ und $v_2$ der Dimension $n = 8$ gebildet werden. Hierzu werden die ersten acht Werte von $v_1$ als „Chunk“ geladen. Anschließend werden die ersten acht Werte von $v_2$ geladen. Diese werden nun multipliziert und im „Shared Memory“ addiert. Es wurden also zwei Ladeoperation durchgeführt und die geladenen Zahlen zu 100 % verwendet.

Eine deutlich größere Herausforderung stellen dagegen Matrix-Vektor-Multiplikationen mit dünnbesetzten Matrizen dar. Hier müssen je Thread einzelne Werte aus dem Vektor entnommen werden, welche mitnichten nebeneinanderliegen, da auch die von Null verschiedenen Werte in der Matrix nicht nebeneinander liegen. Dies führt dazu, dass ein großer Teil der aus dem globalen Speicher übertragenen Daten verworfen werden muss und so die effektive Bandbreite reduziert wird.

Als vereinfachtes Beispiel soll hier eine Matrix-Vektor-Multiplikation einer dünnbesetzten Matrix betrachtet werden. Eine Zeile der Matrix hat Nicht-Null-Einträge in den Spalten 1, 12 und 16. Zur Multiplikation werden daher nun aus dem Vektor als erstes die Einträge an Stelle 1 bis 8 geladen. Eintrag 1 wird zur Multiplikation verwendet; die Einträge 2 bis 8 werden ungenutzt verworfen. Das Verhältnis von genutzten zu geladenen Werten (anders ausgedrückt: die Speicherbandbreitenausnutzung) liegt bei $\frac{1}{8} = 12,5\,\%$. Für die zweite Multiplikation werden die Einträge an den Stellen 12 bis 20 geladen. Nun können die Einträge an den Stellen 12 und 16 für eine Multiplikation verwendet werden. Die Speicherbandbreitenausnutzung liegt hier dann bei $\frac{2}{8} = 25\,\%$. In der Summe wurden also nur $18,75\,\%$ der übertragenen Werte verwendet und der Rest ungenutzt verworfen.

Dies und der Umstand, dass Matrix-Vektor-Multiplikationen für dünnbesetzte Matrizen den rechenintensivsten Anteil innerhalb des CG-Verfahrens darstellen, macht ihre effektive Implementierung auf GPUs zum Gegenstand aktueller Forschung (siehe [74, 75, 76]). Kap. 5.1 betrachtet daher vertiefend Optimierungsmöglichkeiten hinsichtlich des Matrixspeicherformats, um eine möglichst hohe Speicherbandbreitenausnutzung zu erreichen.

## 3.3 Zusammenfassung

GPUs sind externe Recheneinheiten, welche an die Hauptplatine eines Servers angeschlossen werden. Diese verfügen über eine hohe Anzahl an Rechenkernen und können so Rechenoperationen massiv-parallelisiert durchführen. Dies wird unter anderem zur Berechnung großer Probleme der linearen Algebra ausgenutzt. Die Programmierung von GPUs kann mittels der Programmiersprachen CUDA C oder OpenCL erfolgen.

Der Aufbau von auf dem CUDA-Konzept basierenden GPUs hinsichtlich ihres Programmier-, Rechen-, und Speicherkonzeptes werden skizziert, sowie die sich daraus ergebenden Möglichkeiten und Beschränkungen abgeleitet. Insbesondere werden „instruction-limited kernels“ und „bandwidth-limited kernels“ eingeführt.

# Kapitel 4

# Iterative Lösung großer dünnbesetzter linearer Gleichungssysteme

Wesentlicher Teil der Lösung elektroquasistatischer Feldsimulationen ist die Lösung von linearen Gleichungssystemen. Dies kann iterativ z.B. mit dem CG-Verfahren geschehen. Zur Reduktion der notwendigen Iterationen werden Vorkonditionierer eingesetzt. Eine Klasse insbesondere für glatte Lösungen effizienter Vorkonditionierer sind die sogenannten Mehrgitterverfahren. Hier haben algebraische Mehrgitterverfahren den Vorteil, dass sie als „Black-Box"-Verfahren ohne Kenntnis der Geometrie verwendet werden können.

## 4.1 Die Methode der konjugierten Gradienten

Ein lineares Gleichungssystem (LGS) mit symmetrisch positiv definiter Matrix kann durch das CG-Verfahren iterativ näherungsweise gelöst werden [34]. Dies wird im Folgenden kurz skizziert. Dieses Thema wird z. B. in [77, 73, 78] tiefer gehend erläutert.

Die Lösung $\mathbf{u}$ des lineare Gleichungssystem $\mathbf{Au} = \mathbf{b}$ ist äquivalent zur Minimierungsaufgabe

$$\mathbf{u} := \min_{\mathbf{v}} \mathbf{F}(\mathbf{v}) \tag{4.1}$$

bezüglich

$$\mathbf{F}(\mathbf{v}) := \frac{1}{2}(\mathbf{v}, \mathbf{Av}) - (\mathbf{b}, \mathbf{v}). \tag{4.2}$$

Hier wird in einer endlichen Anzahl von Iterationsschritten, welche unter Vernachlässigung von Rundungsfehlern aufgrund der Maschinengenauigkeit maximal der Anzahl der Unbekannten $n$ entspricht, die Lösung angenähert.

Ausgehend von einem Anfangswert $\mathbf{u}_0$ und dem korrespondierenden Residuum $\mathbf{r}_0 = \mathbf{b} - \mathbf{A}\,\mathbf{u}_0$ wird ein $K$-dimensionaler Krylov-Unterraum berechnet, der durch $span\{\mathbf{r}_0, \mathbf{Ar}_0, \mathbf{A}^2\mathbf{r}_0, ..., \mathbf{A}^{K-1}\mathbf{r}_0\}$ aufgespannt wird.

Im CG-Verfahren wird die $\mathbf{A}$-orthogonale Basis $span\{\mathbf{q}_1, \mathbf{q}_2, ..., \mathbf{q}_K\}$ aufgespannt, für welche $(\mathbf{q}_i, \mathbf{Aq}_j) = 0$ für $i \neq j$ gilt.

Die einzelnen Vektoren $\mathbf{q}$ werden dabei durch

$$\mathbf{q}_{k+1} = \mathbf{r}_{k+1} + \beta_k \mathbf{q}_k \tag{4.3}$$

bestimmt. $\beta_k$ ergibt sich aus der Orthogonalität von $\mathbf{q}_{k+1}$ bezüglich $\mathbf{q}_k$ und

$$\mathbf{r}_{k+1} = \mathbf{r}_k - \alpha_k\, \mathbf{Aq}_k. \tag{4.4}$$

$\alpha$ stellt das Ergebnis der Minimierungsaufgabe (4.2) dar, welche nach Ableitung $\alpha_k = \frac{\mathbf{r}_k^T \mathbf{r}_k}{\mathbf{q}_k^T \mathbf{Aq}_k}$ lautet. Hieraus lassen sich

$$\mathbf{u}_{k+1} = \mathbf{u}_k + \alpha_k \mathbf{q}_k \tag{4.5}$$

und

$$\mathbf{r}_{k+1} = \mathbf{r}_k - \alpha \mathbf{Aq}_k \tag{4.6}$$

ermitteln [33].

Der Rechenaufwand für einen Iterationsschritt ergibt sich aus einer Matrix-Vektor-Multiplikation, zwei Skalarprodukten und drei skalaren Multiplikationen von Vektoren.

Die Konvergenzrate des CG-Verfahrens hängt von der Konditionszahl

$$\varkappa(\mathbf{A}) = \frac{\lambda_{max}(\mathbf{A})}{\lambda_{min}(\mathbf{A})} \tag{4.7}$$

der Systemmatrix **A** ab, wobei die Konditionszahl $\varkappa(\mathbf{A})$ für die FEM wiederum von der minimalen Kantenlänge abhängt. Die Iterationszahl $K$ zum Erreichen einer Toleranz $\epsilon$ verhält sich mit

$$K \leq log(2/\epsilon)\sqrt{\varkappa(\mathbf{A})} \leq \frac{C}{h_{FEM}} \tag{4.8}$$

umgekehrt proportional zur kleinsten Kantengröße $h_{FEM}$ des FEM-Gitters [34].

Zur Reduktion der Iterationsschritte werden Vorkonditionierer verwendet. Hierzu werden **A** durch $\mathbf{B}^{-1}\mathbf{A}$ und **b** durch $\mathbf{B}^{-1}\mathbf{b}$ ersetzt [33]. Der Vorkonditionierer $\mathbf{B}^{-1}$ sollten die Inverse der Systemmatrix **A** möglichst gut annähern. Dies reduziert die Konditionszahl deutlich [73, 79]. Es gilt

$$\varkappa(\mathbf{B}^{-1}\mathbf{A}) \ll \varkappa(\mathbf{A}). \tag{4.9}$$

Aufgrund des in (4.8) aufgezeigten Zusammenhangs folgt mit (4.9), dass bei deutlich reduzierter Konditionszahl auch die Anzahl der Iterationen $K$ deutlich reduziert wird. Man spricht dann vom vorkonditionierten CG-Verfahren oder Preconditioned Conjugate Gradients (PCG)-Verfahren. Abb. 4.1 zeigt in vergleichender Form die Pseudocodes für das CG-Verfahren mit und ohne Vorkonditionierung.

Einer der einfachsten Vorkonditionierer ist der Jacobi-Vorkonditionierer. Hier wird die Systemmatrix durch ihre Hauptdiagonale approximiert. Dieser Vorkonditionierer besticht vor allem durch seine Einfachheit, da die Inverse der Hauptdiagonalen direkt durch Inversion ihrer Einträge zu erhalten ist. Wenn **D** die Hauptdiagonalmatrix der $n \times n$-Matrix **A** ist, ist $\mathbf{D}^{-1} = \mathrm{diag}\Big(\frac{1}{\mathbf{A}(i,i)}\Big) = \mathbf{B}^{-1}$ mit $i = 1,..,n$. Hier wurden mehrere GPU-Implementierungen vorgestellt [80, 76, 81],[B4].

Weitere Vorkonditionierer sind die Incomplete Choleski (IC)- und Approximate Inverse (AInv)- Vorkonditionierung [73]. Auch diese wurden erfolgreich auf GPUs portiert [82, 83, 84, 85, 86].

```
$\mathbf{r} \leftarrow \mathbf{b} - \mathbf{A}\,\mathbf{u}$

$\mathbf{p} \leftarrow \mathbf{r}$
$rr \leftarrow < \mathbf{r}, \mathbf{r} >$
while (Residuum zu groß) do
    $\mathbf{y} \leftarrow \mathbf{A}\,\mathbf{p}$
    $\alpha \leftarrow rr / < \mathbf{y}, \mathbf{p} >$
    $\mathbf{u} \leftarrow \mathbf{u} + \alpha\,\mathbf{p}$
    $\mathbf{r} \leftarrow \mathbf{r} - \alpha\,\mathbf{y}$

    $rr_{i-1} \leftarrow rr$
    $rr \leftarrow < \mathbf{r}, \mathbf{r} >$
    $\beta \leftarrow rr / rr_{i-1}$
    $\mathbf{p} \leftarrow \mathbf{r} + \beta\,\mathbf{p}$
end while
```

(a) ohne Vorkonditionierer

```
$\mathbf{r} \leftarrow \mathbf{b} - \mathbf{A}\,\mathbf{u}$
$\mathbf{z} \leftarrow \mathbf{B}^{-1}\,\mathbf{r}$
$\mathbf{p} \leftarrow \mathbf{z}$
$rr \leftarrow < \mathbf{r}, \mathbf{z} >$
while (Residuum zu groß) do
    $\mathbf{y} \leftarrow \mathbf{A}\,\mathbf{p}$
    $\alpha \leftarrow rr / < \mathbf{y}, \mathbf{p} >$
    $\mathbf{u} \leftarrow \mathbf{u} + \alpha\,\mathbf{p}$
    $\mathbf{r} \leftarrow \mathbf{r} - \alpha\,\mathbf{y}$
    $\mathbf{z} \leftarrow \mathbf{B}^{-1}\,\mathbf{r}$
    $rr_{i-1} \leftarrow rr$
    $rr \leftarrow < \mathbf{r}, \mathbf{z} >$
    $\beta \leftarrow rr / rr_{i-1}$
    $\mathbf{p} \leftarrow \mathbf{z} + \beta\,\mathbf{p}$
end while
```

(b) mit Vorkonditionierer

Abbildung 4.1: Algorithmen für das CG-Verfahren mit und ohne Vorkonditionierer. Die Änderungen bei der Implementation mit Vorkonditionierer sind in Rot hervorgehoben.

## 4.2 Algebraische Mehrgitterverfahren als Vorkonditionierer für die Methode der konjugierten Gradienten

Die Klasse der Mehrgitter- oder Multigridverfahren stellt mit einer asymptotischen Komplexität von $\mathcal{O}(n)$ [87] die effizientesten Vorkonditionierer dar. Diese unterteilt sich in geometrische Mehrgitterverfahren (GMG)[87] und algebraische Mehrgitterverfahren (AMG) [88] . Beiden Verfahren haben gemein, dass das Problem auf mehreren Ebenen $h \in [1, 2, .., H]$ in immer gröberer Auflösung hierarchisch approximiert wird. Damit haben die Level jeweils eine kleinerer Anzahl an Freiheitsgraden $n_h$ als die hierarchisch tiefer liegenden Level. Der Ablauf einer AMG-Iteration ist in Abb. 4.3 für einen drei Level-Algorithmus schematisch dargestellt.

Für ein Zwei-Level-Verfahren mit den Levels $h$ und $H = h + 1$ lässt sich ein Iterationsschritt $l \rightarrow l + 1$ für verschiedene Behandlung des groben Gitters, schreiben als

$$\mathbf{u}_{l+1} = \mathbf{S}_h\Bigg(\mathbf{S}_h(\mathbf{u}_l) + \mathbf{P}_H^h\,(\mathbf{A}_H)^{-1}\Big(\mathbf{R}_h^H\big(\mathbf{b}_l - \mathbf{A}_h\,\mathbf{S}_h(\mathbf{u}_l)\big)\Big)\Bigg). \tag{4.10}$$

In jedem Level werden kurzwellige Fehlerkomponenten der Approximationslösung mittels eines Relaxationsglätters $\mathbf{S}_h \in \mathbb{R}^{n_h \times n_h}$ reduziert. Während auf einer CPU hierfür Gauß-Seidel-Verfahren [73] oder Symmetric Successive Over-Relaxing (SSOR)-Glätter [89] eingesetzt werden, ist es auf GPUs bei Berechnung von 3D-FEM-Matrizen am effizientesten, Jacobi-Glätter zu nutzen, da diese eine deutlich höhere Parallelität auf Kosten eines geringeren Glättungseffekts haben [90]. Gauß-Seidel-Glätter können über eine sogenannte Rot-Schwarz-Färbung parallelisiert werden. Hier werden sich nicht berührende Knoten schwarz und die restlichen Knoten rot „gefärbt“. In der Lösungsphase werden erst die schwarzen und anschließend die roten Knoten geglättet [87]. Während dies für Matrizen, welche aus einem FD-Gitter resultieren, sehr gut funktioniert, führt die hohe Verbindungsdichte von 3D-FEM-Gittern auf Tetraederbasis zu einen hohen Ungleichgewicht zwischen roten und schwarzen Knoten. Dies gilt insbesondere bei Diskretisierungen zweiter Ordnung, bei welchen aufgrund der noch höheren Anzahl an Verbindungen zwischen den Knoten das Verhältnis von schwarzen zu roten Knoten zwischen 1:26 und 1:29 liegt.

Mittels der Restriktionsmatrix $\mathbf{R}_h^{h+1} \in \mathbb{R}^{n_{h+1} \times n_h}$ werden Vektoren auf die nächste Ebene $h \to h+1$ übertragen. Auf der gröbsten Ebene $H$ kann das System mittels eines direkten Lösers gelöst werden

Anschließend wird das vom höheren Level ermittelte Ergebnis, welches den Fehler des Lösungsvektors darstellt, auf die aktuelle Ebene mittels Prolongationsmatrix $\mathbf{P}_h^{h-1} \in \mathbb{R}^{n_{h-1} \times n_h}$ übertragen und der Lösungsvektor um dieses korrigiert. Im letzten Schritt wird der Lösungsvektor erneut geglättet [87, 88]. (4.10) lässt sich rekursiv auf mehrere Ebenen erweitern.

Dieser Ablauf, die sogenannte „Solve“-Phase, wird in jedem CG-Iterationsschritt als Vorkonditionierer durchgeführt. Sie ist strukturell für GMG- und AMG-Algorithmen identisch. Ihr gegenüber steht die „Setup“-Phase, in welcher der Vorkonditionierer kreiert wird.

```
function AMG($\mathbf{b}$,$\mathbf{u}$,$h$)    ▷ $\mathbf{u}$ = Lösungsvektor, $\mathbf{b}$=rechte Seite , $h$=aktueller AMG-Level
    if $h = H$ then                            ▷ H gibt den maximalen AMG-Level an
        solve($\mathbf{b}_H, \mathbf{u}_H$)    ▷ Löse direkt
    else
        $\mathbf{S}_h \, \mathbf{u}_h$          ▷ Führe Jacobi-Glättung durch
        $\mathbf{r}_h \leftarrow \mathbf{b}_h - \mathbf{A}_h \, \mathbf{u}_h$    ▷ Berechne das Residuum
        $\mathbf{r}_{h+1} \leftarrow \mathbf{R}_h^{h+1} \, \mathbf{r}_h$    ▷ Restriktion
        AMG($\mathbf{r}_{h+1}, \mathbf{e}_{h+1}, h+1$)    ▷ Rekursiver Aufruf
        $\mathbf{u}_h \leftarrow \mathbf{P}_{h+1}^{h} \, \mathbf{e}_{h+1} + \mathbf{u}_h$    ▷ Prolongation und Korrektur
        $\mathbf{S}_h \, \mathbf{u}_h$          ▷ Führe Jacobi-Glättung durch
    end if
end function
```

Abbildung 4.2: Lösungsphase des AMG-Vorkonditionierers als Pseudocode

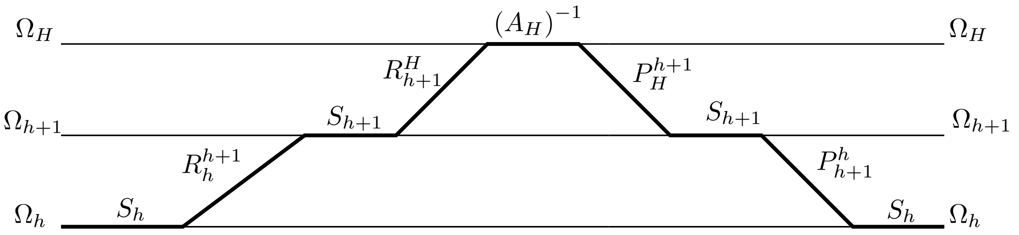


Abbildung 4.3: Lösungsphase des Mehrgitterverfahrens mit drei Levels (eigene Darstellung nach [87]).

Die „Solve“-Phase ist als Pseudocode in Abb. 4.2 dargestellt, was eine sequenzielle Implementation von (4.10) darstellt. Wenn der aktuelle Level $h$ nicht der gröbste Level ist, wird der Lösungsvektor $\mathbf{u}_h$ mittels einer Jacobi-Relaxation durch den Operator $\mathbf{S}_h$ geglättet. Anschließend wird mittels Matrix-Vektor-Multiplikation das Residuum $r_h$ ermittelt und auf den nächsten Level restringiert. Hiermit wird der nächste Level rekursiv aufgerufen. Als Ergebnis erhält man den Fehler $\mathbf{e}_{h+1}$, welcher aus der Beziehung $\mathbf{r}_{h+1} = \mathbf{A}_{h+1}\mathbf{e}_{h+1}$ folgt und in den gröberen Levels ermittelt wurde. Dieser wird mittels Prolongationsmatrix $\mathbf{P}_{h+1}^{h}$ auf den Level $h$ übertragen und korrigiert die Lösung $\mathbf{u}_h$. Kurzwellige Fehler werden anschließend noch einmal mittels Jacobi-Relaxation geglättet.

Auf dem gröbsten Level wird das lineare Gleichungssystem, analog zu (4.10) direkt gelöst. Der Ablauf ist in Abb. 4.3 für einen Drei-Level-

Algorithmus grafisch dargestellt, welcher den Ablauf einer Iteration auf den verschiedenen Ebenen veranschaulicht. Dies macht auch deutlich, warum diese „Solve“-Phase „V-Cycle“ genannt wird – das Bild entspricht einem auf dem Kopf stehenden „V“.

Durch die Glättung kurzwelliger Fehler auf unterschiedlich groben Gittern werden auf dem feinsten Gitter effektiv sowohl kurz- als auch langwellige Fehler geglättet, sodass die Anzahl der Iterationen unabhängig von der geometrischen Auflösung des Problems ist [91].

### 4.2.1 Smoothed-Aggregation Algebraic Multigrid

Geometrische und algebraische Mehrgitterverfahren unterscheiden sich in der Struktur des Aufbaus ihrer Level. Während geometrische Mehrgitterverfahren hierzu Information aus der Struktur und Geometrie des Problems bzw. des Diskretisierungsgitters beziehen, erfolgt dies in algebraischen Mehrgitterverfahren ausschließlich aus der Systemmatrix.

Da in dieser Arbeit algebraische Mehrgitterverfahren verwendet werden, sei für Geometrische Mehrgitterverfahren auf [92, 78] verwiesen. Ansätze zur Implementierung von GMG-Verfahren auf einer und mehreren GPUs sind bei [93] zu finden.

Algebraische Mehrgitterverfahren unterteilen sich ferner in Classical AMG- [88, 94, 91] und SA AMG-Verfahren [95, 96, 91]. Bei Classical AMG-Verfahren werden aus einem gewichteten Beitrag der Knoten des feinen Gitters die Knoten des gröberen Gitters ermittelt. Dabei kann ein Knoten des feinen Gitters auch einen Beitrag zu mehreren Knoten des groben Gitters leisten [91].

Algebraische Mehrgitterverfahren auf Basis von „Smoothed Aggregation“ bilden ihre Level durch Aggregation mehrerer Knoten. Dabei ist ein Feingitter-Knoten immer Teil exakt eines Knotens auf dem groben Gitter [95]. Hieraus werden Prolongationsmatrix und die Systemmatrix des nächsten Levels ermittelt, was im Folgenden weiter erläutert wird. Dieser Vorgang ist für eine Systemmatrix nur einmal im Setup des Vorkonditionierers durchzuführen [91].

In einem ersten Schritt des Setup-Prozesses werden „schwache Verbindungen“ zweier Knoten eliminiert. Durch die Elimination „schwacher Verbindungen“ kann die Anzahl der Verbindungen der FEM-Knoten bereits Initial reduziert werden. Dem zugrunde liegt die Tatsache, dass der Einfluss der über diese „schwache Verbindungen“ verknüpften Knoten aufeinander gering ist [95, 90].

Eine Verbindung $\mathbf{A}_h(i,j)$ gilt als „schwache Verbindungen“, wenn

$$|\mathbf{A}(i,j)_h| < \theta\sqrt{|\mathbf{A}(i,i)\mathbf{A}(j,j)|}, \tag{4.11}$$

gilt. Dabei ist $\mathbf{A}_h(i,j)$ der Eintrag für die Verbindung zweier Knoten $i$,$j$. $\mathbf{A}_h(i,i)$ und $\mathbf{A}(j,j)$ sind die Einträge auf der Hauptdiagonalen sowie der Parameter $\theta \in [0 \leq \theta \leq 1]$ das Stärkemaß [95].

Hierdurch wird aus der Systemmatrix $\mathbf{A}_h$ die Matrix $\mathbf{C}_h \in \mathbb{R}^{n\times n}$ ermittelt, welche nur die starken Verbindungen berücksichtigt und dadurch dünner besetzt ist als $\mathbf{A}_h$. Wenn $\theta = 0$ gesetzt wird, folgt daraus, dass die gesamte Systemmatrix betrachtet wird, also $\mathbf{C}_h = \mathbf{A}_h$ ist [90].

Anschließend werden die aneinandergrenzenden Knoten zu sogenannten Aggregaten von Knoten gruppiert. Dabei besteht ein Aggregat aus einem Wurzelknoten $i$ und allen seinen Nachbarknoten $j$. Aus der Matrix $\mathbf{C}_h$ wird dies ermittelt, indem die Knoten $\mathbf{C}_h(i,j) \neq 0$ des Wurzelknotens $i$ ermittelt werden. Dies ist als Pseudo-Code in Abb. 4.4 dargestellt.

Beispielhaft ist der Ablauf einer Aggregation in Abb. 4.5 dargestellt. Ausgangspunkt ist ein 2D-FEM-Gitter, welches in Abb. 4.5a dargestellt ist. Abb. 4.5b zeigt, wie ausgehend von einem Knoten die Wurzelknoten bestimmt werden. Ein Knoten wird Wurzelknoten, wenn er und seine Nachbarn keine Verbindung zu einem Wurzelknoten haben. Im Beispiel ist die Suchrichtung von links nach rechts und oben nach unten. Im zweiten Schritt werden die Nachbarknoten eines Wurzelknoten seinem Aggregat zugeteilt, was Abb. 4.5c zeigt. Einige, hier schwarz dargestellt Knoten, werden nun keinem Aggregat zugeteilt. In einem letzten Schritt – Abb. 4.5d – werden diese verbleibenden Knoten einem angrenzenden Aggregat, erneut gemäß der oben genannten Suchrichtung zugeteilt und die Aggregate vervollständigt.

```
function AGGREGATION($\mathbf{A}$,$\mathbf{Agg}$)                ▷ Aus $\mathbf{A}$ wird $\mathbf{Agg}$ kreiert
    1. Durchlauf: Identifikation der Wurzelknoten
    for Alle Knoten do
        if Knoten und Nachbarknoten grenzen
            nicht an andere Wurzeln then
            Knoten wird Wurzelknoten
        end if
    end for
    2. Durchlauf: Formen der Aggregate
    for Alle Knoten, außer Wurzelknoten do
        if Knoten grenzt an Wurzelknoten then
            Knoten wird Wurzelknoten zugeteilt
        end if
    end for
    3. Durchlauf: Elimination von Ein-Knoten-Aggregaten
    for Verbleibende Einzelknoten do
        Weise Einzelknoten angrenzendem Aggregat zu
    end for
    Forme Matrix $\mathbf{Agg}$
end function
```

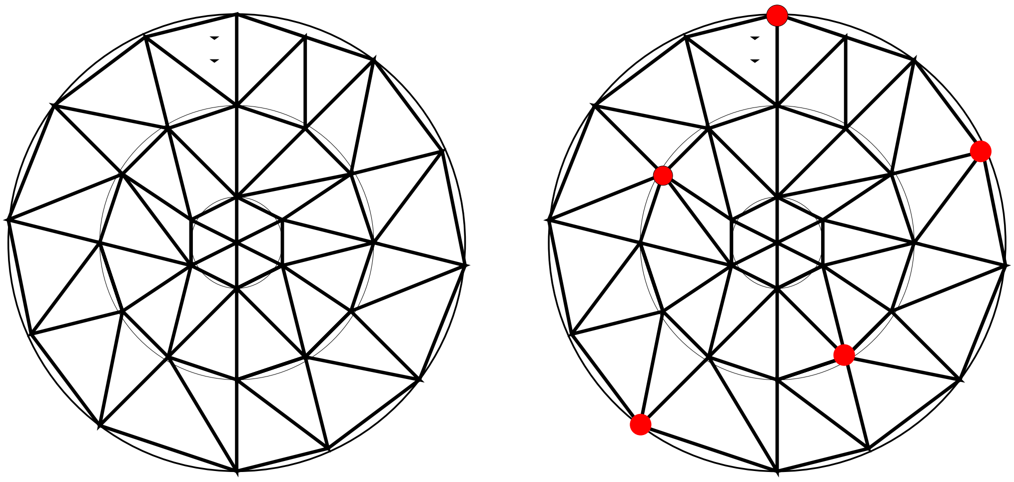
Abbildung 4.4: Algorithmus zur Bildung der Aggregate als Pseudocode.

Die Beziehung der Knoten zu Aggregaten wird in einer Matrix $\mathbf{Agg}_h \in [0,1]^{n_{h+1} \times n_h}$ gespeichert, welche jeder Reihe einen Knoten und jeder Spalte ein Aggregat zuordnet. Ist der Knoten $i$ im Aggregat $j$ enthalten, wird $\mathbf{Agg}_h(i,j) = 1$ gesetzt, ansonsten 0.

Mit einem Kandidaten-Vektor $\mathbf{k}_h \in \mathbb{R}^{n_h}$ wird die ungeglättete Interpolationsmatrix $\mathbf{T}_h^{h+1} \in \mathbb{R}^{n_{h+1} \times n_h}$ ermittelt. Ohne Kenntnis von Kandidatenvektoren für das jeweilige Problem kann ein Einheitsvektor als $\mathbf{k}_{h=1}$ verwendet werden. Bell et. al. [90] empfiehlt die Nutzung von Niedrigenergievektoren, welche über ein Relaxationsverfahren aus einem Zufallsvektor ermittelt werden. Versuche mit den in dieser Arbeit verwendeten Modellen haben jedoch gezeigt, dass sich in diesem Fall die Iterationszahl des linearen Lösers hierbei nicht ändert. $\mathbf{T}_{h+1}^{h}$ ist von der Besetzung identisch mit der Matrix $\mathbf{Agg}_h$. Allerdings werden von $\mathbf{T}_{h+1}^{h}$ die folgenden Eigenschaften gefordert:

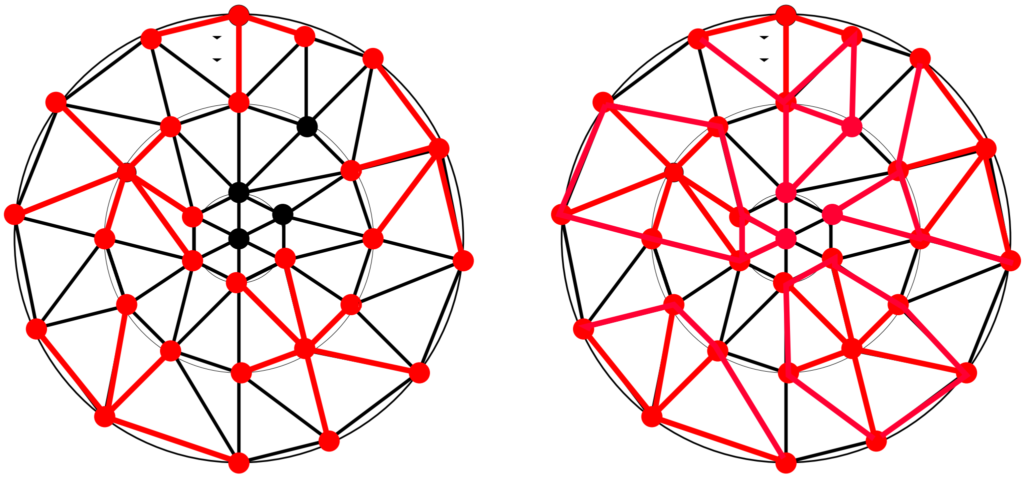
$$\mathbf{k}_h = \mathbf{T}_h^{h+1} k_{h+1} \tag{4.12a}$$

$$\mathbf{T}_h^{h+1^T} \mathbf{T}_h^{h+1} = \mathbf{I} \tag{4.12b}$$



(a) FEM-Gitter

(b) Wurzelknoten bestimmen – vgl. Abb. 4.4 – 1. Durchlauf



(c) Aggregate bilden – vgl. Abb. 4.4 – 2. Durchlauf

(d) Restknoten Aggregaten zuweisen – vgl. Abb. 4.4 – 3. Durchlauf

Abbildung 4.5: Bildung von Aggregaten mit dem SA-AMG-Algorithmus.

Um (4.12) zu erfüllen, werden $\mathbf{k}_{h+1}$ und $\mathbf{T}_{h+1}^{h}$, für ein Beispiel mit $n_h = 5$ und $n_{h+1} = 2$, wie folgt konstruiert:

$$\mathbf{k}_h = \begin{bmatrix} k_1 \\ k_2 \\ k_3 \\ k_4 \\ k_5 \end{bmatrix} \quad , \quad \mathbf{T}_h^{h+1} = \begin{bmatrix} \frac{k_1}{c1} & \\ \frac{k_2}{c1} & \\ & \frac{k_3}{c2} \\ & \frac{k_4}{c2} \\ & \frac{k_5}{c2} \end{bmatrix} \quad , \quad \mathbf{k}_{h+1} = \begin{bmatrix} c_1 \\ c_2 \end{bmatrix}. \tag{4.13}$$

Dabei bilden sich die Skalierungsfaktoren $c_1 = ||[k_1, k_2]||$ und $c_2 = ||[k_3, k_4, k_5]||$ [90].

Abschließend wird der erhaltene ungeglättete Interpolationsoperator $\mathbf{T}_{h+1}^{h}$ geglättet, um in die lokale Interpolation Faktoren längerer Reichweite mit einzubeziehen. So kann mittels eines gewichteten Jacobi-Verfahrens die gesuchte Prolongationsmatrix $\mathbf{P}_{h+1}^{h}$ spaltenweise mit

$$\mathbf{P}_{h+1}^{h}(:,j) = (\mathbf{I} - \omega \mathbf{D}_h^{-1} \mathbf{A}_h) \mathbf{T}_h^{h+1}(:,j) \tag{4.14}$$

ermittelt werden.

Die Restriktionsmatrix $\mathbf{R}_h^{h+1}$ ergibt sich aus der Beziehung

$$\mathbf{R}_h^{h+1} = \mathbf{P}_{h+1}^{h}{}^{T}. \tag{4.15}$$

Mit Kenntnis der $\mathbf{R}_h^{h+1}$ und $\mathbf{P}_{h+1}^{h}$-Operatoren kann die Systemmatrix des nächsten Levels $\mathbf{A}_{h+1}$ über ein Galerkin-Produkt [97] ermittelt werden. Sie ergibt sich mittels einer doppelten Matrix-Matrix-Multiplikation aus

$$\mathbf{A}_{h+1} = \mathbf{R}_h^{h+1} \mathbf{A}_h \mathbf{P}_{h+1}^{h}. \tag{4.16}$$

Durch diese Abfolge können die Level des algebraischen Mehrgitter-Vorkonditionierers rekursiv ermittelt werden. Die Rekursion wird hierbei klassischerweise abgebrochen, wenn eine bestimmte Anzahl an Freiheitsgraden im höchsten Level unterschritten wird oder wenn eine bestimmte Anzahl an Levels erreicht wurde.

### 4.2.2 Bibliotheken

Das Lösen linearer Gleichungssysteme mittels algebraischer Mehrgitterverfahren oder mittels CG-Verfahren unter Verwendung eines AMG-Vorkonditionierers wird heute in mehreren Open-Source oder kommerziell verfügbaren Bibliotheken umgesetzt.

Als eine Bibliothek zum Lösen linearer Gleichungssysteme ist hier das Portable Extensible Toolkit for Scientific Computing (PETSc) [98, 99] zu nennen. Als Multigrid-Vorkonditionierer verwendet PETSc unter anderem „Boomer-AMG“, welcher Teil der Bibliothek High Performance Preconditioners (hypre) ist [100]. „Boomer-AMG“ kann dabei auch unabhängig von PETSc eingesetzt werden. Dies ist eine in MEQSICO implementierte Möglichkeit für einen CPU-basierten AMG-Vorkonditionierer. Vergleiche zwischen PETSc mit Boomer-AMG und GPU-basierten Implementationen wurde in [B5] präsentiert.

Eine weitere große Bibliothek ist das Trilinos-Projekt, welches unter anderem über eine große Anzahl von linearen Gleichungssystemlösern und Vorkonditionierern verfügt. Mit dem Paket „ML“ verfügt Trilinos über einen sehr umfangreichen AMG-Vorkonditionierer [101]. MEQSICO verwendet diesen als bevorzugten Vorkonditionierer im PCG-Verfahren. Daher wird Trilinos ML für aller Referenzberechnungen in dieser Arbeit verwendet. Auf verfügbare Bibliotheken, welche algebraische Mehrgitterverfahren auf GPUs berechnen, wird im Folgenden in Kap. 5.1.2 eingegangen.

## 4.3 Zusammenfassung

Zur Lösung großer dünn besetzter Gleichungssysteme werden iterative Gleichungssystemlöser eingesetzt.

In diesem Kapitel wird das CG-Verfahren eingeführt. Dieses löst ein lineares Gleichungssystem $\mathbf{A}\mathbf{u} = \mathbf{b}$ hinsichtlich $\mathbf{u}$, ausgehend von einem Startwert $\mathbf{u}_0$. Die Matrix $\mathbf{A}$ muss dabei symmetrisch positiv definit sein.

Für sehr große Gleichungssysteme und feine Auflösungen im FEM-Gitter benötigt das Verfahren jedoch eine hohe Anzahl an Iterationen und damit eine sehr lange Rechendauer. Um diese zu reduzieren, werden Vorkonditionierer eingesetzt. Diese nähern die Inverse der Systemmatrix $\mathbf{A}$ an, verbessern die Konditionszahl des modifizierten Gleichungssystems und reduzieren die Iterationszahl. Dabei setzt sich der Aufwand für einen Vorkonditionierer aus dem Aufwand für seine Konstruktion - die „Setup-Phase“ - und dem Aufwand in der CG-Iteration - die „Solve-Phase“ - zusammen.

Als effiziente Vorkonditionierer, insbesondere bei glatten Lösungen, wie sie typischerweise bei elektroquasistatischen Problemen vorliegen, werden Mehrgitterverfahren vorgestellt. Diese lösen ein Problem auf einer gröber werdenden Anzahl an Raumdiskretisierungsgittern und erfassen so auch langwellige Fehler. Algebraische Mehrgitterverfahren generieren die Gitter nur aus der Systemmatrix und benötigen keine Kenntnis über die geometrische Entsprechung des Diskretisierungsgitters. Sie können daher als „Black-Box-Löser“ verwendet werden. Als weitere Verfeinerung wird das algebraische Mehrgitterverfahren basierend auf der sogenannten „Smoothed Aggregation“ vorgestellt und Setup- und Solve-Phase in einzelnen Schritten erläutert.

# Kapitel 5

# Einsatz von Grafikkarten-Beschleunigung in der Feldsimulation

In diesem Kapitel werden Möglichkeiten zur Nutzung von GPUs zur Beschleunigung einer elektroquasistatischen Feldsimulation untersucht. Hierzu wird der LGS-Löser betrachtet und unter Einsatz von GPUs beschleunigt. Es werden entwickelte Implementationen von Matrix-Vektor-Multiplikationen für dünnbesetzte Matrizen präsentiert und im CG-Verfahren und einem AMG-Vorkonditionierer verwendet. Alle zur Lösung des LGS notwendigen Teile werden anschließend mit einer neu entwickelten Bibliothek in eine Umgebung überführt, welche die parallele Berechnung auf mehreren GPUs ermöglicht. Weiterhin wird die FEM-Assemblierung betrachtet und zur Berechung auf GPUs portiert. Neben der Berechnung auf GPUs wird ein Kombination aus Matrix-Vektor-Multiplikation und Assemblierung präsentiert.

## 5.1 Beschleunigung des linearen Gleichungssystemlösers

Der lineare Gleichungssystemlöser stellt bei Ansätzen zur Beschleunigung des Simulationsablaufes die relevanteste Komponente dar. Auf ihn wird in einer Simulation die meiste Zeit verwendet, weshalb eine Beschleunigung hier den größten Effekt auf die gesamte Simulationszeit hat. Dies liegt zum einen an der relativen Dauer pro Iteration, zum anderen aber auch daran, dass er in jedem Schritt des Zeitintegrators ausgeführt werden muss.

In MEQSICO wird das CG-Verfahren genutzt, wie es in Kap. 4.1 vorgestellt wurde. Bei Nutzung eines Vorkonditionierers teilt dieses sich in die Setup-Phase des Vorkonditionierers und die jeweilige Lösungsphase auf. Innerhalb der Lösungsphase berechnet das PCG-Verfahren iterativ Lösungsapproximationen. Dabei besteht jede Iteration aus einer Sparse Matrix Vector Multiplication (SpMV), einigen Vektor-Vektor-Operationen und der Ausführung des Vorkonditionierers.

Im einzelnen Iterationsschritt ohne Vorkonditionierer dominiert die SpMV, auf welche ein deutlich höherer Anteil der Rechenzeit entfällt als auf eine Vektor-Vektor-Operation. Der benötigte Zeitaufwand für den Vorkonditionierer hängt maßgeblich von seiner Art ab. Jedoch werden auch innerhalb der meisten Vorkonditionierer SpMV-Operationen durchgeführt. Ein Beispiel hierfür sind Multigrid-Verfahren, in welchen auf jeder Ebene mindestens fünf SpMVs durchgeführt werden. Im Kap. 5.1.1 sollen daher Speicherformate für dünnbesetzte Matrizen und ihre Eigenschaften bei einer Verwendung auf GPUs betrachtet werden.

Weiterhin zeichnen sich GPUs im Allgemeinen durch einen deutlich kleineren globalen Speicher aus als den, welchen der Hauptspeicher eines Systems zur Verfügung stellt. Dies beschränkt in der Praxis die implementierbare Problemgröße. Im Allgemeinen ist es erforderlich, alle Komponenten für die PCG-Iteration – wie Vektoren, Matrix und Vorkonditionierer – im globalen Speicher der GPU abzulegen. Ein Laden aus dem Hauptspeicher des Hostsystems würde die Operationsausführung auf der GPU durch die hinzukommenden Kommunikationskosten unrentabel machen. Hier sind Möglichkeiten zu finden, den nutzbaren Hauptspeicher zu erweitern. So können auch große Probleme, welche hinsichtlich ihrer Rechendauer beschleunigt werden sollen, effektiv auf Grafikkarten gelöst werden. Eine Möglichkeit ist die Nutzung mehrerer GPUs, welche im Folgenden betrachtet wird.

Es sind mehrere Aspekte zur Beschleunigung der Lösung des linearen Gleichungssystems zu untersuchen:

- Die Möglichkeit der Beschleunigung einer SpMV-Operation durch die Verwendung einer Grafikkartenbeschleunigung.
- Vorkonditionierungsmethoden und ihre Portierbarkeit auf GPUs.
- Weitere Parallelisierung durch die Nutzung mehrerer GPUs.

### 5.1.1 Matrix-Speicherformate und ihr Einfluss auf die Geschwindigkeit der Matrix-Vektor-Multiplikation

Dünn besetzte Matrizen werden in sogenannten Sparse-Matrix-Formaten gespeichert. Diese bieten den Vorteil, dass möglichst nur die von Null verschiedenen Einträge gespeichert werden und so ein deutlich kleiner Speicherbedarf bestehen soll. Das einfachste Speicherformat ist das koordinatenorientierte Coordinate list (COO)-Format [74]. Hier wird ein Wert ungleich Null mit der zugehörigen Spalte und Zeile gespeichert – vgl. Abb. 5.1. Dieses Format ist speicherintensiv ausgelegt; es eignet sich jedoch gut für Matrixmanipulationen, wie das Transponieren einer Matrix, im Vergleich mit anderen Formaten.

$$\begin{pmatrix} 1 & 2 & 0 & 3 \\ 4 & 5 & 0 & 0 \\ 0 & 0 & 6 & 0 \\ 7 & 0 & 0 & 8 \end{pmatrix} \Rightarrow$$

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| Zeile | 1 | 1 | 1 | 2 | 2 | 3 | 4 | 4 |
| Spalte | 1 | 2 | 4 | 1 | 2 | 3 | 1 | 4 |
| Wert | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |

Abbildung 5.1: Matrix-Speicherung im COO-Format.

Das am weitesten verbreitete Format stellt das Compressed Sparse Row (CSR)-Format dar [73]. Hierbei werden in absteigender Zeilenreihenfolge der Wert und die Spalte gespeichert. In einem dritten Array wird für jede Zeile ein Pointer gespeichert, welcher im Wert- und Spalten-Array auf den ersten Wert dieser Zeile zeigt. Dies ist in Abb. 5.2 an einem Beispiel

$$\begin{pmatrix} 1 & 2 & 0 & 3 \\ 4 & 5 & 0 & 0 \\ 0 & 0 & 6 & 0 \\ 7 & 0 & 0 & 8 \end{pmatrix} \Rightarrow$$

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| Zeilenindex | 1 | 4 | 6 | 7 | 9 | | | |
| Spalte | 1 | 2 | 4 | 1 | 2 | 3 | 1 | 4 |
| Wert | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |

Abbildung 5.2: Matrix-Speicherung im CSR-Format.

veranschaulicht. Aufgrund seiner Verbreitung akzeptieren die meisten Bibliotheken zur Berechnung dünnbesetzter Matrizen das CSR-Format als Eingabeformat [101, 98, 100]. Ein Schwerpunkt aktueller Forschung ist daher die Modifikation des Formates, um dieses auf GPU mit maximalem Geschwindigkeitsgewinn verwenden zu können [102, 103, 104, 105, 106, 107].

Die aufwändigste Operation beim iterativen Lösen eines linearen Gleichungssystems ist die Durchführung der SpMV. Bestandteil aktueller Forschung stellt daher die effizienten Implementation der SpMV auf GPUs dar [74, 75, 108]. Hier liegt der Fokus auf dem effizienten Aufruf der Matrix- und Vektoreinträge um die Kommunikation zwischen ALU und globalen Speicher zu minimieren und so eine höhere Beschleunigung zu erreichen.

Das Problem liegt hier insbesondere in den oft nicht nebeneinanderliegenden Matrixeinträgen und den daher zu nutzenden verteilten Einträgen des Vektors. Dies führt erstens dazu, dass mit hoher Wahrscheinlichkeit die Einträge jedes Vektors für jeden Matrixeintrag neu geladen werden müssen. Zweitens werden immer Speicherbereiche von 32 oder 128 Byte geladen und in einem Block genutzt [67], wie bereits in Kap. 3.2 erläutert wurde. Bei im Speicher nebeneinanderliegenden Daten wird dadurch eine höhere Übertragungsdichte und damit Effizienz erreicht. Im Fall einer Matrix-Vektor-Multiplikation müssen jedoch oft größere Teile der geladenen Daten verworfen werden. Dadurch wird eine erhöhte Auslastung der Speicherbandbreite mit nicht benötigten Daten erzeugt. Die Speicherbandbreite stellt üblicherweise den limitierenden Faktor für die Berechnungsgeschwindigkeit einer solchen Operation dar [64]. Dieses Verhalten wird mit der Bezeichnung „Bandwidth-limited Kernel“ beschrieben.

Bei Bandmatrizen, wie sie etwa bei Finite-Differenzen-Verfahren auftreten, ist das Diagonal (DIA)-Format eine gute Alternative. Dieses speichert

$$\begin{pmatrix} 1 & 2 & 0 & 3 \\ 4 & 5 & 0 & 0 \\ 0 & 0 & 6 & 0 \\ 7 & 0 & 0 & 8 \end{pmatrix} \Rightarrow$$

| Diag-Index | -3 | -1 | 0 | 1 | 3 | | | |
|---|---|---|---|---|---|---|---|---|
| Wert | 7 | ⋆ | ⋆ | ⋆ | 4 | 0 | 0 | ⋆ |
| | 1 | 5 | 6 | 8 | ⋆ | 2 | 0 | 0 |
| | ⋆ | ⋆ | ⋆ | 3 | | | | |
| Länge | 4 | | | | | | | |

Abbildung 5.3: Matrix-Speicherung im DIA-Format.

$$\begin{pmatrix} 4 & -1 & 0 & 0 \\ -1 & 4 & -1 & 0 \\ 0 & -1 & 4 & -1 \\ 0 & 0 & -1 & 4 \end{pmatrix} \Rightarrow$$

| Diag-Index | -1 | 0 | 1 | | | | | |
|---|---|---|---|---|---|---|---|---|
| Wert | -1 | -1 | -1 | ⋆ | 4 | 4 | 4 | 4 |
| | ⋆ | -1 | -1 | -1 | | | | |
| Länge | 4 | | | | | | | |

Abbildung 5.4: Matrix-Speicherung im DIA-Format für ein Finite-Differenzen-Problem.

die besetzten Diagonalen einer Matrix mit allen Einträgen inklusive Nullen – vgl. Abb. 5.3. Bei einer SpMV-Operation erfolgt dann eine Sequenz von Vektor-Vektor-Multiplikationen. So werde alle besetzten, und damit gespeicherten Diagonalen, welche auch als Reihe von Vektor verstanden werden können, mit dem Eingabevektor der SpMV-Operation multipliziert und die Ergebnisse aufaddiert. Die speicherdichte Form ermöglicht eine maximale Ausnutzung der GPU.

Die gute Implementierbarkeit und das Potential des Formates werden in [B4] aufgezeigt. Hier wird die Implementierung für Finite-Differenzen in einem LGS-Löser auf einer GPU dargestellt und die Vor- und Nachteile bei der Lösung mittels PCG untersucht. Außerdem wird eine eigene SpMV-Implementation basierend auf dem DIA-Format vorgestellt. Der Vorteil des DIA-Formates für FD-Matrizen wird in Abb. 5.4 ersichtlich.

Da unstrukturierte Gitter nicht aus einer geringen Anzahl an vollbesetzten Diagonalen bestehen, erzeugt das DIA-Format für die in dieser Arbeit betrachteten Problemstellungen jedoch keinen Mehrwert. Es hat einen sehr hohen Speicherbedarf durch die Anzahl an zu speichernden Null-Werten. Außerdem führt es zu vielen unnötigen Rechenoperationen mit eben jenen Nulleinträgen.

Die höchsten Geschwindigkeiten auf GPUs erreichen sogenannte ELLPACK-Formate und insbesondere die auf ihnen basierenden hybriden HYB Formate [74]. Hierbei soll erreicht werden, dass ein Thread oder Thread-Block, je nach Parallelisierungsansatz, eine Zeile der Matrix bearbeitet. Hierzu werden die Nicht-Null-Einträge zeilenweise in Arrays gespeichert. Dabei definiert die Zeile mit den meisten Werten die Länge aller Arrays, welche alle die gleiche Länge haben. Bei kürzeren Zeilen werden am Ende Platzhalter eingefügt. Ein Beispiel ist in Abb. 5.5 dargestellt. Dies ermöglicht eine effektive Datenübertragung aus dem globalen Speicher zu den „Streaming Multiprocessors" der GPU und damit eine hohe Berechnungsperformance.

$$\begin{pmatrix} 1 & 2 & 0 & 3 \\ 4 & 5 & 0 & 0 \\ 0 & 0 & 6 & 0 \\ 7 & 0 & 0 & 8 \end{pmatrix} \Rightarrow$$

| Spalte | 1 | 2 | 4 |
|---|---|---|---|
| | 1 | 2 | ⋆ |
| | 3 | ⋆ | ⋆ |
| | 1 | 4 | ⋆ |

| Wert | 1 | 2 | 3 |
|---|---|---|---|
| | 4 | 5 | ⋆ |
| | 6 | ⋆ | ⋆ |
| | 7 | 8 | ⋆ |

Abbildung 5.5: Matrix-Speicherung im ELLPACK-Format.

Als nachteilig erweisen sich die Platzhalter, wenn eine hohe Varianz bezüglich der Einträge pro Zeile besteht. Insbesondere, wenn durch wenige Zeilen mit vielen Einträgen die Anzahl der notwendigen Einträge für jede Zeile heraufgesetzt wird. Hier setzt das Hybrid (HYB)-Format an, bei welchen es sich um eine Kombination aus ELLPACK-Format und COO-Format handelt. Der Großteil der Werte wird hierbei in einer Matrix im ELLPACK-Format gespeichert, sodass jedoch die Anzahl der Platzhalter limitiert bleibt. Die überzähligen Einträge der wenigen Zeilen, welche sich nachteilig auf das Format auswirken, werden in einer zweiten COO-Matrix gespeichert. Bei einer Matrix-Vektor-Multiplikation wird der Eingangsvektor nun mit beiden Matrizen multipliziert und die Ergebnisvektoren anschließend addiert. Der Vorteil wird beim Vergleich der im ELLPACK-Format gespeicherten Matrix in Abb. 5.5 mit der gleichen Matrix bei Speicherung im HYB-Format deutlich, wie dies in Abb. 5.6 dargestellt ist.

$$\begin{pmatrix}1 & 2 & 0 & 3\\ 4 & 5 & 0 & 0\\ 0 & 0 & 6 & 0\\ 7 & 0 & 0 & 8\end{pmatrix} \Rightarrow$$

a)

| Spalte | 1 | 2 |
|---|---|---|
| | 1 | 2 |
| | 3 | $\star$ |
| | 1 | 4 |

| Wert | 1 | 2 |
|---|---|---|
| | 4 | 5 |
| | 6 | $\star$ |
| | 7 | 8 |

b)

| Zeile | 1 |
|---|---|
| Spalte | 4 |
| Wert | 3 |

Abbildung 5.6: Matrix-Speicherung im HYB-Format. Matrix a) nutzt das ELLPACK-Format; Matrix b) nutzt das COO Format.

### 5.1.2 Vorkonditionierungsmethoden für CG-Verfahren und ihre Umsetzung auf Grafikprozessoren

Die Umsetzung eines CG-Verfahrens unterscheidet sich bei Verwendung von Bibliotheken wie CUBLAS, CUSPARSE [67] oder CUSP [80] nur unwesentlich von einer CPU-basierten Implementation mit der Standartbibliothek der linearen Algebra – Basic Linear Algebra Subprograms (BLAS). Wesentlicher Unterschied ist, dass die Matrix und der Rechte-Seite-Vektor in den globalen Speicher der GPU geladen werden müssen und dass alle Vektoren sowie die Matrix dort Platz finden müssen. Nur so lässt sich eine akzeptable Berechnungsgeschwindigkeit erreichen, ohne die Berechnungsgeschwindigkeit durch die Bandbreite der Hostsystemanbindung zu limitieren, welche mindestens eine Größenordnung unter der des globalen Speichers liegt.

Aufwendiger stellt sich die Implementation des Vorkonditionierers auf einer GPU dar. Grundsätzliche Funktion und prinzipieller Aufbau eines AMG-Vorkonditionierers sind in Kap. 4.2 dargestellt worden. Um einen AMG-Vorkonditionierer effizient auf GPUs zu nutzen, ist es zwingend erforderlich, die Lösungsphase auf die GPU zu portieren. Hierzu wurden verschiedene Ansätze präsentiert, welche sich vor allem auf die Lösungsphase fokussieren [109, 93, 110, 111, 112, 113]. Die Entwickler der CUSP Library [80] stellten einen Ansatz vor, welcher sowohl die Setup- als auch die Lösungsphase auf GPUs ausführte [90]. Wesentliche Ergebnisse sind jedoch, dass der größte Teil der Beschleunigung in der Lösungsphase erreicht wird. Aufgrund der inhärenten Serialisierung des Setup-Vorganges sind hier nur Beschleunigungen eines Faktors 1,5 bis 2 zu erwarten. Dem steht jedoch eine große Speicherintensität im Verlauf des Setups entgegen, welche vor allem durch die Zwischenspeicherung weiterer Matrizen notwendig ist. Es muss sowohl die Matrix $\mathbf{C}_h$, die um die schwächsten Verbindungen reduzierte Systemmatrix, als auch die im Galerkin-Produkt $\mathbf{A}_{h+1} = \mathbf{R}_h^{h+1}\mathbf{A}_h\mathbf{P}_{h+1}^h$ ermittelte Zwischenmatrix $\mathbf{A}_h\mathbf{P}_{h+1}^h$ gespeichert werden. Dies führt bei großen Problemen dazu, dass im Setup aus Speicherplatzgründen auf das Host-basierte Setup ausgewichen werden muss [B5].

In der Solve-Phase ist der Vorteil beim Einsatz von GPUs jedoch wieder deutlich vorhanden. Er wird hierbei vor allem auf den feineren Levels erreicht, wo die massive Parallelität ausgenutzt werden kann. Die CUSP-Implementation wird im Folgenden als Basis verwendet, da sie quelloffen ist und eine gute Integration in eine der schnellsten Bibliotheken für lineare Algebra mit dünn besetzten Matrizen liefert [90]. Hierdurch ist es einfach möglich den Code zu erweitern. Ferner werden die Erweiterungen, welche in [B5] vorgestellt wurden, verwendet, um CUSP mit gemischten CPU/GPU AMG-Implementationen nutzen zu können.

#### 5.1.2.1 Ansätze zur Nutzung eines Mixed-Precision AMG-Vorkonditionierers

Grundsätzlich sind GPUs auf die Berechnung von einfachgenauen Gleitkommazahlen optimiert, da für 3D-Grafik-Anwendungen, für die GPUs ursprünglich entwickelt wurden, keine doppelte Zahlendarstellungsgenauigkeit benötigt wird. Die Genauigkeitsanforderungen an numerische Simulationen machen aber die Nutzung von Gleitkommazahlen in doppeltgenauer Zahlendarstellung zum Standard für diese Anwendungen. Aufgrund der steigenden Bedeutung des wissenschaftlichen Rechnens auf GPUs wurde in den Nvidia GPUs GK110 und GK210 der Kepler Generation, je SM neben den 192 CUDA-Cores – oder Shader-Einheiten – zusätzlich 64 separate ALUs verbaut, welche ausschließlich für Brechungen mit doppeltgenauer Zahlendarstellung zuständig sind [71, 72]. Diese GPUs sind in den in dieser Arbeit verwendeten Grafikkarten vom Modell Tesla K20 bzw. Tesla K80 verbaut. Nichtsdestotrotz bleibt die Leistungfähigkeit bei Berechnungen mit einfachgenauer Zahlendarstellung deutlich höher. Bei einer bandbreitenlimitierten Anwendung ist dies zusätzlich dadurch der Fall, dass nur die halbe Datenmenge vom globalen Speicher zu den Recheneinheiten übertragen werden muss.

Mixed-Precision-Ansätze rechnen daher einzelne, vor allem rechen- und zeitintensive, Teile in einfacher Genauigkeit [93]. In [B6] wurde hierzu ein Ansatz vorgestellt, welcher in MEQSICO durch Modifikation der CUSP-Bibliothek implementiert wurde.

Grundüberlegung ist es hier, dass der zeitintensivste Teil während einer CG-Iteration der AMG-Vorkonditionierer ist. Zusätzlich wird von einem Vorkonditionierer in einer CG-Iteration nur die Approximation der Inversen der Systemmatrix gefordert (vgl. Kap. 4.1). Ein Genauigkeitsverlust hat geringere Auswirkungen als in der umgebenden CG-Schleife. Daher wird in diesem Ansatz ausschließlich der AMG-Vorkonditionierer in einfachgenauer Zahlendarstellung aufgestellt und verwendet.

In der Setup-Phase wird die Systemmatrix $\mathbf{A}$, respektive ihr Vektor der Matrix-Einträge, von doppeltgenauer in einfachgenaue Zahlendarstellung konvertiert. Die Vektoren, welche Angaben zu Zeilen und Spalten enthalten, können für die Matrix mit einfachgenauer Zahlendarstellung mittels View-Funktion [74] – also einem Zeiger für Vektoren – wiederverwendet werden. Mit dieser Matrix wird nun der AMG-Vorkonditionierer erstellt. Bereits hier ist eine Zeitersparnis vorhanden, da die Berechnungen, wie insbesondere das Galerkin-Produkt, in einfachgenauer Zahlendarstellung ausgeführt werden.

Der Mixed-Precision-PCG-Ansatz ist in Abb. 5.7 dargestellt und dem PCG-Ansatz aus Kap. 4.1 gegenübergestellt. Zusätzliche benötigte Variablen sind die Vektoren in einfachgenauer Zahlendarstellung $\mathbf{r}_{SP}$ und $\mathbf{z}_{SP}$. Ferner wird der Vorkonditionierer $\mathbf{B}^{-1}$ durch $\mathbf{B}_{SP}^{-1}$ ersetzt um kenntlich zu machen, dass dieser ebenfalls in einfachgenauer Zahlendarstellung arbeitet.

In der CG-Iteration wird vor Aufrufen des Vorkonditionierers dessen Eingabevektor $\mathbf{r}$, welcher in doppeltgenauer Zahlendarstellung vorhanden ist, in einfachgenaue Zahlendarstellung konvertiert. Anschließend wird die

```
$\mathbf{r} \leftarrow \mathbf{b} - \mathbf{A}\,\mathbf{u}$

$\mathbf{z} \leftarrow \mathbf{B}^{-1}\,\mathbf{r}$

$\mathbf{p} \leftarrow \mathbf{z}$
$rr \leftarrow <\mathbf{r}, \mathbf{z}>$
while (Residuum zu groß) do
    $\mathbf{y} \leftarrow \mathbf{A}\,\mathbf{p}$
    $\alpha \leftarrow rr/<\mathbf{y}, \mathbf{p}>$
    $\mathbf{u} \leftarrow \mathbf{u} + \alpha\,\mathbf{p}$
    $\mathbf{r} \leftarrow \mathbf{r} - \alpha\,\mathbf{y}$

    $\mathbf{z} \leftarrow \mathbf{B}^{-1}\,\mathbf{r}$

    $rr_{ALT} \leftarrow rr$
    $rr \leftarrow <\mathbf{r}, \mathbf{z}>$
    $\beta \leftarrow rr/rr_{ALT}$
    $\mathbf{p} \leftarrow \mathbf{z} + \beta\,\mathbf{p}$
end while
```

(a) PCG-Verfahren

```
$\mathbf{r} \leftarrow \mathbf{b} - \mathbf{A}\,\mathbf{u}$
$\mathbf{r}_{SP} \leftarrow \mathbf{r}$                              ▷ 64-Bit ⇒ 32-Bit
$\mathbf{z}_{SP} \leftarrow \mathbf{B}_{SP}^{-1}\,\mathbf{r}_{SP}$
$\mathbf{z} \leftarrow \mathbf{z}_{SP}$                              ▷ 32-Bit ⇒ 64-Bit
$\mathbf{p} \leftarrow \mathbf{z}$
$rr \leftarrow <\mathbf{r}, \mathbf{z}>$
while (Residuum zu groß) do
    $\mathbf{y} \leftarrow \mathbf{A}\,\mathbf{p}$
    $\alpha \leftarrow rr/<\mathbf{y}, \mathbf{p}>$
    $\mathbf{u} \leftarrow \mathbf{u} + \alpha\,\mathbf{p}$
    $\mathbf{r} \leftarrow \mathbf{r} - \alpha\,\mathbf{y}$
    $\mathbf{r}_{SP} \leftarrow \mathbf{r}$                          ▷ 64-Bit ⇒ 32-Bit
    $\mathbf{z}_{SP} \leftarrow \mathbf{B}_{SP}^{-1}\,\mathbf{r}_{SP}$
    $\mathbf{z} \leftarrow \mathbf{z}_{SP}$                          ▷ 32-Bit ⇒ 64-Bit
    $rr_{ALT} \leftarrow rr$
    $rr \leftarrow <\mathbf{r}, \mathbf{z}>$
    $\beta \leftarrow rr/rr_{ALT}$
    $\mathbf{p} \leftarrow \mathbf{z} + \beta\,\mathbf{p}$
end while
```

(b) Mixed-Precision-PCG-Verfahren

Abbildung 5.7: Algorithmus für das Mixed-Precision-PCG-Verfahren im Vergleich zum in Abb. 4.1 vorgestellten PCG-Verfahren als Pseudocode.

„Solve"-Phase des AMG-Vorkonditionierers in einfachgenauer Zahlendarstellung durchgeführt, wodurch der Geschwindigkeitsgewinn in der CG-Iteration erreicht wird. Da diese Operation bandbreitenlimitiert ist und eine einfachgenaue Gleitkommazahl nur die Hälfte an Speicherplatz benötigt, ist für sie eine Zeitreduktion von bis zu 50 % möglich. Da allerdings auch Integerwerte übertragen werden, deren Transferrate nicht erhöht wird, wirkt sich dies negativ auf die Beschleunigung aus. Nach der Berechnung wird der Ausgabevektor **z** wieder in doppeltgenaue Zahlendarstellung zurückkonvertiert. Die Konvertierung entspricht einem Kopiervorgang mit zusätzlichem Zeitbedarf und wirkt sich entsprechend ebenfalls negativ auf die Beschleunigung aus.

Der größte Nachteil ergibt sich jedoch aus dem zusätzlichen Speicherbedarf der Methode, da der Werte-Vektor der Systemmatrix doppelt gespeichert werden muss. Sowohl in einfachgenauer als auch in doppeltgenauer Zahlendarstellung. Damit ergibt sich hier ein zusätzlicher Speicherbedarf im

Umfang der Hälfte des größten Vektors der Berechnung – des doppeltgenauen Systemmatrixwertevektors. Dieser Speicherplatz muss an einem Stück im Speicher vorhanden sein. Weiterhin müssen zusätzliche Vektoren für die konvertierten Eingangs- und Ausgangswerte vorhanden sein.

### 5.1.3 Die Nutzung mehrerer Grafikkarten zur Lösung des linearen Systems

Der Einsatz von GPUs zur Beschleunigung von EQS-Feldsimulationen resultiert aus der Notwendigkeit, LGS schneller zu lösen. Dies kann jedoch nur gelingen, wenn die Zeit für den Datentransport zwischen Hauptspeicher und GPU-Speicher durch die Beschleunigung in der Rechenzeit überkompensiert wird. Dies ist dann der Fall, wenn die Daten auf der GPU, wie etwa die Systemmatrix, mehrfach genutzt werden. Dies begründet auch, weshalb ein ’swapping’, der Austausch des Matrixteils auf der GPU, nicht effizient ist. Damit ist die Systemgröße durch den üblicherweise deutlich kleineren GPU-Speicher beschränkt.

Im Zuge dieser Arbeit wurde ein Code entwickelt, welcher das CG-Verfahren zu Lösung des linearen Systems auf mehreren GPUs parallel ausführt und in [B7, B8, B9] präsentiert. Ferner ist er in der Lage, einen GPU-basierten AMG-Vorkonditionierer auf GPUs voll parallel einzubinden. Ziele sind es daher, die Speicherbeschränkung zu überwinden und eine zusätzliche Beschleunigung des Lösungsvorganges zu erreichen.

Multi-GPU-Ansätze in GMG-Verfahren wurden in [93] durch Göddeke umfassend untersucht. Eine Multi-GPU Bibliothek für „unsmoothed aggregation“ und „classical AMG“ wurde in [113] von Nvidia präsentiert. Bei beiden Ansätzen erfolgt die Parallelisierung über MPI [59]. Die Streamingprozessoren werden also als einzelne Knoten einer Distributed-Memory-Architektur verwendet. Die in [B7] vorgestellte Implementierung parallelisiert hingegen die Verwendung mehrerer GPU mit OpenMP [60] im Sinne einer Shared-Memory-Architektur.

Der Code basiert auf der CUSP Bibliothek [80], wurde jedoch nicht in diese implementiert, sondern setzt auf dieser auf und macht die wesentlichen

Bestandteile für ein PCG-Verfahren mit AMG-Vorkonditionierer Multi-GPU-fähig. Damit kann an weiteren Entwicklungen der CUSP-Bibliothek partizipiert werden. Dabei setzt der Code auf template-basierte C++-Klassen. Wesentliche Bestandteile und Entwicklungen sind Vektor- und Matrixklassen, Kommunikationsroutinen für einen effizienten Datenaustausch zwischen den GPUs und die bereits genannte CG-Routine mit dem zugehörigen AMG-Vorkonditionierer. Diese Bestandteile werden im Folgenden in den Kap. 5.1.3.1 bis 5.1.3.4 beschrieben.

#### 5.1.3.1 Vektoren und Matrizen

Eine Kopie eines Vektors wird vollständig auf jeder GPU gespeichert. Dies ist vertretbar, da ein Vektor im Vergleich zur Systemmatrix eine deutlich geringere Größe hat und dies Vorteile bei der Implementation einiger Operationen, insbesondere der SpMV hat. Weiterhin verfügt jeder Vektor pro GPU über einen View [74]. Ein „View“ ist einen Zeiger auf Beginn und Ende eines Speicherbereiches, welcher Beginn und Ende des Vektorabschnittes zeigt, für welchen die jeweilige GPU verantwortlich ist. Durch den „View“ ist es nicht notwendig, weiteren Speicher zu allokieren und abzugleichen. Ferner wurden im Zuge dieser Arbeit hiermit Methoden entwickelt, bestimmte lineare Operationen parallel auszuführen.

Eine Matrix benötigt deutlich mehr Speicher als korrespondierende Vektoren. Zusätzlich muss sie in zusammenhängenden Arrays gespeichert werden. Eine Aufteilung auf die einzelnen GPUs ist hier daher vorteilhafter als eine Kopie der Gesamtmatrix auf jeder GPU. Hierzu wird die Systemmatrix zeilenweise aufgeteilt und in Teilmatrizen auf die GPUs verteilt. Zuerst wird dazu die Eingangsmatrix in das CSR-Format konvertiert, das die zeilenweise Aufteilung sofort aus seiner Struktur möglich macht - vgl. Kap. 5.1.1. Anschließend werden die Teile auf die GPUs kopiert und in das Ursprungsformat zurückkonvertiert.

Dies ermöglicht eine sehr schnelle Verteilung ohne rechenintensive Umwandlungen und stellt inhärent eine „load balancing“ sicher, wenn die Matrix-

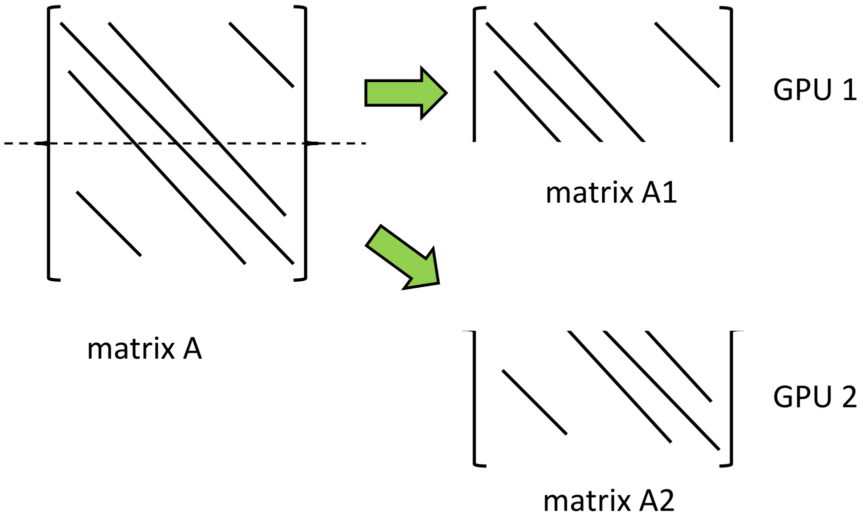


Abbildung 5.8: Schematische Darstellung der Aufteilung einer Matrix auf zwei GPUs in der Multi-GPU-Matrix-Klasse.

struktur näherungsweise homogen ist. Diese Aufteilung ist für eine Matrix mit zwei GPUs in Abb. 5.8 dargestellt.

#### 5.1.3.2 Vektor-Vektor- und Sparse-Matrix-Vektor-Operationen auf mehreren GPUs

Die grundlegenden Operationen, wie Vektor-Vektor- oder SpMV-Operationen, werden auf allen GPUs, unter Nutzung von OpenMP [60], parallel durchgeführt. Dabei entspricht die Anzahl der OpenMP-Threads der Anzahl der GPUs. Jeder OpenMP-Thread steuert eine GPU und führt bei einer Vektor-Vektor-Operation diese nur mit dem ihm durch seinen View zugewiesenen Teilbereich durch. Bei einem Inneren Produkt werden so zum Beispiel erst alle Teile berechnet und die skalaren Ergebnisse der Teile anschließend auf dem Hostrechner addiert. SpMV-Operationen werden ähnlich durchgeführt,

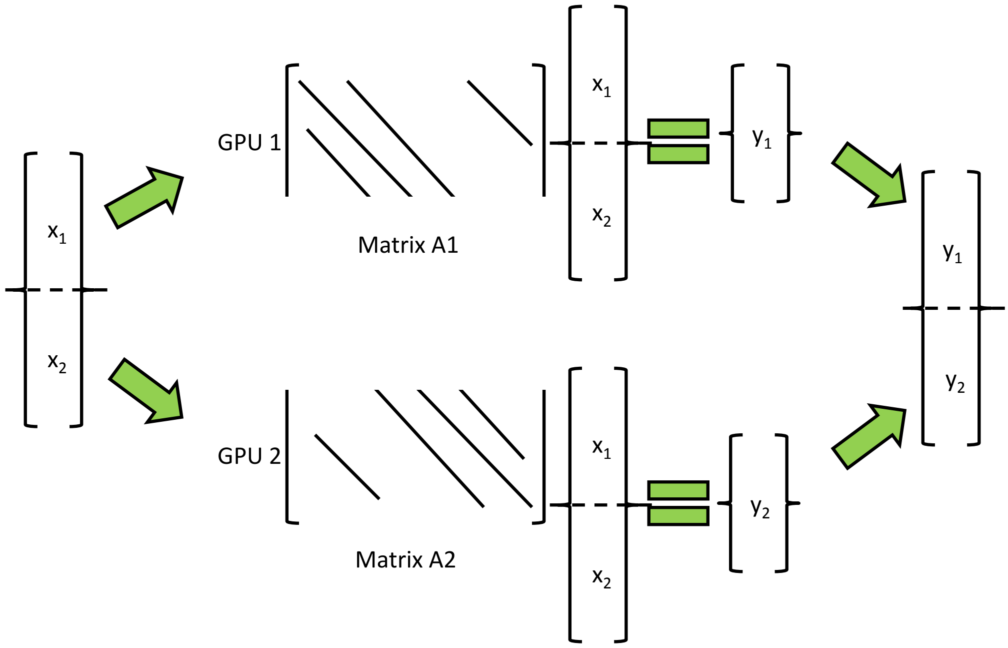


Abbildung 5.9: Durchführung einer SpMV auf zwei GPUs.

allerdings wird hier die auf jeder GPU vorhandene Kopie des Gesamtvektors verwendet. Das Ergebnis ist ein Teilvektor, der dem korrespondierenden „View“ entspricht. Diese werden anschließend durch eine Kommunikationsroutine zum Gesamtvektor zusammengesetzt, wie in Abb. 5.9 veranschaulicht ist.

#### 5.1.3.3 Kommunikationsroutinen

Den kritischsten Teil des Multi-GPU-Codes stellen effiziente Kommunikationsroutinen dar. Dies ist vor allem durch die Speicherbandbreite zu begründen. Die zum Zeitpunkt der Erstellung dieser Arbeit größte Speicherbandbreite bietet die Nvidia Tesla V100 GPU mit einer theoretischen Bandbreites von 900 GB/s [70] (vgl. Kap. 3). Die Geschwindigkeit einer SpMV ist, wie in Kap. 5.1.1 dargestellt, durch diese Bandbreite limitiert, da die ALUs die Operationen schneller ausführen können, als die GPU neue Daten aus dem Globalen Speicher nachliefern kann. Ein Datentransfer zwischen den GPUs ist durch die Bandbreite des Peripheral Component Interconnect Express (PCIe)-Bus beschränkt, welcher auf eine Bandbreites von 8 GB/s limitiert ist [67]. Diesem deutlichen Unterschied in der Bandbreite muss bei Verwendung mehrerer GPUs besonders Rechnung getragen werden. Daher wurden Kommunikationsroutinen entwickelt, welche durch eine weitere Parallelisierung der Kommunikations- und Rechenvorgänge den Kommunikationsvorgang möglichst effektiv gestalten. Hier werden im Folgenden die Routinen „Copy-1n“ zur Verteilung eines Vektors von einer auf mehrere GPUs und „Gather-nn“ zur Synchronisation eines Vektors zwischen mehreren GPUs betrachtet.

**Copy-1n** Die „Copy-1“-Routine kopiert Daten von einer GPU auf $n_{GPU}$ GPUs, mit $n_{GPU}$ der Anzahl aller GPUs des Multi-GPU-Verbundes. Der einfachste Ansatz wäre, die Daten nacheinander auf alle GPUs zu kopieren. Zum Vergleich sei die Referenzzeit $t_{ref}$ die Zeit, welche das Kopieren eines Vektor von einer GPU auf eine Zweite benötigt. Dann würde sich für den einfachen Ansatz für $n_{GPU}$ GPUs ein Zeitbedarf von $(n_{GPU} - 1)t_{ref}$ ergeben. Auch anspruchsvollere parallele Übertragungsmuster würden grundsätzlich dieser Limitierung unterliegen, da alle Daten durch den Hostrecher gesendet werden müssten und dort serialisiert würden. Mit Einführung von CUDA 4.0 [67] wurde jedoch die Routine *DirectAccess* eingeführt. *DirectAccess* ermöglicht, Daten direkt von einer GPU zu einer Anderen zu übertragen, ohne diese durch den Hostrechner zu transferieren. Hierdurch kann der Datentransport parallelisiert werden. Mit der Routine *cudaMemcpyAsync* ist die Übertragung von Daten möglich, ohne dass das Programm mit der Ausführung weiterer Instruktionen auf den Abschluss des Kopiervorgangs wartet. Da GPU-Computing-Karten der Tesla-Serie über zwei Kopier- und eine Recheneinheit verfügen [71, 72], können gleichzeitig Eingangsdaten auf die GPU übertragen, Berechnungen durchgeführt und Ausgangsdaten zurück auf den Hostrechner kopiert werden.

Die dritte Funktion, welche GPUs bieten und welche bei der „Copy-1n“-Routine zum Einsatz kommt, sind *Streams*. Diese bezeichnen die Instruktions-

Pipelines von CUDA. Innerhalb eines Streams werden die Instruktionen hintereinander ausgeführt. Parallele Streams achten jedoch von sich aus nicht auf den Abschluss der Operationen anderer Streams. Hierbei ist zu beachten, dass diese vom Host ausgehen und nicht pro GPU angewendet werden. Es ist daher auch möglich, dass ein Stream mehrere GPUs steuert, was hier genutzt wird. Für eine effiziente Übertragung ist es also erforderlich, dass jede GPU parallel kommuniziert und dass möglichst beide Kopiereinheiten genutzt werden, um die Übertragungsbandbreite zu maximieren.

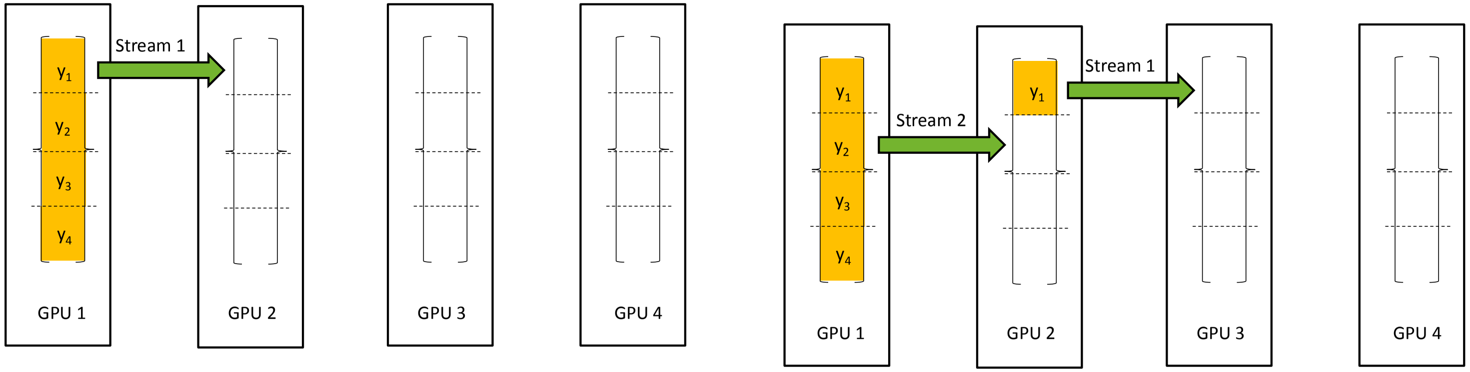
Der entwickelte Kommunikationsablauf ist in Abb. 5.10 dargestellt. Jeder Stream kopiert einen Teil der Daten von der ersten zur zweiten GPU, von der zweiten zur dritten GPU und so weiter mit $(i \mapsto i + 1)$. Hierdurch wird über mehrere Streams erreicht, dass über eine möglichst lange Zeit der maximale Datendurchsatz erreicht wird. Die Zeit maximaler Kopieraktivität kann durch die Verwendung von einer größeren Anzahl an Streams und Datenblöcken als der Anzahl der GPUs noch weiter erhöht werden. Allerdings kann die so erreichte Zeitreduktion auch leicht durch zusätzlich eingebrachte Latenzen überkompensiert werden, was zu einer Verlangsamung statt einer weiteren Beschleunigung führt.

Es ergibt sich bei $n_{GPU}$ GPUs und $s_{GPU}$ Streams die Anzahl der Kopiersequenzen zu $s_{GPU} + (n_{GPU} - 2)$, wobei der Anteil des Vektors, welcher verschoben wird, $\frac{1}{s_{GPU}}$ groß ist. Damit dauert der Kopiervorgang

$$t_{copy-1n} = s_{GPU} + (n_{GPU} - 2))(\frac{1}{s}t_{ref} + t_{latenz}). \tag{5.1}$$

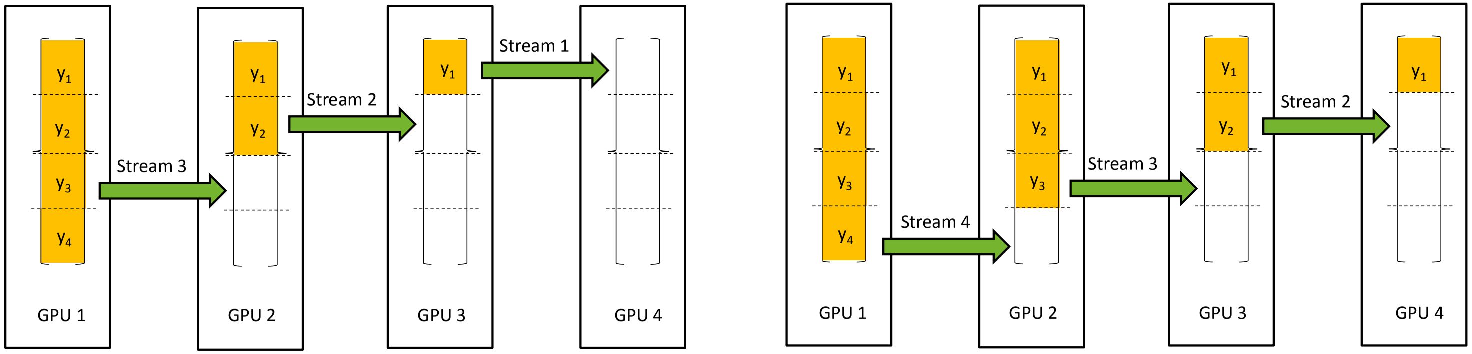
Dies bedeutet, dass die Kopierzeit theoretisch unabhängig von der GPU-Zahl gegen die Zeit strebt, welche eine einfache Kopie von einer GPU zu einer Zweiten benötigt. Eigene praktische Messungen zeigen, dass bei vier GPUs der Typs Nvidia Tesla K20X durch Erhöhung der Stream-Anzahl auf $s_{GPU} = 32$ die Kopierzeit gegenüber der minimal notwendigen Zahl von $s_{GPU} = (n_{GPU} - 1)$ Streams reduziert wird.

**Gather-nn** Die „Gather-n1“-Routine stellt das Sammeln aller Anteile eines Vektors von den verschiedenen GPUs auf einer GPU dar. Da alle Daten auf die Ziel-GPU transferiert werden müssen, ist eine Serialisierung



(a) Zeitschritte t=0 s.

(b) Zeitschritte t=$t_{ref}$.



(c) Zeitschritte t=2 $t_{ref}$.

(d) Zeitschritte t=3 $t_{ref}$.

Abbildung 5.10: Kommunikationsablauf für die Routine „Copy-1n“ mit vier GPUs.

erforderlich und die Übertragungszeit ergibt sich, unter Vernachlässigung des GPU-internen Kopiervorgangs von Beitrag auf Sammelvektor, zu

$$t_{gather-n1} = \frac{n_{GPU} - 1}{n_{GPU}} t_{ref}. \tag{5.2}$$

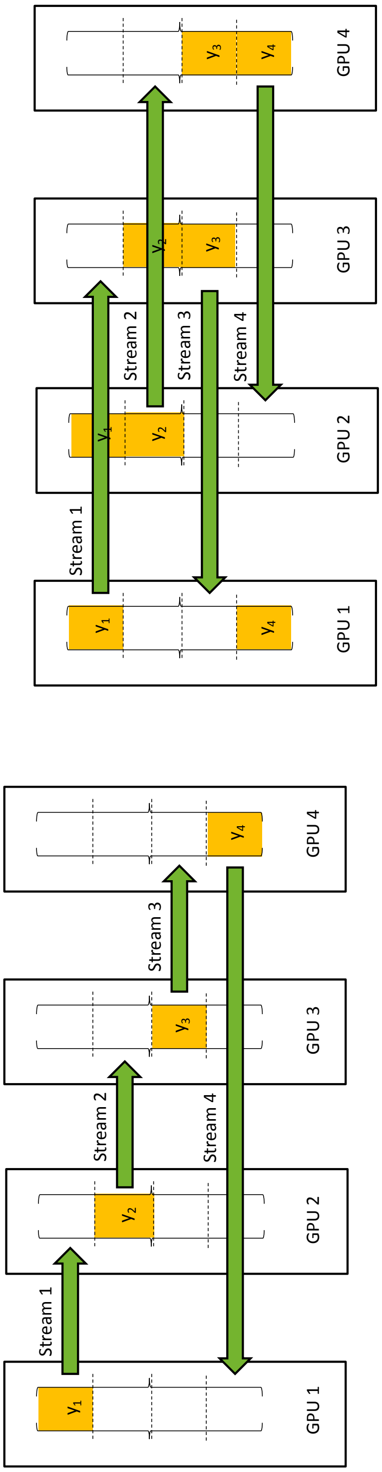
Die „Gather-nn“-Routine basiert auf der „Gather-n1“-Routine, bietet aber bessere Parallelisierbarkeit und wird daher im Folgenden näher in den Fokus genommen. Ausgangspunkt dieser Routine ist, dass z. B. durch eine BLAS-Level-1 Routine oder eine SpMV auf jeder GPU nur der korrespondierende Vektorteil vorhanden ist. Ziel ist es nun, auf allen GPUs den gesamten aktuellen Vektor verfügbar zu machen. Dies stellt in Sequenz eine „Gather-n1“-Routine gefolgt von einer „Copy-1n“-Routine dar. Diese Kombination wird daher aus der Summe von (5.1) und (5.2) mit

$$t_{kombiniert} = ((s_{GPU} + (n_{GPU} - 2)) \frac{1}{s_{GPU}} \cdot t_{ref} + t_{latenz}) + \frac{n_{GPU} - 1}{n_{GPU}} t_{ref} \tag{5.3}$$

abgeschätzt. Für eine ausreichende Anzahl von Streams strebt (5.3) gegen $t_{kombiniert} = 2\, t_{ref}$. Aufgrund von Latenzen gilt in der Praxis jedoch immer $t_{kombiniert} > 2\, t_{ref}$.

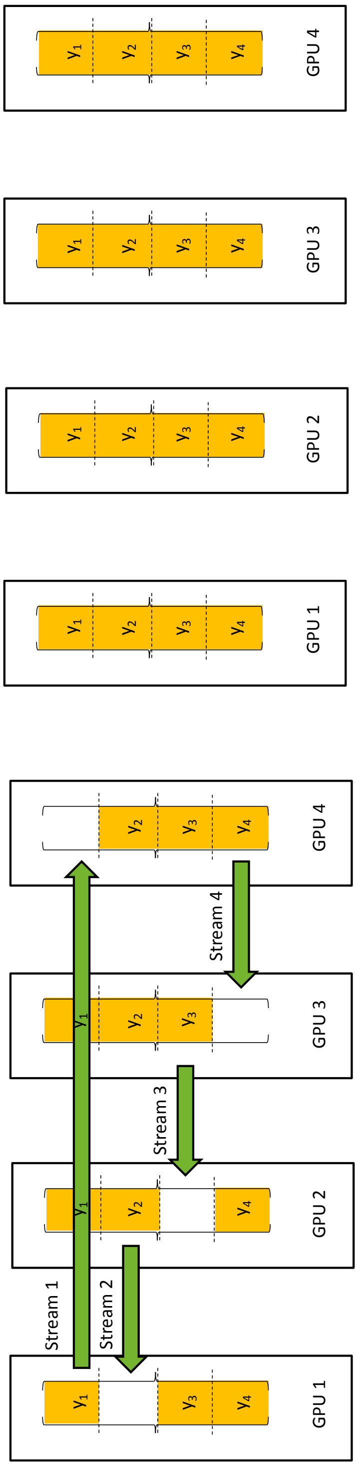
Um diese Serialisierung zu umgehen und die Kommunikationszeit zu reduzieren wurde daher eine separate Routine entwickelt, welche den Datenaustausch weiter parallelisiert. Ziel ist es wieder, beide Kopiereinheiten einer jeden GPU möglichst die gesamte Zeit voll auszulasten. Hierzu wird je Datenstück ein Stream erstellt – also so viele, wie GPUs im Verbund vorhanden sind. Anschließend kopiert jede GPU ihr Datenstück auf die nächste GPU. Die letzte GPU kopiert dabei auf die Erste. Anschließend wechselt der Stream die GPU auf diejenige, welche gerade das Kopierziel war und kopiert den im letzten Schritt auf die GPU kopierten Teil wieder auf die Nächste. Dieser Prozess wird $s_{GPU}$ mal wiederholt, sodass jedes Datenstück auf jede GPU kopiert wurde. Der Vorteil dieser „Kreis-Implementation“ liegt darin, das über die gesamte Dauer jede GPU dauerhaft Daten sendet und empfängt, die Bandbreite also maximiert wird. Der Ablauf ist in Abb. 5.11 dargestellt.

Daraus lassen sich folgende Beobachtungen ableiten: Es ist hier nicht sinnvoll mehr Streams als GPUs zu nutzen, da keine GPU zu einem Zeitpunkt



(a) Zeitschritte t=0 s.

(b) Zeitschritte t=$t_{ref}$.



(c) Zeitschritte t=2 $t_{ref}$.

(d) Zeitschritte t=3 $t_{ref}$.

Abbildung 5.11: Kommunikationsablauf für die Operation „Gather-nn“ mit vier GPUs.

untätig ist und durch mehr Kopiervorgänge nur zusätzliche Latenzen erreicht werden, jedoch keine höhere effektive Datenrate. Hieraus ergibt sich auch, dass die „Gather-nn“-Operation - wie die „Gather-n1“-Operation -

$$t_{gather-nn} = \frac{n_{GPU} - 1}{n_{GPU}} t_{ref} \tag{5.4}$$

benötigt, obwohl das $n_{GPU}$-fache an Daten verschoben wird.

Wenn die Anzahl der Streams gleich einem Vielfachen der Anzahl an GPUs ist, kann in der Theorie weiterhin die maximale Bandbreite erzielt werden. In der Praxis besteht neben den höheren Latenzen jedoch auch die Gefahr, dass nicht alle GPUs gleichzeitig an die folgende GPU senden, weil der eine Kopiervorgang noch andauert und auf der anderen GPU der nächste schon gestartet wurde. Dies führt dazu, dass GPUs warten müssen bis eine Kopiereinheit frei ist, jedoch die falsche Einheit gewählt wird oder etwa eine GPU mit zwei Kopiereinheiten sendet oder empfängt, was durch die Programmierung nicht beeinflussbar ist. Hierdurch kann die geordnete, parallele Übertragung potentiell nicht aufrechterhalten werden und eine Serialisierung ist die Folge. Wenn die Anzahl der Streams kein Vielfaches der Anzahl der GPUs ist, ist die gerade beschriebene Serialisierung garantiert.

#### 5.1.3.4 Der lineare Gleichungssystemlöser

Beim linearen Gleichungssystemlöser handelt es sich um eine Implementierung des PCG-Verfahrens mit den in Kap. 5.1.3.1 beschriebenen Multi-GPU Vektor- und Matrix-Klassen sowie Kommunikationsroutinen an den notwendigen Punkten. Zusätzlich wird eine parallelisierte Version des AMG-Vorkonditionierers eingeführt. Dazu wird der AMG-Vorkonditionierer der CUSP-Library [90] auf dem Hostrechner in der Setup-Phase aufgebaut (vgl. Kap. 4.2.1). Dieser wird dann in seine einzelnen Komponenten zerlegt, wie die Restriktions-, Prolongations- und Systemmatrizen der einzelnen Level. Anschließend werden diese unter dem Schirm einer Multi-GPU AMG-Vorkonditioniererklasse in die entsprechenden Multi-GPU-Datentypen überführt und auf die einzelnen GPUs verteilt.

Der Ablauf der Iterationsschleife inklusive AMG-Vorkonditionierer ist in Abb. 5.12 dargestellt. Dieser stellt eine Umsetzung des in Kap. 4.1 vorgestell-

```
while (Residuum zu groß) do
    gather-nn (p)
    $y \leftarrow \mathbf{A} * \mathbf{p}$
    $\alpha \leftarrow < \mathbf{r}, \mathbf{z} > / < \mathbf{y}, \mathbf{p} >$        ▷ Multi-GPU inneres Produkt
    $\mathbf{u} \leftarrow \mathbf{u} + \alpha * \mathbf{p}$
    $\mathbf{r} \leftarrow \mathbf{r} - \alpha * \mathbf{y}$
    gather-nn-async (r)        ▷ parallel zum AMG
    function AMG(r,z,h=1)        ▷ h gibt den aktuellen AMG-Level an
        if $h = H$ then        ▷ H gibt den maximalen AMG-Level an
            gather-nn ($\mathbf{r}_h$)
            $\mathbf{r}\_host \leftarrow \mathbf{r}_h[GPU0]$
            solve($\mathbf{r}\_host, \mathbf{z}\_host$)        ▷ Löse direkt
            copy-1n ($\mathbf{z}_h \leftarrow \mathbf{z}\_host$)
        else
            presmooth($\mathbf{z}_h$)
            gather-nn ($\mathbf{z}_h$)
            $\mathbf{res}_h \leftarrow \mathbf{r}_h - \mathbf{A}_h * \mathbf{z}_h$
            gather-nn ($\mathbf{res}_h$)
            $\mathbf{r}_{h+1} \leftarrow \mathbf{R}^h_{h+1} * \mathbf{res}_h$
            AMG($\mathbf{r}_{h+1}, \mathbf{z}_{h+1}, h + 1$)
            gather-nn ($\mathbf{z}_{h+1}$)
            $\mathbf{z}_h \leftarrow \mathbf{P}^{h+1}_h * \mathbf{z}_{h+1} + \mathbf{z}_h$
            gather-nn ($\mathbf{z}_h$)
            postsmooth($\mathbf{z}_h$)
        end if
    end function
    $rz_{alt} \leftarrow rz$
    $rz \leftarrow < \mathbf{r}, \mathbf{z} >$
    $\beta \leftarrow rz/rz_{alt}$
    $\mathbf{p} \leftarrow \mathbf{r} + \beta * \mathbf{p}$
end while
```

Abbildung 5.12: Pseudocode der Iterationsschleife des Multi-GPU PCG-Verfahrens mit AMG-Vorkonditionierer. Kommunikationsroutinen sind in rot hervorgehoben.

ten PCG-Algorithmus (Abb. 4.1) mit integriertem AMG-Vorkonditionierer (Abb. 4.2) dar. Dieser ist um die notwendigen Kommunikationsroutinen erweitert, welche in Rot hervorgehoben sind. Die Routine „gather-nn-async“ ist eine Modifikation, welche den Datenaustausch unter Nutzung einer asynchronen Kopierroutine gleichzeitig mit den Berechnungen des Vorkonditionierer ausführt. Für die Ausführung des Vorkonditionierers ist auf jeder GPU nur der zur GPU korrespondierende Datenausschnitt des Vektors **r** notwendig. Er wird nur zur Berechnung des Residuum auf dem feinsten AMG-Level benötigt (vgl. Abb. 5.12 – Zeile 17), was eine BLAS-Level-1 Operation darstellt.

## 5.2 Massiv-parallele Assemblierung von FEM-Systemmatrizen auf GPUs

Ein in sich bereits paralleler Prozess ist die Assemblierung von FEM-Matrizen, welcher sehr geeignet für die Portierung auf GPUs ist. Dieser Ansatz ist daher unter anderem von [114, 115, 116, 117, 118, 119] untersucht worden. Nach der Beschleunigung des linearen Lösers in Kap. 5.1 ist die Assemblierung der FEM-Systemmatrix nun ein Teil während einer elektroquasistatischen FEM-Feldsimulation, dessen Anteil in einem Zeitschritt gestiegen ist und an Relevanz gewinnt, wie in Abb. 5.13 dargestellt ist.

Aufgrund der Nichtlinearität der Leitwertmatrix ist deren Assemblierung in jeder Newton-Iteration erforderlich und daher von wesentlichem Einfluss auf die Rechenzeit. In der Ausgangssituation wird die Assemblierung parallelisiert auf dem Hostrechner ausgeführt. Daher ist diese zusätzlich sehr von der Anzahl der verfügbaren CPU-Kerne abhängig.

In jedem Fall wird diese jedoch stets sehr viel niedriger sein, als die Anzahl der verfügbaren CUDA-Cores auf einer GPU. Weiterhin kann auch auf dem Hostrechner die Parallelität nicht vollständig ausgeschöpft werden, da bei diesem „peinlich parallelen Prozess“ theoretisch jedes Element unabhängig voneinander assembliert werden kann. Allerdings sind die Beiträge anschließend der auszustellenden Systemmatrix hinzuzufügen. Um hier zu

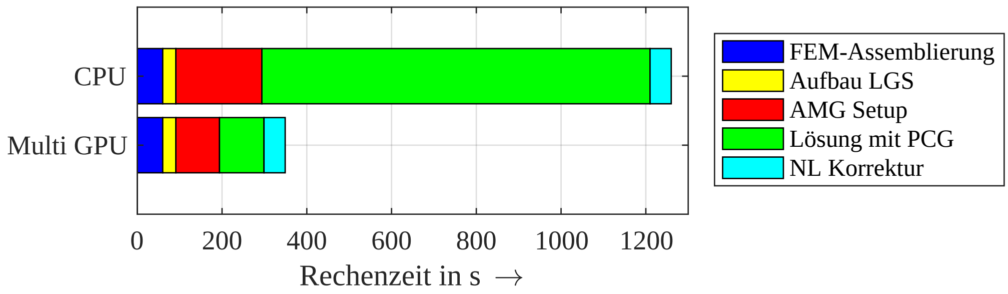


Abbildung 5.13: Zeitverteilung einer Stufe des impliziten SDIRK3(2)-Zeitintegrators für CPU und Multi-GPU LGS-Löser für einen mikrovaristorgesteuerten Langstab-Höchstspannungsisolator (vgl. Kap. 7 - Anwendungsbeispiel Nr. 5 ). Unterteilung in FEM-Assemblierung, Aufbau des LGS, Aufbau des AMG-Vorkonditionierers, Lösung des LGS mittels PCG-Verfahren und Korrektur mittels Newton-Raphons-Verfahren (NL Korrektur).

gewährleisten, dass nicht mehrere Prozesse gleichzeitig in eine Speicherstelle schreiben, ist dieser Vorgang zu serialisieren, was einen deutlichen Einfluss auf die Geschwindigkeit insbesondere mit steigender Anzahl paralleler Prozesse hat. Für die Portierung der Assemblierung ergeben sich nun zwei Möglichkeiten, die im Folgenden beschrieben werden.

Erstens kann die bisherige Matrixassemblierung auf eine GPU portiert und anschließend in die Zielmatrix geschrieben werden. Dies hat zum Ziel, die Assemblierung zu beschleunigen, wobei Eingangs- und Ausgangsvariablen dieselben wie bei der Host-gebundenen Assemblierung bleiben. Der Überbau des Programmes muss hierbei also nicht angetastet werden.

Zweitens ist es aufgrund der massiven Parallelität dieses peinlich-parallelen Prozesses und der Tatsache, dass SpMV bandbreitenlimitierte und nicht instruktionslimitierte Operationen sind, erfolgversprechend, auf sogenannte matrixfreie Implementierung zu setzen. Hierbei wird das einzelne Element assembliert und die lokale Matrix sofort mit dem Eingangsvektor multipliziert. Hierdurch werden Eingangs- und Ausgangsdatenmenge deutlich reduziert [120].

Dieser Ansatz wird im Folgenden insbesondere für die expliziten Zeitintegrationsverfahren relevant, in welchen die oft zu assemblierende Leitwertmatrix nur einen Beitrag zur rechten Seite des linearen Gleichungssystems liefert (vgl. Kap 2.4.2).

### 5.2.1 Beschleunigte Matrix-Assemblierung auf GPUs

In der Assemblierung wird für jedes Element eine lokale Matrix ermittelt, welche anschließend in die Systemmatrix eingefügt wird. Bei der Portierung auf GPUs wird der Ansatz verfolgt, dass jeder Thread ein Element assembliert, um maximale Parallelität zu erreichen. Bisherige Veröffentlichungen setzen darauf, ein Element pro Threadblock zu assemblieren und im gemeinsamen „Shared-Memory“ aufzubauen [121, 122]. GPUs der Tesla-Serie mit Kepler-Architektur können pro Thread jedoch 255 Register nutzen. Hierdurch ist es möglich, auch in doppeltgenauer Zahlendarstellung, welche zwei Register pro Zahl benötigt [67], effiziente Implementierungen mit weiter gesteigerter Parallelität zu erreichen und so das Verhältnis von Berechnungen zu Speichertransaktionen weiter zu erhöhen.

Dabei folgt die Implementation [B1] weitgehend einem klassischen Muster: Zuerst werden die benötigten Daten, wie Integrationspunkte, geometrische Daten und Matrixposition, aus dem globalen GPU-Speicher in die Register des einzelnen Threads geladen. Anschließend wird die lokale Elementenmatrix assembliert und jeder Eintrag als Wert mit zugehöriger Spalte und Zeile in den globalen Speicher geschrieben.

Durch die elementweise Assemblierung sind nun für eine Vielzahl von Matrixeinträgen mehrere Einträge vorhanden, welche traditionell aufaddiert werden. Aufgrund der höheren Effizienz von Sortieralgorithmen auf einer GPU wird dies wie folgt umgesetzt: Die Einträge sind als drei Arrays für Werte, Zeilen und Spalten gespeichert - vgl. Kap. 5.1.1, was für GPUs eine höhere Effizienz ermöglicht als ein Array bestehend aus Kombinationen aus Wert, Zeile und Spalte, wie es bei CPU basierten Verfahren üblich ist [67, 64]. Nun werden diese Arrays hinsichtlich Spalte und Zeile sortiert und anschließend reduziert, sodass nebeneinanderliegende Werte mit gleicher Spalte und Zeile zu einem Wert reduziert werden. Anschließend wird das Ergebnis in die Ausgabematrix geschrieben. Die Implementierung nutzt dabei die Symmetrie der lokalen Matrix aus. Bei der Assemblierung eines Tetraeders zweiter Ordnung hat die lokale Matrix eine Dimension von $10 \times 10$ [40]. Es sind pro Thread also $n_L = 100$ Werte in doppeltgenauer Zahlendarstellung zu speichern, welche jeweils zwei Register belegen. Dies kann durch Ausnutzung der Symmetrie der Matrix auf $n_L = 55$ Werte reduziert werden, wenn nur die Hauptdiagonale und die obere Dreiecksmatrix der lokalen Elementenmatrix gespeichert werden. Wird diese nun exportiert, reduziert sich die Anzahl der zu sortierenden Elemente analog. Die Erschaffung der vollständigen Ausgabematrix wird dann erst abschließend durchgeführt, indem Einträge, welche nicht der Hauptdiagonalen entsprechen, zusätzlich mit umgekehrten Spalten- und Zeilenindizes hinzugefügt werden.

Die Anzahl der sich aus der Assemblierung ergebenden Einträge ist wesentlich höher als die Anzahl der Einträge der Ausgabematrix, da die Knoten und Kanten Teil mehrerer Volumengitterelemente sind. Dies hat zur Folge, dass bereits Probleme mittlerer Größe zwischen $n = 10^5$ und $n = 10^6$ Freiheitsgraden mehr Speicher benötigen, als auf zurzeit existierenden GPUs verfügbar ist. Daher muss die Assemblierung in Teilschritten vorgenommen werden. Die Größe der Teilstücke wird dabei über den freien Speicher der GPU bestimmt. Dieser wird ermittelt und um einen Sicherheitsfaktor $\theta \in [0 < \theta \leq 1]$ erweitert. Da jeder Eintrag aus zwei Integer-Werten zu 32 Bit und einem Wert vom Typ Double zu 64 Bit besteht, kann der Speicherbedarf $S$ in Byte mit der Anzahl der Elemente $n_E$ und der Größe der, wie oben beschrieben, reduzierten lokalen Matrix eines Elements $n_L$ zu $S = 16 \cdot n_L \cdot n_E \cdot \theta$ ermittelt werden. Bei eigenen Versuchen hat sich für $\theta$ ein Faktor von $(0,5 < \theta \leq 0,7)$ bewährt.

### 5.2.2 Kombination von Assemblierung und Sparse-Matrix-Vektor-Multiplikation

Aufgrund des großen Einflusses auf die Rechenzeit, welchen bei einer GPU-basierten Assemblierung die Reduktion der elementweisen Ergebnisse und das Erstellen der Matrix einnehmen, ist es naheliegend, die Assemblierung sofort mit einer Matrix-Vektor-Multiplikation zu verbinden [120, 123, 124, 125].

Dies ist insbesondere bei nichtlinearen Materialien vorteilhaft, da hier die Assemblierung sehr häufig neu vorgenommen werden muss [B1]. Durch die direkte Kombination der Operationen werden folgende Vorteile erreicht:

- Die Menge der Ausgabedaten je GPU-Thread sinkt deutlich.
- Es ist keine Sortierung der Daten erforderlich.
- Das zeitintensive Aufstellen der Ausgabematrix entfällt.
- Der Speicherbedarf wird deutlich reduziert.

Neben den Daten, welche bei einer GPU-basierten Matrix-Assemblierung benötigt werden, wie sie in Kap. 5.2.1 beschrieben ist, sind zusätzlich ein Eingabe- und ein Ausgabevektor zu übergeben. Hiermit können die einzelnen Einträge der Elementenmatrix nun ermittelt und sofort mit dem Eingabevektor multipliziert werden. Anschließend wird der Eintrag der Elementenmatrix verworfen. Dies führt bereits im Assemblierungskernel zu einer deutlich geringeren Registernutzung, wodurch die Effizienz weiter deutlich gesteigert werden kann. Durch weitere Optimierungen, wie die Zwischenspeicherung der Integrationspunkte im Shared Memory, die Nutzung von „*const _restrict_*“ Zeigern, welche ab CUDA-Architekturmodell 3.5 ein aggressives L1-Caching ermöglichen, dem Entrollen aller Schleifen und der Optimierung der Befehlsfolge, um einen möglichst großen Instruction Level Parallism (ILP) zu schaffen [67], kann der Ausgangskernel noch einmal um einen Faktor Zwei bis Fünf beschleunigt werden [B1].

Die Anzahl der Multiplikationen in diesem Ansatz übersteigt die Anzahl der Multiplikationen bei einer normalen SpMV dabei um den Faktor 250 bei Elementen zweiter Ordnung, was den deutlich höheren Rechenaufwand veranschaulicht. Es ist dennoch möglich, diesen Ansatz soweit zu optimieren, dass die Zeit zur Durchführung der Operation in derselben Größenordnung wie eine klassische SpMV möglich ist. Dies ist vor allem dadurch zu begründen, dass Operationen innerhalb eines Threads sehr viel schneller durchgeführt werden können, als eine Speicheranfrage in den globalen Speicher der GPU. Daher dominieren diese Anfragen und das sichere Zurückschreiben der Ergebnisse in den globalen Speicher den hier vorgestellten kombinierten Assemblierungs- und SpMV-Kernel. Für das sichere Zurückschreiben sind Schranken einzubauen, um den Zugriff mehrerer Kerne auf dieselbe Speicherstelle zu verhindern. Dies führt zu einer gewissen Serialisierung und reduziert die Geschwindigkeit bei der Datenausgabe.

## 5.3 Zusammenfassung

Es werden Möglichkeiten diskutiert, elektroquasistatische 3D FEM-Feldsimulationen durch die Verwendung von GPUs zu beschleunigen. Dabei werden die rechenzeitdominierenden Teile – die Assemblierung der Leitwert-Matrix und die Lösung des resultierenden linearen Gleichungssystems – betrachtet.

Es werden in [B4] entwickelte SpMV-Operationen für GPUs und ihre Einbettung in GPU-beschleunigte CG-Verfahren diskutiert. Als Vorkonditionierer wird ein GPU-beschleunigter AMG-Vorkonditionierer nach [B5] vorgestellt. Ein Mixed-Precision-Ansatz zur Reduzierung des Flaschenhalses der GPU-Speicherbandbreite wird analog zu [B6] entwickelt.

Davon ausgehend wird eine Bibliothek entwickelt [B7, B9, B8], welche das gesamte PCG-Verfahren inklusive AMG-Vorkonditionierer zur Berechnung auf mehreren GPUs parallelisiert. So wird der limitierende Faktor Speicherplatz umgangen und die Berechnung weiter beschleunigt. Für das Multi-GPU beschleunigte PCG-Verfahren werden Vektor- und Matrix-Klassen sowie Kommunikationsroutinen entwickelt.

Als weitere Option einer GPU-Beschleunigung wird die FEM-Matrix-Assemblierung entwickelt [B1]. Die vorgestellte Implementation ist auf die Architektur moderner GPGPUs optimiert. Dadurch wird die Assemblierung der einzelnen Tetraeder massiv beschleunigt, wohingegen die Bildung der Gesamtmatrix durch serialisierte Addition zeitlich deutlich aufwendiger bleibt. Um die Bildung einer globalen Matrix zu umgehen, wird eine Kombination aus gleichzeitiger SpMV und Assemblierung vorgestellt. Bei dieser findet die Multiplikation sofort auf den lokalen Elementenmatrizen statt. Die Ausgabedatenmenge und ihre Komplexität werden reduziert, wodurch eine deutliche Beschleunigung erreicht wird.

# Kapitel 6

# Beschleunigung der EQS-Simulation durch Adaption der Zeitintegrationsverfahren

In diesem Kapitel werden Ansätze zur Beschleunigung von elektroquasistatischen Feldsimulationen betrachtet. Ausgangspunkt sind die in Kap. 5 präsentierten Ansätze zur Beschleunigung der Simulation mit GPUs. Hierdurch verschiebt sich die relative Gewichtung der Rechenzeit der einzelnen Teile eines Zeitschrittes und macht potentiell die Betrachtung vorher nicht rentabler Ansätze interessant. So wird für das bisher Verwendete implizite Zeitintegrationsverfahren SDIRK3(2) ein Ansatz zur Reduktion der Zeit zu wiederholten Aufstellen des AMG-Vorkoditionierers präsentiert. Als vorher nicht rentabler Ansatz wird das explizite RKC-Verfahren untersucht. Hier werden Ansätze zur adaptiven Stufen- und Zeitschrittwahl präsentiert. Ferner wird die, im Vergleich zu impliziten Verfahren, andere Struktur eines expliziten Verfahrens ausgenutzt. Durch Startwertgenerierung für den LGS-Löser wird eine weitere Beschleunigung erreicht. Hier werden neue Ansätze entwickelt, welche auf der „Subspace Projection Extrapolation“-Methode und auf Modellordnungsreduktion basieren.

## 6.1 Ansätze zur Beschleunigung der EQS-Simulation bei Nutzung von impliziten Zeitintegrationsverfahren

Das SDIRK3(2)-Verfahren wurde in Kap. 2.4.1 bereits als implizites Zeitintegrationsverfahren vorgestellt. Es handelt sich um ein SDIRK-Verfahren, das je Zeitschritt in vier Stufen eine Lösung zweiter und eine Lösung dritter Ordnung erzeugt, welche für einen Fehlerschätzer genutzt werden können (vgl. Kap. 2.4.1). Bei nichtlinearen Problemen, wie sie in dieser Arbeit betrachtet werden, macht dies die Lösung eines nichtlinearen Gleichungssystems, z. B. durch eine sukzessive Linearisierung mit dem Newton-Raphson-Verfahren erforderlich.

In Kap. 2.4.1 ist mit (2.32) der Aufbau einer mehrstufigen impliziten Zeitintegration mittels Butcher-Schema gegeben. Es wird von der implizit zeitdiskretisierten, elektroquasistatischen Gleichung für einen Stufe $j \rightarrow j+1$ im Zeitschritt $t_i \rightarrow t_{i+1}$ ausgegangen:

$$\left(\frac{1}{a_d\,\Delta t}\mathbf{M}_\varepsilon + \mathbf{K}_\kappa(\mathbf{u}_{j+1})\right)\mathbf{u}_{j+1} = \frac{1}{a_d\,\Delta t}\mathbf{M}_\varepsilon\,\mathbf{u}_j + \mathbf{b}_{j+1}. \tag{6.1}$$

Dabei verwendet (6.1) den Hauptdiagonaleneintrag $a_d$ des Butcher-Tableaus zur Stufe $j$ (welcher für alle Stufen konstant ist), die Zeitschrittweite $\Delta t$, die Permittivitätsmatrix $\mathbf{M}_\varepsilon$, die Leitwertmatrix $\mathbf{K}_\kappa(\mathbf{u})$, die Rechte Seite $\mathbf{b}$ und die Potentialverteilung $\mathbf{u}$, deren Index $j$ die betrachtete Stufe im Zeitschritt $i$ angibt. Einsetzten von

$$\mathbf{f}(t_i + c_j\Delta t, \mathbf{u}_{j+1}) \approx \frac{\mathbf{u}_{j+1} - \mathbf{u}_j}{a_d\,\Delta t}$$

in (2.21) führt zu (6.1). $\mathbf{u}_j$ ermittelt sich aus der Linearkombination der gewichteten Ergebnisvektoren der vorherigen Stufen zu

$$\mathbf{u}_j = \mathbf{u}_i + \Delta t \sum_{l=1}^{j-1} a_{j,l}\,\mathbf{u}'_l, \quad j = 1, ..., 4, \tag{6.2}$$

```
function SDIRK-Stufe($\mathbf{u}_j$,$\mathbf{u}_{j+1}$,$j$,$t$,$\Delta t$)    ▷ Aus $\mathbf{u}_j$ wird $\mathbf{u}_{j+1}$ errechnet
    Assemblieren der nichtlinearen Matrix $\mathbf{K}_\kappa(\mathbf{u}_j)$
    Aufstellen der rechten Seite $\mathbf{rhs} = \mathbf{b}_{j+1} + \frac{1}{a_d\,\Delta t}\mathbf{M}_\varepsilon\,\mathbf{u}_j$:
    Aufstellen der Systemmatrix $\mathbf{A} = \left(\frac{1}{a_d\,\Delta t}\mathbf{M}_\varepsilon + \mathbf{K}_\kappa(\mathbf{u}_j)\right)$
    Setze $k = 0$, $r_{NL} = 1$    ▷ $k$: Newton-Iteration; $r_{NL}$: NL Residuum
    while $r_{NL} > tol_{NL}$ do
        Aufstellen des AMG-Vorkonditionierers
        Lösen von $\mathbf{u}_{j+1,k+1} = \mathbf{A}^{-1}\,\mathbf{rhs}$ mit AMG-CG-Verfahren
        Assemblieren der Matrizen $\mathbf{K}_\kappa(\mathbf{u}_{j+1,k+1})$, $\vec{J}(\mathbf{u}_{j+1,k+1})$
        Aufstellen der rechten Seite $\mathbf{rhs} = \mathbf{b}_{j+1} + \frac{1}{a_d\,\Delta t}\mathbf{M}_\varepsilon\,\mathbf{u}_j$:
        Update der Systemmatrix $\mathbf{A} = \left(\frac{1}{a_d\,\Delta t}\mathbf{M}_\varepsilon + \mathbf{K}_\kappa(\mathbf{u}_{j+1,k+1})\right)$
        Ermitteln von $r_{NL} = ||\mathbf{A}\,\mathbf{u}_{j+1,k+1} - \mathbf{rhs}||$
        Update der rechten Seite: $\mathbf{rhs} = \mathbf{rhs} + \vec{J}(\mathbf{u}_{j+1,k+1})\,\mathbf{u}_{j+1,\underline{k}}$
        Update Systemmatrix: $\mathbf{A} = \mathbf{A} + \vec{J}(\mathbf{u}_{j+1,k+1})\,\mathbf{u}_{j+1,\underline{k+1}}$
        $k = k + 1$
    end while
    Berechne SDIRK-Stufenableitungsvektor mit (6.3)
end function
```

Abbildung 6.1: Algorithmus für einen Zeitschritt im SDIRK3(2)-Verfahren als Pseudocode.

mit den SDIRK-Ableitungsvektor

$$\mathbf{u}'_l = \frac{1}{a_d \Delta t}\left(\mathbf{u}_l - \mathbf{u}_i - \Delta t \sum_{k=1}^{l-1} a_{l,k}\,\mathbf{u}'_k\right), \tag{6.3}$$

welcher aus den vorangegangenen Stufen ermittelt wird. Umstellen von (6.1) ergibt, dass pro Stufe das folgende System zu lösen ist:

$$\mathbf{u}_{j+1} = \left(\frac{1}{a_d\,\Delta t}\mathbf{M}_\varepsilon + \mathbf{K}_\kappa(\mathbf{u}_{j+1})\right)^{-1}\left(\frac{1}{a_d\,\Delta t}\mathbf{M}_\varepsilon\,\mathbf{u}_j + \mathbf{b}_{j+1}\right). \tag{6.4}$$

Die Implementation einer Stufe ist als Pseudocode in Abb. 6.1 dargestellt. Wie bereits in Kap. 2.4.1 dargestellt, sind die Vorteile des SDIRK3(2)-Verfahrens seine Stabilität und Robustheit, insbesondere bei steifen Problemen [43]. Dies ermöglicht es, zuverlässig Probleme auch mit größeren Zeitschrittweiten zu lösen.

In Abb. 5.13 ist dargestellt, welchen Zeitraum die einzelnen Komponenten im Lösungsprozess einnehmen. Dabei werden die Zeiten für die CPU-berechnete Referenzrechendauer als auch die Rechendauer des in Kap. 5.1 präsentierte Multi-GPU-PCG-Löser dargestellt. Man erkennt, dass sich durch die Verwendung von GPUs der prozentuale Anteil der Rechenzeit vom PCG-Löser hin zu den anderen Komponenten verschiebt, welche nun die schnelle Lösung maßgeblich beeinflussen. Hier ist insbesondere für größere Probleme, wie die in Kap. 7 definierten Beispielprobleme mit Ansatzfunktionen zweiter Ordnung, die Zeit zu nennen, die das Aufstellen des Vorkonditionierers benötigt.

Um den zeitlichen Aufwand des wiederholten Aufstellens des Vorkonditionierers zu reduzieren, wurde im Rahmen dieser Arbeit ein Ansatz vorgestellt [B10], welcher darauf abzielt, große Teile des einmal aufgestellten Vorkonditionierers wiederzuverwenden. Dies unterschlägt vorerst, dass sich die nichtlineare Leitwertmatrix und somit die Systemmatrix in jeder Stufe und jedem Zeitschritt ändert. Betrachtet man die in Kap. 2.2 vorgestellten Mikrovaristoren, stellt man fest, dass die sich deren Leitwert über einen großen Teil der Feldstärke-Leitwert-Kurve (Abb. 2.2) nur geringfügig ändern, um nach Überschreiten eines Schwellenwertes stark anzusteigen.

Es wird ein zweistufiger Ansatz vorgeschlagen: Über weite Teile, beispielsweise einer Sinus-Anregung eines Problems, ist das nichtlineare Material kaum leitfähig ($\kappa < 10^{-10}\frac{S}{m}$) und die Änderungen in der Systemmatrix sind gering, daher werden die gröberen Level des Vorkonditionierers konstant gehalten. Nur auf dem feinsten Level, auf welchem die Systemmatrix als Glätter genutzt wird, wird die aktuelle, korrekte Systemmatrix verwendet und so kleinere Differenzen durch den Jacobi-Glätter korrigiert.

Es wird die Iterationszahl des CG-Verfahrens für die Lösung der linearen Gleichungssysteme in den folgenden Zeitschritten und Stufen beobachtet und mit der Iterationszahl der ersten Iteration nach Aufstellen des Vorkonditionierers verglichen. Wird hier ein Schwellenwert $\epsilon_\kappa$ überschritten, wird der Vorkonditionierer angepasst. Dazu muss dieser nicht gänzlich neu aufgestellt werden. Im hier verwendeten Ansatz können der Restriktions- und Prolongationsoperator weiter genutzt werden , da sich das FEM-Gitter nicht ändert. Es wird nur das Galerkin-Produkt (4.16) für jedes Level des Vorkonditionierers neu errechnet. Damit wird eine Abweichung zur kompletten Neuaufstellung des Vorkonditionierers akzeptiert, die aus der nicht angepassten Glättung der Interpolationsmatrix $\mathbf{T}_{h+1}^{h}$ (vgl. Kap. 4.2.1) resultiert [90]. Allgemein zeigen die verwendeten Anwendungsbeispiele für elektroquasistatische Feldprobleme in Kap. 7, dass häufig nicht einmal die Schwelle zum Neuaufstellen des Vorkonditionierers erreicht wird, für welche ein Schwellenwert von $\epsilon_\kappa = 25\,\%$ gewählt wurde.

## 6.2 Ansätze zur Beschleunigung der EQS-Simulation unter Nutzung von expliziten Zeitintegrationsverfahren

Beim, in Kap. 2.4.2 vorgestellten, RKC-Verfahren [56] handelt es sich um ein explizites Zeitintegrationsverfahren, das sich auch für steife Probleme eignet. Es nutzt eine variable Anzahl interner Stufen, welche das Stabilitätsgebiet entlang der negativen reellen Achse erweitern. Allgemein benötigen explizite Zeitintegrationsverfahren kleinere Zeitschrittweiten als implizite Verfahren um stabil zu sein, was sie aufgrund der hohen Rechenzeit zeitlich unrentabel machen kann [43]. Ihr prinzipieller Vorteil ist jedoch, dass bei nichtlinearen Problemen keine Newton-Iterationen erforderlich sind. Dies wird aus der – bereits als (2.33) vorgestellten – Ausgangsformel für das RKC-Verfahren ersichtlich:

$$\mathbf{y}_0 = \mathbf{u}_i, \tag{6.5a}$$

$$\mathbf{y}_1 = \mathbf{y}_0 + \tilde{\mu}_1 \Delta t \mathbf{f}(t_i\,,\, \mathbf{u}_i), \tag{6.5b}$$

$$\begin{aligned} \mathbf{y}_j = & (1 - \mu_j - \nu_j)\mathbf{y}_0 + \mu_j \mathbf{y}_{j-1} + \nu_j \mathbf{y}_{j-2} + \\ & \tilde{\mu}_j\, \Delta t\, \mathbf{f}(t_i + \Delta t c_{j-1}\,,\, \mathbf{y}_{j-1}) + \tilde{\gamma}_j \Delta t \mathbf{f}(t_i\,,\, \mathbf{u}_i), \qquad \text{für } j = 2, .., s \end{aligned} \tag{6.5c}$$

$$\mathbf{u}_{i+1} = \mathbf{y}_s. \tag{6.5d}$$

Dabei ist die Formel für die Elektroquasistatik (2.21) die auszuwertende Funktion $\mathbf{f}(t, \mathbf{u})$, welche umgestellt für das RKC-Verfahren wie folgt aussieht:

```
function RKC_Schritt($\mathbf{u}_0$,$\mathbf{u}$,$j$,$t$,$\Delta t$)          ▷ Aus $\mathbf{u}_0$ wird $\mathbf{u}$ errechnet
    Initialisiere Koeffizienten $\tilde{\mu}_1$
    $\mathbf{f0} \leftarrow \mathbf{f}(t, \mathbf{u}_0)$          ▷ Hier wird eine CG-Iteration durchgeführt
    $\mathbf{y1} \leftarrow \mathbf{u}_0 + \tilde{\mu}_1 \Delta t\, \mathbf{f0}$
    $\mathbf{ym1} \leftarrow \mathbf{y1}$   ;   $\mathbf{ym2} \leftarrow \mathbf{u}_0$
    for j=2,...,s do
        Aktualisiere Koeffizienten $\mu_j$, $\nu_j$, $\tilde{\mu}_j$, $\tilde{\gamma}_j$
        $\mathbf{fn} \leftarrow \mathbf{f}(t + c_j\, \Delta t, \mathbf{ym1})$   ▷ Hier wird eine CG-Iteration durchgeführt
        $\mathbf{y} \leftarrow (1 - \mu_j - \nu_j)\mathbf{u}_0 + \mu_j \mathbf{ym1} + \nu_j \mathbf{ym2} + \tilde{\mu}_j\, \Delta t\, \mathbf{fn} + \tilde{\gamma}_j\, \Delta t\, \mathbf{f0}$
        Verschiebe Vektoren: $\mathbf{ym2} \leftarrow \mathbf{ym1}$ ; $\mathbf{ym1} \leftarrow \mathbf{y}$
    end for
    $\mathbf{u} \leftarrow \mathbf{y}$
end function
```

Abbildung 6.2: Algorithmus für einen Zeitschritt im RKC-Verfahren als Pseudocode.

$$\mathbf{f}(t_i, \mathbf{u}_i) = \frac{d\mathbf{u}_i}{dt} \;=\; \mathbf{M}_\varepsilon^{-1}\Big(-\; \mathbf{K}_\kappa(\mathbf{u}_i)\, \mathbf{u}_i + \;\mathbf{b}(t)\Big)\,. \tag{6.6}$$

Zur Funktionsauswertung ist die Inverse der Masse-Matrix $\mathbf{M}_\varepsilon$ zu bestimmen. Da $\mathbf{M}_\varepsilon$ nicht nur aus einer Hauptdiagonalen besteht und somit nicht direkt invertierbar ist, erfolgt die Lösung iterativ über ein PCG-Verfahren. Daher spricht man hier auch von einem semi-expliziten Verfahren. Dieses muss über einen schnellen PCG-Löser verfügen, um zeitlich effizient zu sein. Die Funktionsauswertungen $\mathbf{f}(t_i, \mathbf{u}_i)$ in (6.5c) findet mit bekannten Werten $\mathbf{u}_i$ bzw. $\mathbf{y}_{j-1}$ statt. Die nichtlineare Matrix $\mathbf{K}_\kappa(\mathbf{u}_i)$ in der auszuwertenden Funktion (6.6) wird daher mit bereits bekanntem Wert assembliert. Im impliziten Zeitintegrationsverfahren erfolgt die Assemblierung $\mathbf{K}_\kappa(\mathbf{u}_{i+1})$ hingegen mit einem noch zu bestimmenden $\mathbf{u}_{i+1}$, was ein sukkzessive Linearisierung erforderlich macht. Abb. 6.2 stellt die Berechnung eines Zeitschrittes noch einmal als Pseudoalgorithmus dar.

Da die nichtlineare Matrix $\mathbf{K}_\kappa(\mathbf{u}_i)$ mit bereits bekanntem Wert assembliert wird, kann sie auf die rechte Seite gezogen werden. Das Ergebnis ist ein Multiple Right-Hand-Side (MRHS)-Problem mit konstanter Systemmatrix. Durch diese konstante Systemmatrix kann zu Beginn ein exakter Vorkonditionierer erstellt werden, welcher in jedem Lösungsschritt verwendet werden kann. Eine wesentliche Problemstellung des vorangehenden Kapitels 6.1 entfällt somit. Weiterhin reduziert der Wegfall der Newton-Iterationen die Anzahl der Assemblierungsvorgänge deutlich, welche zusätzlich nicht mehr mit der Permittivitätsmatrix addiert werden müssen. Weiterhin kann hier für $\mathbf{K}_\kappa(\mathbf{u}_i)\,\mathbf{u}_i$ das in Kap. 5.2.2 vorgestellte, kombinierte Assemblierungs- und SpMV-Verfahren eingesetzt werden, was die Aufstellung des linearen Gleichungssystem noch einmal deutlich beschleunigt.

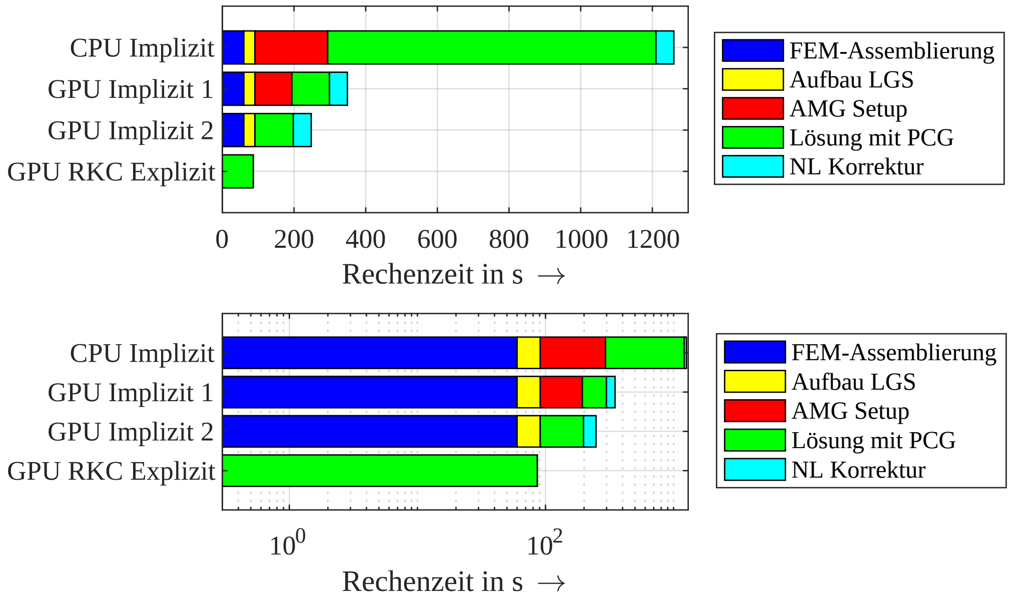


Abbildung 6.3: Rechendauer für eine Stufe für drei implizite SDIRK3(2)-Zeitintegratoren („CPU Implizit“, „GPU Implizit 1“ und „GPU Implizit 2“) und den expliziten RKC-Zeitintegratoren anhand eines Beispielproblems (Kap. 7 - Problem Nr. 5) unterteilt in die einzelnen Abschnitte gem. der Pseudoalgorithmen Abb. 6.1 bzw. Abb. 6.2. Darstellung in linearer und logarithmischer Skalierung.

Die Abb. 6.3 stellt eine Erweiterung der Abb. 5.13 dar und verdeutlicht den wesentlich geringeren Zeitaufwand für die Berechnung einer Stufe mit RKC-Verfahren. Neben den impliziten Zeitintegratoren auf Basis des SDIRK3(2)-Verfahrens ist auch die Zeitverteilung einer Stufe für das RKC-Verfahren dargestellt. Im Einzelnen bezeichnet „Trilinos ML“ das implizite SDIRK3(2)-Verfahren unter Verwendung des CPU-basierten AMG-PCG

Lösers basierend auf Trilinos ML. „Implizit 1“ bezeichnet dasselbe Zeitintegrationsverfahren mit dem in Kap. 5 vorgestellten Multi-GPU-PCG-Löser [B7]. „Implizit 2“ bezeichnet die in Kap. 6.1 vorgestellt Modifikation des impliziten Zeitintegrators, in welcher der AMG-Vorkonditionierer möglichst weitgehend wiederverwendet wird [B10]. „RKC“ bezeichnet den expliziten RKC-Zeitintegrator. Dieser nutzt zusätzlich die GPU-basierte Assemblierungs- und SpMV-Kombination, welche in Kap. 5.2.2 vorgestellt wurde [B1].

Der große Geschwindigkeitsgewinn durch die Verwendung des Multi-GPU AMG-PCG-Lösers wurde bereits in Kap. 5.1.3.4 aufgezeigt. Im Ansatz „Implizit 2“ kann der nun im Ansatz „Implizit 1“ prozentual deutlich höhere Anteil für das AMG-Setup auf einen vernachlässigbar hohen Wert reduziert werden. Mit dem RKC-Verfahren wird abschließend die Zeit für die Reassemblierung der Leitwertmatrix deutlich reduziert. Damit benötigt eine Stufe eines Zeitschrittes noch einmal deutlich weniger Rechenzeit. Das RKC-Verfahren benötigt für die Berechnung einer gesamte Stufe nur noch die Zeit, welche die impliziten Verfahren für die Assemblierung der Matrix benötigen.

Wie jedoch bereits genannt, ist die Anzahl notwendiger Zeitschritte für das RKC-Verfahren zum Erreichen gleicher Genauigkeit deutlich höher. Für die hier verwendeten elektroquasistatischen Modelle ist im Allgemeinen mit einem Faktor von zwei bis fünf zu rechnen. Untersuchungen an Anwendungsbeispielen sind in Kap. 7.5 dargestellt.

Um transiente Effekte möglichst zu minimieren, kann die Anregungsfunktion, etwa eine Sinus-Spannung mit $f = 50\,\mathrm{Hz}$, zu Beginn der Simulation mit einer linear steigenden Funktion multipliziert werden, um einen aufschwingenden Sinus zu erzeugen [29]. Eigene Versuche haben gezeigt, dass bei den verwendeten expliziten Verfahren, anders als bei dem impliziten SDIRK3(2)-Verfahren, die Simulation nicht erfolgreich durchgeführt werden kann.

Mit der in [29] vorgestellten Anregungsfunktion

$$F_{alt}(t) = \begin{cases} \hat{U}\,\dfrac{4}{3\,T}\,t\,\sin(\omega t)\;, & t \leq \dfrac{3}{4}T \\ \hat{U}\,\sin(\omega t)\;, & t > \dfrac{3}{4}T \end{cases} \tag{6.7}$$

kann die elektroquasistatische Simulation am Ende des Aufschwingvorgangs bei Verwendung expliziter Zeitintegrationsansätze instabil werden, selbst wenn sehr kleine Zeitschrittweiten und hohe Stufenzahl gewählt werden. Daher wurde diese zu

$$F_{neu}(t) = \begin{cases} \hat{U}\,\dfrac{2}{T}\,t\,\sin(\omega t)\,, & t \leq \dfrac{T}{2} \\ \hat{U}\,\sin(\omega t)\,, & t \leq \dfrac{T}{2} \end{cases} \tag{6.8}$$

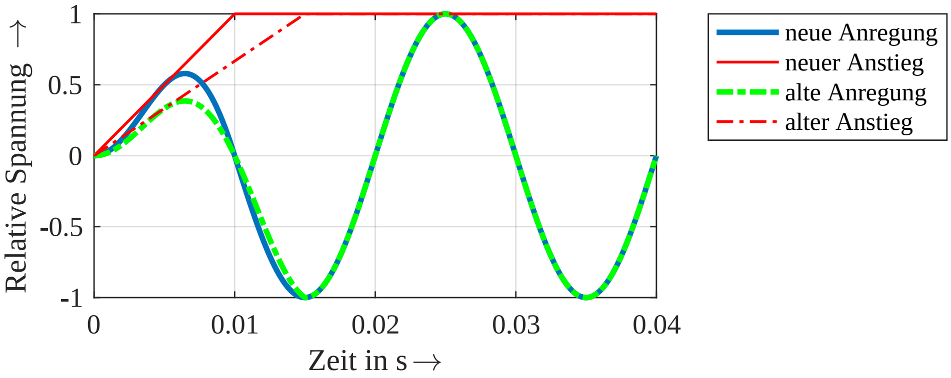
angepasst, womit der $C_0$-Punkt vom Maximum in den Nulldurchgang des Sinus verlegt wird.

Der Verlauf der neuen und alten Anregungsfunktion ist in Abb. 6.4a dargestellt. Ferner sind die alte und neue Anstiegsfunktion zu sehen, welche auf eine Sinusspannung aufmultipliziert werden.

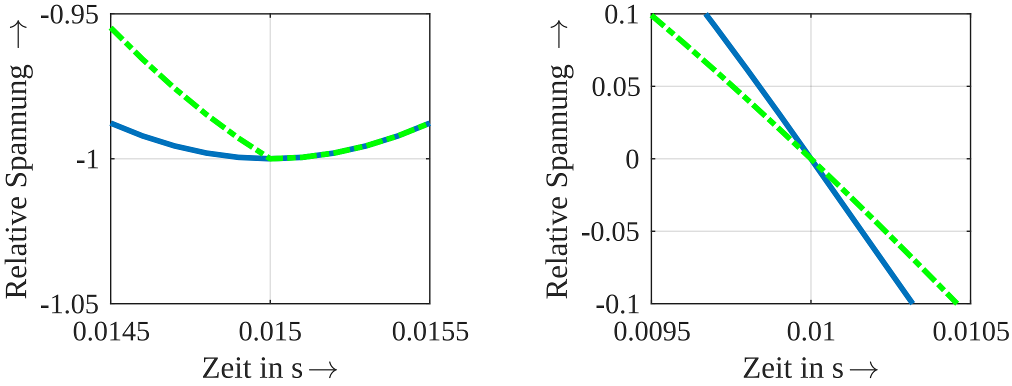
Abb. 6.4b zeigt die Anregungsfunktionen am Ende des alten Anstiegs. Man erkennt den Knick in der alten Anregungsfunktion. Wird nun beispielsweise mittels Trapezansatz eine numerische Abbildung zum Zeitpunkt 15 ms gebildet, hat diese einen negativen Wert, während sie eigentlich 0 sein müsste. In Abb. 6.4c ist hingegen der Verlauf der Anregung zu sehen, wenn der lineare Anstieg im Nulldurchgang des Sinus endet. Hierdurch wird erreicht, dass das explizite Verfahren bei Übergang vom aufschwingenden Sinus zum normalen Sinus wesentlich stabiler ist und nicht am Ende des aufschwingenden Sinus divergiert. Des weiteren ist davon auszugehen, dass am Nulldurchgang der anliegenden Spannung wegen $||\mathbf{u}_D|| = 0$ auch $||\mathbf{u}|| \approx 0$ ist. Das führt für Mikrovaristormaterial – welches bei einem $|\vec{E}| \approx 0$ praktisch nicht leitfähig ist (vgl. Abb. 2.2) – wiederum zu sehr geringen Werten in der Leitwertmatrix $\mathbf{K}_\kappa\big(\mathbf{u}\big)$ und der Jacobimatrix $\mathbf{J}\Big(\mathbf{K}_\kappa(\mathbf{u})\Big) = \frac{\mathrm{d}\mathbf{K}_\kappa(\mathbf{u})}{\mathrm{d}\mathbf{u}}$. Daher ist das Problem zu diesem Zeitpunkt, am Nulldurchgang der anliegenden Spannung, weniger steif, als zu Zeiten höherer Spannung, an denen das Mikrovaristormaterial aussteuert.

### 6.2.1 Ansätze zur Stufenwahl

In jedem Zeitschritt ruft der RKC-Algorithmus $s$ Stufen, mit $s \geq 2$, rekursiv auf. Hierdurch wird eine Stabilitätsregion entlang der negativen reellen Achse



(a) Alte und neue Anregungsfunktion.



(b) Altes Ende des Anstiegs.

(c) Neues Ende des Anstiegs.

Abbildung 6.4: Modifikation der anregenden 50 Hz-Sinusspannung durch Anpassung der linearen Spannungssteigung zu Simulationsbeginn.

geschaffen, deren Ränder sich mit $\beta(s) = 0,653s^2$ approximieren lassen [58] (vgl. Kap. 2.4.2).

[56] schlägt vor, die Anzahl der Stufen $s$ in Abhängigkeit vom Spektralradius $\sigma$ der Jacobimatrix $\mathbf{J}(\mathbf{f}(t, \mathbf{u}))$ von (6.6) und der Zeitschrittweite $\Delta t$ zu wählen:

$$s = \sqrt{\frac{\Delta t\, \sigma(\mathbf{J}(\mathbf{f}(t, \mathbf{u})))}{0,653}} \tag{6.9}$$

Die Zeitschrittweite $\Delta t$ wird in Abhängigkeit vom akzeptierten Zeitintegrationsfehler gewählt. Davon ausgehend wird mit (6.9) die Anzahl der Stufen gewählt, dass das Verfahren für $\Delta t$ stabil ist. Auf dieser Grundlage wurden

```
function RKC_Zeitintegration($\mathbf{u}_0$,$t$,$T$)          ▷ Zeitintegration bis $t = T$
    Bestimme/Setze Zeitschrittweite $\Delta t$
    Bestimme/Setze Stufenzahl $s$
    um1 ← $\mathbf{u}_0$
    while ($t \leq T$) do
        RKC_Schritt(um1,u,$s$,$t$,$\Delta t$)                     ▷ vgl. Abb. 6.2
        Schätze Zeitintegrationsfehler $e$
        if $e > tol$ then                          ▷ Zeitschritt wird verworfen
            Bestimme Spektralradius $\sigma(\mathbf{J}(\mathbf{f}(t,\mathbf{u})))$ und passe $s$ an
            Passe Zeitschrittweite $\Delta t$ an
        else                                       ▷ Zeitschritt wird akzeptiert
            um1 ← u
            $t \leftarrow t + \Delta t$                       ▷ Gehe zum nächsten Zeitschritt
            Speichere u für Postprocessing
            if 25 akzeptierte Zeitschritte then
                Bestimme Spektralradius $\sigma(\mathbf{J}(\mathbf{f}(t,\mathbf{u})))$ und passe $s$ an
            end if
        end if
    end while
end function
```

Abbildung 6.5: Algorithmus für einen Zeitschritt im RKC-Verfahren mit Spektralradiusbestimmung durch Power-Iteration als Pseudocode.

verschiedene Ansätze für EQS-Probleme implementiert, welche im Folgenden beschrieben werden.

Zum einen wurde der Vorschlag aus [56] übernommen. Hier wird der Spektralradius, bzw. eine obere Abschätzung mittels Rayleigh-Quotient durch eine Power-Methode ermittelt. Als Startwert wird der ermittelte Eigenvektor der letzten Power-Methoden-Iteration oder bei noch nicht vorhandenem Eigenvektor die letzte berechnete Ableitung $\mathbf{f}(t, \mathbf{u})$ gewählt. Die Iteration wird dabei alle 25 Zeitschritte oder bei einer Änderung der Zeitschrittweite, aufgrund eines zu großen geschätzten Zeitintegrationsfehlers, durchgeführt. Der Ansatz ist als Pseudocode in Abb. 6.5 dargestellt.

Für die Nutzung bei nichtlinearen Simulationen mit Mikrovaristoren ergeben sich hierbei folgende Nachteile: Erstens ist aufgrund des Problemtyps bei jeder Iteration ein lineares Gleichungssystem zu lösen. Dies macht eine Power-Iteration sehr aufwendig, da diese im Schnitt deutlich mehr Iterationen benötigt, als für die Summe der Stufen aller zwischen zwei Power-Iterationen

```
function RKC_Zeitintegration($\mathbf{u}_0$,$t$,$T$)    ▷ Zeitintegration bis $t = T$
    Bestimme/Setze Zeitschrittweite $\Delta t$
    Bestimme/Setze Stufenzahl $s$
    um1 ← $\mathbf{u}_0$
    while ($t \leq T$) do
        RKC_Schritt(um1,$\mathbf{u}$,$s$,$t$,$\Delta t$)    ▷ vgl. Abb. 6.2
        Schätze Zeitintegrationsfehler $e$
        if $e > tol$ then    ▷ Zeitschritt wird verworfen
            Erhöhe Stufenzahl $s$
            if Zweiter verworfener Zeitschritt hintereinander then
                $t \leftarrow t - \Delta t$    ▷ Gehe zum vorherigen Zeitschritt
                Bestimme/Ändere Zeitschrittweite $\Delta t$
                Passe um1 aus gespeicherten Daten an
            end if
        else    ▷ Zeitschritt wird akzeptiert
            um1 ← $\mathbf{u}$
            $t \leftarrow t + \Delta t$    ▷ Gehe zum nächsten Zeitschritt
            Speichere $\mathbf{u}$ für Postprocessing
            if 10 akzeptierte Zeitschritte then
                Reduziere Stufenzahl $s$
            end if
        end if
    end while
end function
```

Abbildung 6.6: Algorithmus für einen Zeitschritt im RKC-Verfahren mit Heuristik als Pseudocode.

liegenden Zeitschritte. Zweitens ist diese bei Aussteuerung des Materials deutlich häufiger als die angestrebten 25 Zeitschritte zu wiederholen. Dadurch erhöht sich die Anzahl der notwendigen Iterationen in einer Anzahl, dass mindestens doppelt so viele lineare Systeme gelöst werden müssen, als dies ohne Power-Iteration der Fall wäre. Dies hat einen massiven Einfluss auf die benötigte Simulationszeit und damit auf die Konkurrenzfähigkeit der expliziten RKC-Zeitintegrationsmethode.

Als robust und bzgl. der Rechenzeit am effektivsten hat sich ein heuristisches Verfahren hinsichtlich der Wahl der Stufen und Zeitschrittweite erwiesen, welches in Abb. 6.6 dargestellt ist. Hierbei wird der lokale Zeitintegrationsfehler mit dem Fehlerschätzer aus (2.35) nach [56] geschätzt. Ziel ist es, dass der relative Zeitintegrationsfehler $e_{rel} = \frac{||\mathbf{e}||}{||\mathbf{u}||}$ in einem Wertebereich $a \leq e_{rel} \leq b$ in Abhängigkeit von einem gewählten Toleranzwert $\epsilon$ gehalten wird. Für die betrachteten Beispiele der Elektroquasistatik haben sich hier die Werte $a = 10^{-2}\,\epsilon$ und $b = 0,7\,\epsilon$ bewährt. Bei Überschreiten der Fehlerschranke wird der Zeitschritt erneut mit einer erhöhten Anzahl an Stufen gerechnet. Ist auch dies nicht erfolgreich, geht der Algorithmus zum Zeitschritt $t_{i-1}$ zurück und beginnt die Rechnung mit kleinerer Zeitschrittweite von dort erneut. Der Hintergrund hierfür ist, dass das Stabilitätsgebiet oft schon beim Ausgangswert vor Auftreten des zu hohen Zeitintegrationsfehlers verlassen wurde und weder eine Erhöhung der Stufen noch eine Zeitschrittreduktion das Divergieren der Lösung verhindern können.

Trotz dieser Ansätze ist festzustellen, dass für die gegebene Problemklasse die Nutzung eines expliziten Ansatzes mit einer erheblich höheren Gefahr versehen ist, dass die Rechnung aufgrund zu groß gewählter Schrittweite scheitert. Hier erweist sich das implizite Verfahren aufgrund seiner A-Stabilität als deutlich robuster [43].

### 6.2.2 Startwertschätzung mittels „Subspace Projection Extrapolation“ -Methode

Wie in Kap. 2.4.2 dargestellt, muss man aufgrund der nichttrivialen Massenmatrix, trotz explizitem Zeitschrittverfahren, zur Lösung von (6.6) Matrizen invertieren. Da die Massenmatrix $\mathbf{M}_\varepsilon$ konstant ist, wird das lineare Gleichungssystem

$$\mathbf{f}_i \;=\; \mathbf{f}(t_i, \mathbf{u}_i) \;=\; \mathbf{M}_\varepsilon{}^{-1}\Big(-\;\mathbf{K}_\kappa(\mathbf{u}_i)\,\mathbf{u}_i +\;\mathbf{b}(t)\Big) \tag{6.10}$$

zu einem MRHS-Problem, in welchem sich nur die rechte Seite $\mathbf{b}_i = -\,\mathbf{K}_\kappa(\mathbf{u}_i)\,\mathbf{u}_i +\;\mathbf{b}(t)$ zeitabhängig ändert.

Die Subspace Projection Extrapolation (SPE) [126, 127] ist eine Methode, Informationen aus bereits ermittelten Lösungen $\mathbf{f}_i$ des LGS zu nutzen, um für kommende Lösungsvorgänge einen Startvektor $\mathbf{f}_{0,i+1}$ zu ermitteln. Dieser wird aus dem Unterraum der Orthonormal-Vektoren $\mathbf{Q} \in \mathbb{R}^{n\times R}$ der Lösungen $\mathbf{f}_r$ der letzten $R$ Zeitschritte generiert. Durch Vorgabe eines Startvektor für das CG-Verfahren, welcher bereits nahe am Ergebnisvektor liegt, können die Anzahl der Iterationen reduziert und der Lösungsprozess beschleunigt werden. Dabei führt der Prozess folgende Schritte durch:

1. Erzeugen von orthonormalen Vektoren $\mathbf{q}_r$ aus den Lösungsvektoren der $R$ letzten Lösungsvektoren $\mathbf{f}_r$ für $\tilde{r} \leq r \leq i$ mit $\tilde{r} = i - R$ mittels eines Modified Gram-Schmidt (MGS)-Algorithmus :

$$\begin{aligned} \mathbf{q}_1 &= \mathbf{f}_{\tilde{r}}, \\ a_r &= \frac{\left\langle \mathbf{f}_{(\tilde{r}+r-1)} \, , \, \mathbf{q}_{r-1} \right\rangle}{\left\langle \mathbf{q}_{r-1} \, , \, \mathbf{q}_{r-1} \right\rangle} \\ \mathbf{q}_r &= \frac{\mathbf{f}_{(\tilde{r}+r-1)} - a_r \, \mathbf{q}_{r-1}}{||\mathbf{f}_{(\tilde{r}+r-1)} - a_r \, \mathbf{q}_{r-1}||}, \qquad r = 2, .., R\,. \end{aligned} \tag{6.11}$$

2. Erstellen der Projektionsmatrix $\mathbf{Q} = [\mathbf{q}_1|...|\mathbf{q}_R]$.

3. Erzeugen der projizierten Systemmatrix $\mathbf{M}_{\varepsilon,p} \in \mathbb{R}^{R\times R}$ und der projizierten rechten Seite $\mathbf{b}_{p,i+1} \in \mathbb{R}^R$

$$\begin{aligned} \mathbf{M}_{\varepsilon,p} &= \mathbf{Q}^T \mathbf{M}_\varepsilon \mathbf{Q}, \\ \mathbf{b}_{p,i+1} &= \mathbf{Q}^T \mathbf{b}_{i+1}. \end{aligned} \tag{6.12}$$

4. Lösen des projizierten LGS $\mathbf{f}_{p,i+1} = \mathbf{M}_{\varepsilon,p}^{-1} \, \mathbf{b}_{p,i+1}$ der Größe $m \times m$ mit einem direkten Löser. Hierbei ist zu beachten, dass die projizierte rechte Seite $\mathbf{b}_{p,i+1}$ jedes Mal erneut berechnet werden muss. Da sich die Systemmatrix $\mathbf{A} = \mathbf{M}_\varepsilon$ nicht ändert, kann die projizierte Systemmatrix $\mathbf{M}_{\varepsilon,p}$ jedoch unverändert gehalten werden und muss erst bei einer neuen Projektionsmatrix $\mathbf{Q}$ neu berechnet werden.

5. Erzeugen des extrapolierten Vektors $\mathbf{f}_{0,SPE,i+1} = \mathbf{Q} \, \mathbf{f}_{p,i+1}$.

Der Gesamtprozess ist mittels CUBLAS-Routinen [67] für die GPU implementiert worden. Lediglich der direkte LU-Solver wird auf der CPU ausgeführt, da dieser für kleine Gleichungssysteme die hohe Parallelität einer GPU nicht ausreichend nutzen kann, um den Prozess gegenüber einer CPU-Implementierung zu beschleunigen. Die Schritte in den (6.11) und

(6.12) sind eine Abfolge von Vektor-Vektor-Rechnungen (BLAS Level 1) oder Matrix-Vektor-Produkten (BLAS Level 2) mit einer voll besetzten Matrix. Hier kann eine GPU ihre maximalen Geschwindigkeitswerte erzielen.

### 6.2.3 Startwertgenerierung mittels Verfahren zur Modellordnungsreduktion

Ein anderer Ansatz, welcher Gegenstand aktiver Forschung ist, ist die Modellordnungsreduktion (engl.: Model Order Reduction (MOR)). [128] gibt eine Übersicht zum Stand der Forschung.

In dieser Arbeit wird MOR basierend auf der Proper Orthogonal Decomposition (POD) betrachtet. Diese ist aufgrund ihrer Anwendbarkeit für lineare und nichtlineare Systeme zu einer weit verbreiteten Methode für die Berechnung reduzierter Basen geworden. Die POD-MOR wird sowohl zu Kompression von Bildern als auch Daten eingesetzt [129]. Neben Gebieten wie der Fluiddynamik, chemischen Reaktionen oder biologischem Wachstum findet die POD-MOR in der Elektrotechnik Verwendung [130, 131, 132, 133, 134].

Üblicherweise sollen mit der MOR sofort die Lösung der Zeitschritte $\mathbf{u}_i$ generiert werden, ohne das eine iterative Lösung des LGS erforderlich ist. Aufgrund seines Aufbaus benötigt das RKC-Verfahren jedoch die diskrete Ableitung $\mathbf{f}_i$. Es wird daher im folgenden ein Ansatz verfolgt, ein $\mathbf{f}_{MOR,i}$ zu generieren. Die sofortige Generation von $\mathbf{u}_{MOR,i}$ hat sich für die betrachteten nichtlinearen Probleme der Elektroquasistatik als nicht ausreichend genau erwiesen. Im Gegensatz zur Generation von $\mathbf{f}_{MOR,i}$ kann der Fehler jedoch nicht durch eine geringe Anzahl von CG-Iterationen korrigiert werden.

Ausgangspunkt ist erneut (6.10). Bei der MOR ist es das Ziel, aus einer ausreichenden Anzahl an verfügbaren Lösungen, sogenannten „Snapshots“ [135], das dynamische Verhalten des Systems zu erfassen und auf seine bestimmenden Kernkomponenten zu reduzieren. Die Snapshot-Matrix $\mathbf{X} = \{\mathbf{f}_1|...|\mathbf{f}_R\} \in \mathbb{R}^{n\times R}$ wird durch Singulärwertzerlegung (engl.: Singular Value Decomposition (SVD) ) in $R$ links- und rechtssinguläre Vektoren, sowie seine Singulärwerte zerlegt [136, 137].

Mittels der linkssingulären Vektoren kann das LGS in ein reduziertes

System der Dimension $R$ überführt werden. Die Anzahl der verwendeten linkssingulären Vektoren kann weiter auf $W < R$ reduziert werden, wenn nur die linkssingulären Vektoren verwendet werden, welche zu den $W$ größten Singulärwerten korrespondieren. Durch Nutzung des so reduzierten Systems wird der Rechenaufwand zur Lösung des linearen Gleichungssystems erheblich reduziert. Im Allgemeinen soll das Lösen des Ausgangssystems hierdurch überflüssig werden und nur noch das reduzierte System mittels eines direkten Gleichungssystemlösers gelöst werden. Dies führt dann zu einer erheblichen Beschleunigung des Lösungsvorganges.

In dieser Arbeit wird MOR jedoch eingesetzt, um, vergleichbar mit der SPE in Kap. 6.2.2, erneut einen Startwert $\mathbf{f}_{0,i+1}$ für das vollständige System zu generieren. Im optimalen Fall wird dabei vor der ersten Iteration des CG-Lösers festgestellt, dass das Residuum der Startlösung $\mathbf{f}_{0,i+1}$ bereits so niedrig ist, dass keine weitere Iteration erforderlich ist. Anderenfalls wird jedoch nachiteriert und eine Lösung, welche den geforderten Genauigkeitsanforderungen entspricht, garantiert. Dies ist insbesondere aufgrund der Nichtlinearität der betrachteten Systeme erforderlich.

Die MOR wurde analog zur SPE wie folgt umgesetzt:

1. Sammeln einer Anzahl $R$ an Lösungsvektoren $\mathbf{f}_r, \quad r = 1, .., R$ und erstellen der Lösungsvektormatrix $\mathbf{X} = \{\mathbf{f}_1|...|\mathbf{f}_R\}$.

2. Durchführen einer Singulärwertzerlegung $\mathbf{X} = \mathbf{U\Sigma V}$ mit $\mathbf{U}$ als Matrix der linkssingulären Vektoren und $\mathbf{V}$ als Matrix der rechtssingulären Vektoren. Die Matrix $\mathbf{\Sigma}$ enthält auf ihrer Hauptdiagonalen die Sigulärwerte von $\mathbf{X}$ in absteigender Reihenfolge.

3. Berechnung der reduzierten Systemmatrix $\mathbf{M}_{\varepsilon,p} = \mathbf{U}^T\mathbf{M}_\varepsilon\mathbf{U}$. Dies kann wie bei der SPE einmal erfolgen und anschließend bis zu einer Änderung der verwendeten Snapshots gehalten werden.

4. Berechnung des reduzierten RHS-Vektor $\mathbf{b}_{p,i+1} = \mathbf{U}^T\,\mathbf{b}_{i+1}$.

5. Lösung des reduzierten Systems $\mathbf{f}_{p,i+1} = \mathbf{M}_{\varepsilon p}^{-1}\,\mathbf{b}_{p,i+1}$ mittels eines direkten Lösers.

6. Rücktransformation der reduzierten Lösung auf die vollständige Lösung mit $\mathbf{f}_{0,POD,i+1} = \mathbf{U}\,\mathbf{f}_{p,i+1}$.

Als Startwert für das CG-Verfahren kann nun anschließend $\mathbf{f}_{0,POD,t+1}$ genutzt werden. Für die Singulärwertzerlegung wurde in dieser Arbeit die Bibliothek „EIGEN 3.3“ verwendet [138].

Als Modifikation des MOR-Algorithmus, welcher dann deutliche Ähnlichkeit mit dem SPE-Algorithmus aufweist, können auch die $R$ letzten Lösungen verwendet werden. Der Algorithmus folgt also der Lösung, muss allerdings auch mehrfach neu berechnet werden, was nur bei einer kleinen Anzahl an Lösungsvektoren rentabel ist. Punkt 1 ändert sich dann wie folgt:

1. Sammeln einer Anzahl $R$ an Lösungsvektoren $\mathbf{f}_r, \quad r = (i-R), .., i$ und erstellen der Lösungsvektormatrix $\mathbf{X} = \{\mathbf{f}_{(i-R)}|...|\mathbf{f}_i\}$

Diese Variante ist der in Kap. 6.2.2 vorgestellten SPE sehr ähnlich. Das MGS-Verfahren der SPE und die SVD der MOR spannen die gleichen Unterräume auf. Wesentlicher Vorteil der SVD ist die Sortierung der Eigenwerte und damit die Möglichkeit, nur die Spalten der Matrix $\mathbf{U}$ zur Reduktion zu verwenden,die zu den $W$ größten Eigenwerten korrespondieren. In der Praxis ergeben sich bei beiden Ansätzen durch die Verwendung einer anderen Basis andere Ergebnisse. Praktische Beispiele hierzu werden in Kap. 7 diskutiert.

Alle Algorithmen dieses Kapitels verwenden die in der linearen Algebra übliche (diskrete) $l2$-Vektornorm. Diese berücksichtigt jedoch keine Informationen zu Raum und Materialien, z.B. Kantenlängen oder Permittivität, so wie es beispielsweise die kontinuirliche $L2$ oder eine Energienorm täte. Eine einfache Modifikation z.B. der SPE oder MOR-Ansätze wäre durch die Verwendung entsprechender gewichteter Normen möglich [139].

## 6.3 Zusammenfassung

Das Kapitel befasst sich mit Möglichkeiten zur Beschleunigung der Zeitintegration für das räumlich diskretisierte elektroquasistatische System gewöhnlicher Differentialgleichungen.

Hierzu wird das implizite Zeitintegrationsverfahren SDIRK3(2) betrachtet. Aufgrund der veränderten Zeitanteile bei Verwendung von GPUs wird die Verwendung eines Ansatzes entwickelt [B9, B10], bei dem der AMG-Vorkonditionierer wiederverwendet und nicht an die aktuelle Systemmatrix angepasst wird, solange ein vorgegebener Schwellenwert nicht überschritten wird.

Als explizites stabilisiertes Zeitintegrationsverfahren wird das RKC-Verfahren betrachtet. Die Struktur eines expliziten Verfahren hat Vorteile bei der Verwendung von GPUs. Hierzu zählen die konstante Systemmatrix und der dadurch nicht zu adaptierende Vorkonditionierer und die Nutzung der GPU-beschleunigten Assemblierung- und Matrixmultiplikation für die Leitwertmatrix, welche in die rechte Seite des LGS eingehen. Für das RKC-Verfahren werden Modifikationen der Anregungsfunktion und Möglichkeiten der adaptiven Zeitschrittweiten- und Stufensteuerung nach [B1] diskutiert. Das sich für den expliziten Ansatz ergebende MRHS-Problem ermöglicht die Verwendung eines Startwertschätzers für die iterative Lösung des LGS. Hier werden die Startwertschätzung auf Basis von SPE und POD-MOR betrachtet.

# Kapitel 7

# Validierung anhand von Anwendungsbeispielen

In diesem Kapitel werden die in Kap. 5 und Kap. 6 präsentierten Ansätze anhand von mehreren Anwendungsbeispielen getestet. Um eine Vergleichbarkeit zu gewährleisten, wurden die Probleme auf verschiedenen Rechensystemen simuliert.

## 7.1 Verwendete Rechensysteme

Im Rahmen des DFG-Antrages Nr. DFG INST 218/54-1 FUGG, „Hochschulweiter CPU- und GPU-Hochleistungsrechner, Bergische Universität Wuppertal“ wurde an der Bergischen Universität Wuppertal ein Rechencluster beschafft, mit dem hochauflösende Modelle im Bereich der Elektroquasistatik GPU-beschleunigt berechnet werden können. Der Gesamtcluster besteht aus acht Knoten mit jeweils fünf angeschlossenen Grafikkarten des Typs Nvidia Tesla K20m. Da sich diese Arbeit thematisch auf die Nutzung mehrerer GPUs auf einem Hostrechner stützt, werden die Rechnungen auf einem Host durchgeführt. Ferner wird noch ein Server mit einer Nvidia Tesla K80 GPU genutzt. Bei diesem Streamingprozessor ist zu beachten, dass er über zwei GPUs mit jeweils eigenem Speicher verfügt. Diese verhalten sich also aus Sicht des Simulationscodes wie zwei separate GPUs. Die Spezifikationen der Rechensysteme und der angeschlossenen GPUs werden in Tab. 7.1 dargestellt.

| Systemnummer | 1 | 2 |
|---|---|---|
| CPU-Modell | 2x Intel Xeon E5-2660 | 2x Intel Xeon E5-2660 v3 |
| Rechenkerne | 16 | 20 |
| Threads | 16 | 40 |
| Grundfrequenz | 2,2 GHz | 2,6 GHz |
| Turbofrequenz | 3,0 GHz | 3,3 GHz |
| Arbeitsspeicher | 128 GB | 256 GB |
| Betriebssystem | Ubuntu 14.04 LTS | Ubuntu 14.04 LTS |
| C++-Compiler | GCC 4.7.3 | GCC 4.7.3 |
| CUDA-Compiler | v.7.5 | v.7.5 |
| Architekturmodell | v.3.5 | v.3.5 |
| Anzahl | 5 | 1 |
| Modell | Nvidia Tesla K20m | Nvidia Tesla K80 |
| Architektur | Kepler | Kepler |
| GPU | GK110 | GK210 |
| CUDA Cores | 2496 | 2x2880 |
| SMs | 13 | 2x13 |
| DP-Rechenleistung | 1,175 TFLOPS | 2,91 TFLOPS |
| CUDA Core Takt | 706 MHz | 560 MHz - 875 MHz |
| Globaler Speicher | 5 GB | 2x 12GB |
| Speicherbandbreite | 208 GB | 2x 240 GB/s |

Tabelle 7.1: Übersicht über die Spezifikationsdaten der verwendeten Rechensysteme.

## 7.2 Definition der Anwendungsmodelle zu den elektroquasistatischen Problemen

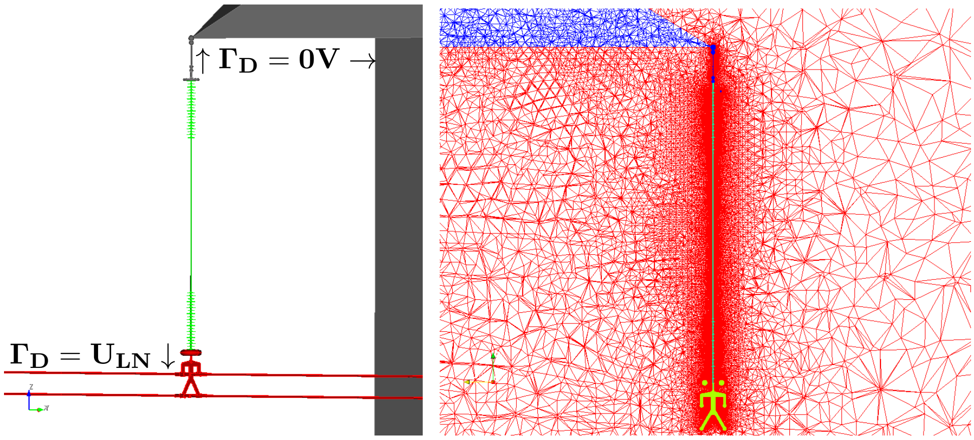
Es werden verschiedene Betriebsmittel der elektrischen Energieübertragungstechnik simuliert. Hierbei handelt es sich um Hochspannungsisolatoren, Hochspannungskabelendverschlüsse und Überspannungsableiter. Dabei kommen in diesen Anwendungen Mikrovaristoren als nichtlineares Feldsteuerelement zum Einsatz. Alle Modelle werden mit FEM-Ansatzfunktionen erster und zweiter Ordnung simuliert. Die Ansatzfunktion hat Einfluss auf die Anzahl der Unbekannten $n$ auf die Besetzungsdichte der Systemmatrix. Es wird insbesondere dieser Einfluss bei der Verwendung von GPUs dargestellt. Tab. 7.2 zeigt die Anzahl der Unbekannten $n$ der einzelnen Probleme, die Nicht-Null-Einträge $nnz$ der Systemmatrix, die Besetzungsdichte der Systemmatrix $\frac{nnz}{n}$ als Mittelwert der Einträge pro Zeile und ob es sich um ein Modell mit nichtlinearen Materialien handelt.

| Nr. | Ordnung Ansatzfunktionen | Unbekannte $n$ | Nicht-Null Einträge $nnz$ | $\frac{nnz}{n}$ | Nichtlineares Material |
|---|---|---|---|---|---|
| 1 | 1 | $1,47 \cdot 10^6$ | $21,8 \cdot 10^6$ | $14,78$ | nein |
| | 2 | $12,04 \cdot 10^6$ | $340,6 \cdot 10^6$ | $28,29$ | |
| 2 | 1 | $1,41 \cdot 10^6$ | $20,3 \cdot 10^6$ | $14,37$ | ja |
| | 2 | $11,63 \cdot 10^6$ | $320 \cdot 10^6$ | $27,52$ | |
| 3 | 1 | $2,92 \cdot 10^6$ | $43,3 \cdot 10^6$ | $14,83$ | ja |
| | 2 | $23,46 \cdot 10^6$ | $669,3 \cdot 10^6$ | $28,53$ | |
| 4 | 1 | $2,15 \cdot 10^6$ | $31,9 \cdot 10^6$ | $14,88$ | ja |
| | 2 | $17,34 \cdot 10^6$ | $494,7 \cdot 10^6$ | $28,53$ | |
| 5 | 1 | $2,04 \cdot 10^6$ | $30,4 \cdot 10^6$ | $14,90$ | ja |
| | 2 | $16,51 \cdot 10^6$ | $471,1 \cdot 10^6$ | $28,53$ | |
| 6 | 1 | $2,69 \cdot 10^6$ | $39,84 \cdot 10^6$ | $14,81$ | ja |
| | 2 | $21,53 \cdot 10^6$ | $614,1 \cdot 10^6$ | $28,52$ | |
| 7 | 1 | $2,65 \cdot 10^6$ | $39,33 \cdot 10^6$ | $14,84$ | ja |
| | 2 | $21,21 \cdot 10^6$ | $605,1 \cdot 10^6$ | $28,53$ | |

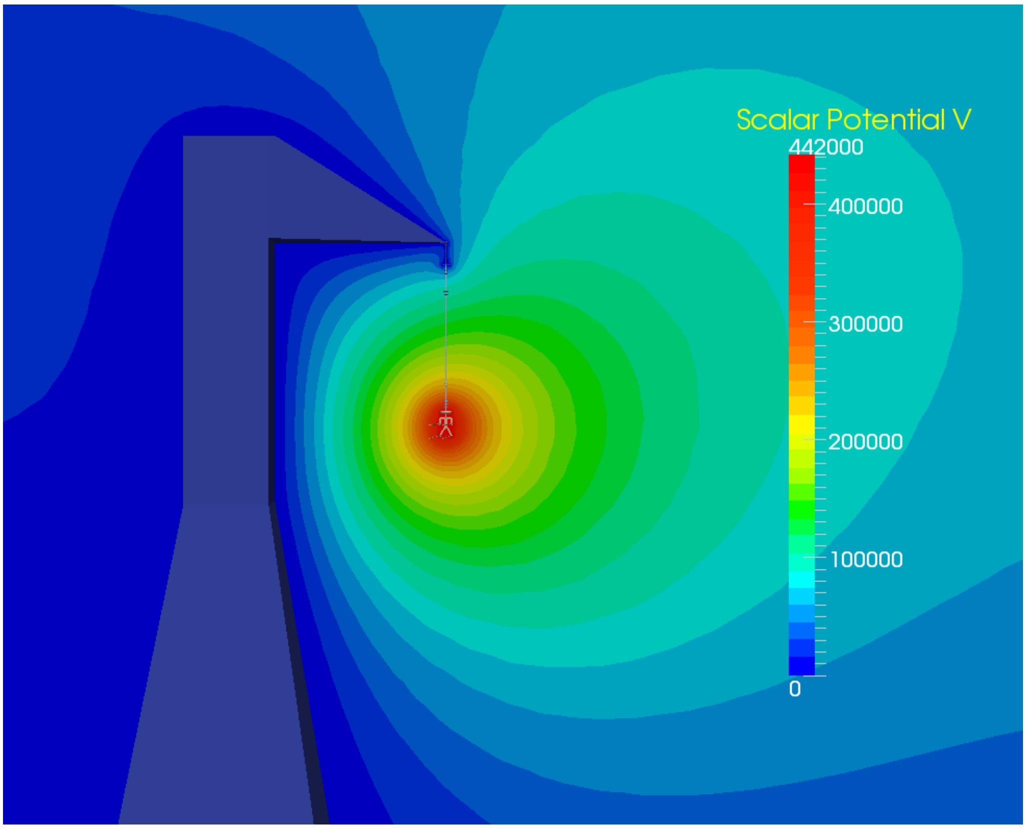
Tabelle 7.2: Übersicht über die Dimension der Beispielprobleme.

### 7.2.1 Konventioneller Höchstspannungsisolator

Bei dem Beispielproblem Nr. 1 handelt es sich um einen Höchstspannungslangstabisolator für 765 kV-Überlandleitungen mit einer Gesamtlänge von 7 m [3][B7]. Die Strecke zwischen spannungsführenden Teilen und geerdeter Aufhängung des Isolators beträgt 6 m. Dieser ist an einem Übertragungsmast 30 m über dem Boden befestigt. Am unteren Ende des Isolators ist



(a) CAD-Modell (mit Dirichleträndern $\Gamma_D$) und Gitternetz



(b) Potentialverteilung

Abbildung 7.1: Der konventionelle Höchstspannungslangstabisolator (Modell Nr. 1).

| Material | relative Permittivität $\varepsilon_r$ | spezifische Leitfähigkeit $\kappa$ in S/m |
|---|---|---|
| Luft | 1 | 0 |
| GFK | 4 | 0 |
| Silikon | 4 | 0 |

Tabelle 7.3: Materialparameter im Modell Nr. 1 - Konventionelle Höchstspannungslangstabisolator

eine Doppelleitung befestigt. Die Phasenspannung $U_{LN}$ beträgt 442 kV, die Spitzenspannung $\hat{U} = \sqrt{2}\ U_{LN} = 625$ kV. Der Mast sowie der Boden werden als geerdet betrachtet. Daher ist hier ein konstantes Potential, also eine Dirichlet-Randbedingung, von 0 V gesetzt. Die anderen Ränder des Rechenraumes sind als offene (homogene) Neumann-Ränder modelliert. In Abb. 7.1a sind ein Computer-Aided Design (CAD)-Model und das Gitternetz entlang des Isolators dargestellt. Die Potentialverteilung zu selbigem Modell ist in Abb. 7.1b dargestellt.

Um auch die Außenraumeinflüsse, etwa das unsymmetrisch erhöhte elektrische Feld zwischen Mast und Leiter, zu erfassen, ist ein ausreichend großen Außenraum mit einzubeziehen. In diesem Modell ist ein Aussenraum von 40×40 m gewählt. Der Effekt des Mastes auf die Potentialverteilung ist in Abb. 7.1b gut zu erkennen. Das resultierende diskrete FEM-Problem hat in diesem Beispiel einen Umfang von $n = 1,47$ Mio. Freiheitsgraden, wenn es mit Ansatzfunktionen erster Ordnung berechnet wird. Wird es weiter mit Ansatzfunktionen zweiter Ordnung berechnet, steigt die Anzahl der Freiheitgrade auf $n = 12$ Mio.

Leitfähige Teile, wie das Kabel oder der Mast, werden als perfekt elektrisch leitfähig angenommen und aus dem Modell entfernt. Das Potential wird als Dirichletrandbedingung auf ihren Oberflächen vorgegeben. Für die spannungsführenden Teile wird hier die in Kap. 6.2 beschriebene Spannungskurve eingeprägt. Die geerdeten Teile sind in Abb. 7.1a als blaues Gitter dargestellt.

Drei Materialen sind im Modell vorhanden. Die umgebende Luft (in Abb. 7.1a als rotes Gitter), der Isolatorstab, welcher aus glasfaserverstärktem Kunststoff (GFK) besteht und die Beschichtung des Stabes und der Silikonschirme (in Abb. 7.1a als grünes Gitter). Ihre Materialparameter sind in Tab. 7.3 dargestellt.

Das Modell wird im Folgenden durch das Einbringen von Mikrovaristoren zur Feldsteuerung um nichtlineare Materialien erweitert. Da es sich jedoch um ein Modell handelt, welches ausschließlich aus nicht leitfähigen Materialien besteht, kann das transiente Verhalten durch Skalierung des Ergebnisses dargestellt werden. Hierzu ist das Modell einmalig als elektrostatisches FEM-Problem zu lösen und kann dann durch Multiplikation des Ergebnisvektors mit $\frac{U_{Neu}}{U_{Alt}}$ skaliert werden.

### 7.2.2 Hochspannungsdurchführung mit Mikrovaristorfeldsteuerelement

Bei Beispielproblem Nr. 2 handelt es sich um eine Hochspannungsdurchführung, welche mit einer Effektivspannung von $U_{LN} = 100$ kV betrieben wird. Mit einer Durchführung wird ein Leiter zum Beispiel durch eine geerdete Wand geführt. Hierfür muss die elektrische Feldverteilung soweit gesteuert und homogenisiert werden, dass es während des Betriebes und bei Spannungsspitzen nicht zu Entladungen oder Durchschlägen kommt. Durchführungen werden vor allem bei Schaltanlagen, Transformatoren und Generatoren benötigt [1].

Um das elektrische Feld berechnen zu können, muss neben der Durchführung auch der Außenraum betrachtet werden. Hierzu gehören die geerdeten Außenwände, welche in Abb. 7.2 in gelb dargestellt sind, außerdem die geerdeten Metallrohre, welche die drei Leiter der drei Phasen umgeben. Diese sind ebenfalls geerdet.

Prinzipiell erfolgt die Steuerung des elektrischen Feldes bei diesem Modell – wie bei konventionellen Durchführungen üblich – statisch durch seine Geometrie [1] und wäre daher als elektrostatisches Problem berechenbar. Da zusätzlich jedoch ein nichtlineares Mikrovaristorsteuerelement eingebracht ist, welches die Potentialverteilung zusätzlich positiv beeinflussen soll, macht dies die elektroquasistatische Simulation erforderlich.

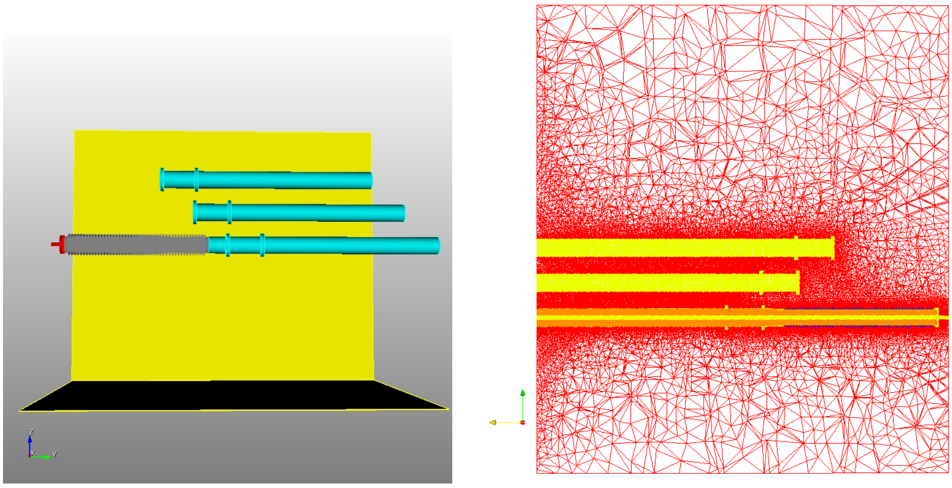


Abbildung 7.2: CAD-Modell und Gitter der Kabeldurchführung (Modell Nr. 2).

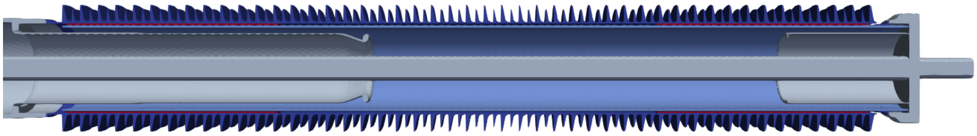


Abbildung 7.3: Schnitt durch das CAD-Modell der Kabeldurchführung (Modell Nr. 2). Mikrovaristormaterial in rot.

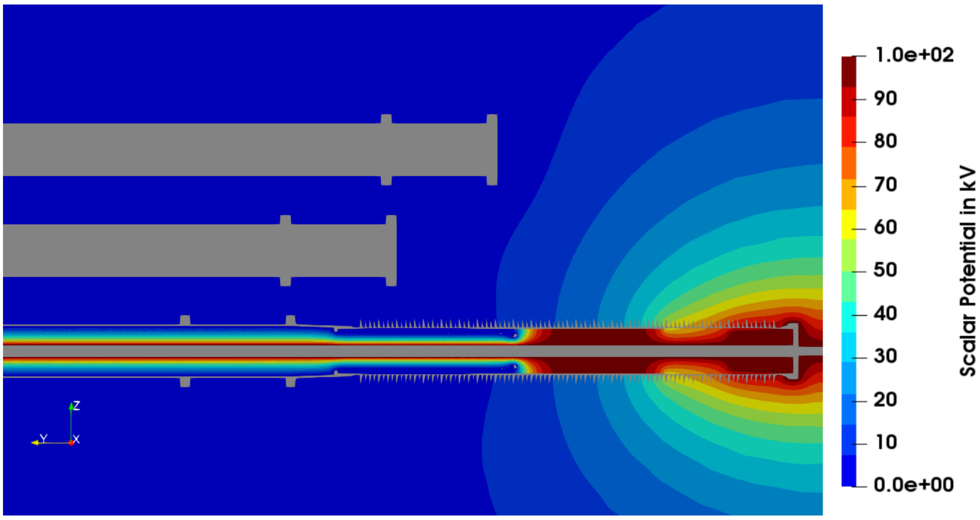


Abbildung 7.4: Potentialverteilung im Modell Nr. 2, der Kabeldurchführung.

| Material | relative Permittivität $\varepsilon_r$ | spezifische Leitfähigkeit $\kappa$ in S/m |
|---|---|---|
| Luft | 1 | 0 |
| Verbundkunststoff | 5 | 0 |
| Silikonschirme | 2,9 | 0 |
| $SF_6$-Gas | 1 | 0 |
| Mikrovaristormaterial | 12 | gem. Materialkurve Abb. 2.2 |

Tabelle 7.4: Materialparameter im Modell Nr. 2 - der Hochspannungskabeldurchführung

In Abb. 7.3 ist die Durchführung als Querschnitt zu sehen. Die Mikrovaristorelemente sind links und rechts als rote Streifen unter den Schirmen zu erkennen.

Das Modell besteht aus fünf Materialien, deren Materialparameter aus Tab. 7.4 entnommen werden können: Der umgebenden Luft (Abb. 7.2 als rotes Gitter), einem GFK-Verbundkunststoffkörper (Abb. 7.4 in Hellblau), den Silikonschirmen (Abb. 7.4 in Dunkelblau), der $SF_6$-Gasfüllung innerhalb des Verbundkunststoffkörpers und Mikrovaristoren als – in Abhängigkeit vom elektrischen Feld – nichtlinear leitfähiges Material (Abb. 7.4 in Rot an den Enden des GFK-Körpers).

Das Modell besteht aus $n_E = 8,6$ Mio. Tetraedern. Dies führt zu $n = 1,4$ Mio. Freiheitsgraden bei der Lösung mit Ansatzfunktionen erster Ordnung. Mit Ansatzfunktionen zweiter Ordnung besteht das zu lösende Problem aus $n = 11,6$ Mio. Freiheitgraden.

### 7.2.3 Höchstspannungsisolatoren mit Mikrovaristorfeldsteuerelement

Bei den Beispielproblemen Nr. 4 und 5 handelt es sich um das Grundmodell, welches bereits aus Problem Nr. 1 - Kap. 7.2.1 - bekannt ist. Im Rahmen der Evaluierung des Einsatzes von feldsteuernden Materialien ist in den Stabisolator eine Schicht aus Mikrovaristormaterial, welches in Kap. 2.2 vorgestellt wurde, eingebracht. Hierdurch soll eine Feldüberhöhung an kritischen

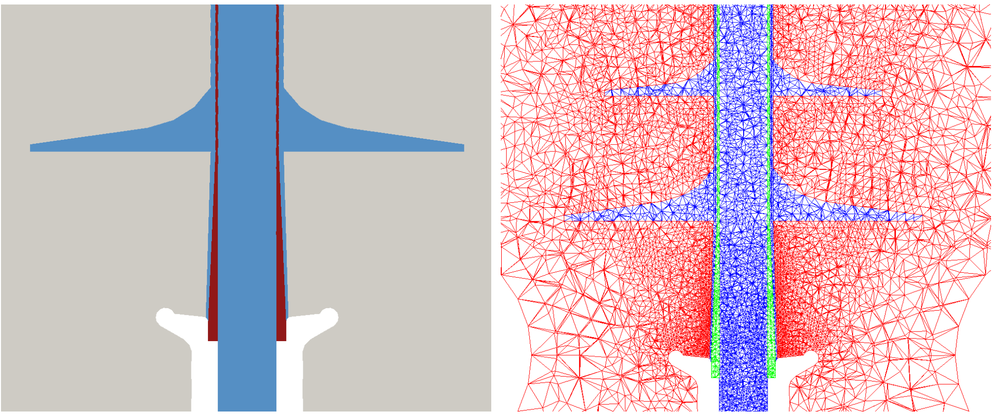


Abbildung 7.5: 2D-Schnitt und Gitternetz für das Modell Nr. 4, den Höchstspannungsisolator mit einer Mikrovaristorschicht von 1 mm (im linken Bild in rot).

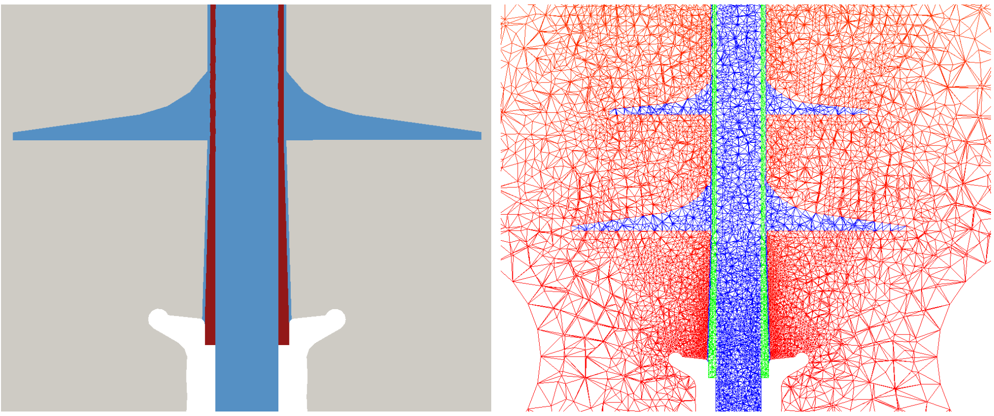


Abbildung 7.6: 2D-Schnitt und Gitternetz für das Modell Nr. 5, den Höchstspannungsisolator mit einer Mikrovaristorschicht von 2 mm (im linken Bild in rot).

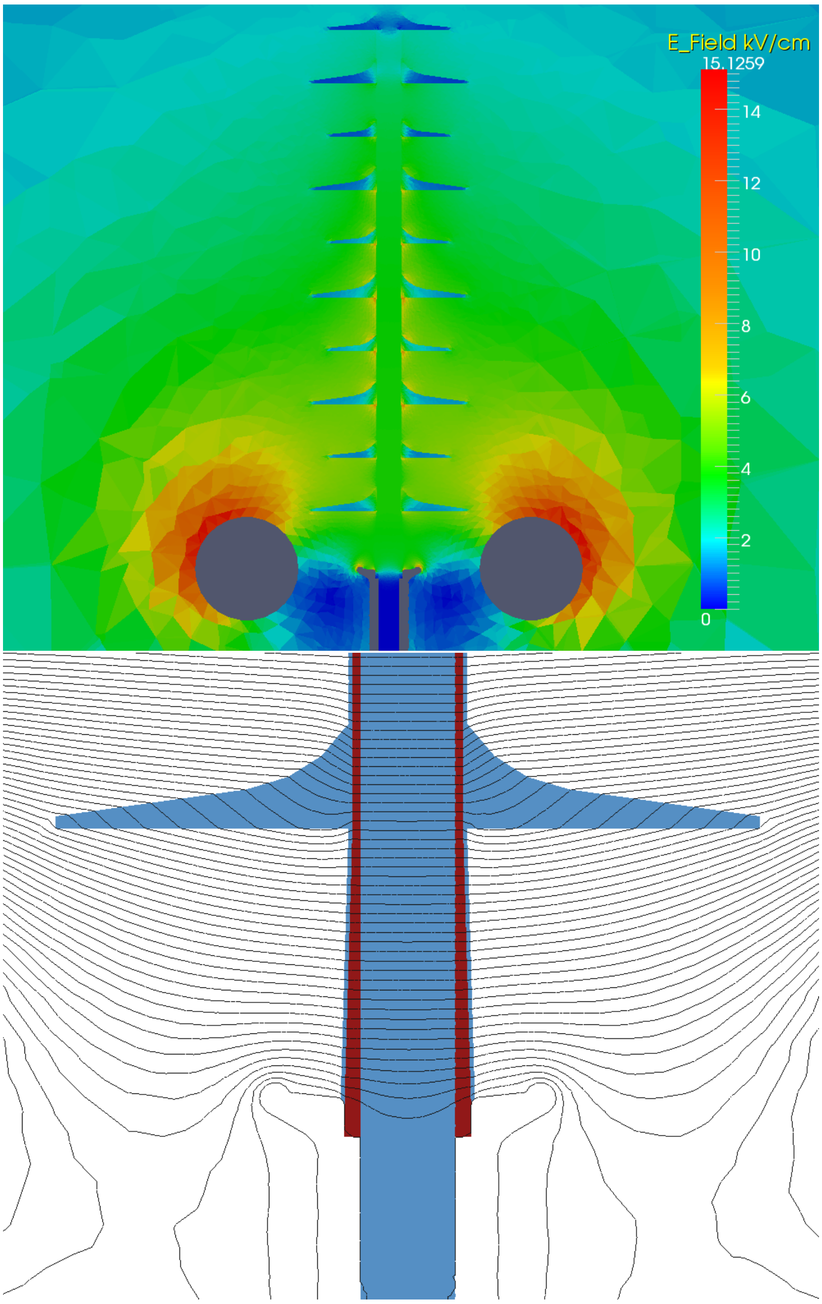


Abbildung 7.7: Elektrische Feldstärke und Äquipotentiallinien für das Modell Nr. 5, den Höchstspannungsisolator mit einer Mikrovaristorschicht von 2 mm.

Punkten abgeschwächt werden und ein Durchschlag verhindert werden [29, 3]. In Problem Nr. 4 ist diese Schicht nach der konischen Verjüngung von vier Millimeter einen Millimeter dick (vgl. Abb. 7.5) und in Problem Nr. 5 zwei Millimeter dick (vgl. Abb. 7.5) [3]. Zur Darstellung des Gesamtmodells und der makroskopischen Potentialverteilung sei an dieser Stelle auf Abb. 7.1a sowie Abb. 7.1b verwiesen.

Entscheidend ist hier nun, dass aufgrund der nichtlinearen Leitfähigkeit des Mikrovaristormaterials eine Simulation mit dem elektroquasistatischen Feldmodell erfolgt, um auch die durch Leitfähigkeit hervorgerufenen Effekte zu erfassen. Das Mikrovaristormaterial hat eine relative Permittivität von $\varepsilon_r = 12$. Die Leitfähigkeit ergibt sich feldstärkeabhängig aus der experimentell ermittelten Materialkurve aus Abb. 2.2. Die anderen Materialparameter sind identisch mit den in Tab. 7.3 genannten Werten.

### 7.2.4 Höchstspannungsisolatoren mit Mikrovaristorfeldsteuerelement und Wassertropfen

In Erweiterung zu den Beispielproblemen Nr. 4 und Nr. 5 wird in den Beispielproblemen Nr. 6 und Nr. 7 ein Stabisolator simuliert, welcher mit Wassertropfen benetzt ist. Ein Wassertropfen wird durch eine Halbkugel mit einem Durchmesser von 5,5 mm dargestellt. Die Feldsteuerung erfolgt auch hier über die 1 mm bzw. 2 mm dicke Mikrovaristorschicht. Durch die Wassertropfen kommt es zu einer Feldüberhöhung und elektrischen Entladungen am Tripelpunkt von Luft, Isolator und Wassertropfen. In [3] wurde bereits gezeigt, dass durch den Einsatz von Mikrovaristormaterialien die Feldstärke soweit homogenisiert werden kann, dass keine Überhöhung der tangentialen Feldstärke über dem Durchbruchsfeldstärkewert auftritt. Aufgrund der Größe der Wassertropfen ist hier ein sehr feines FEM-Gitter anzulegen, was zu einer Erhöhung der Anzahl der Unbekannten $n$ führt. Zusätzlich wird mit Wasser ein weiteres elektrisch leitfähiges Material in das Problem eingebracht. Alle Materialien sind noch einmal in Tab. 7.5 dargestellt.

Das Problem besteht aus $n_E = 16,2$ Mio. Tetraedern für das Modell mit

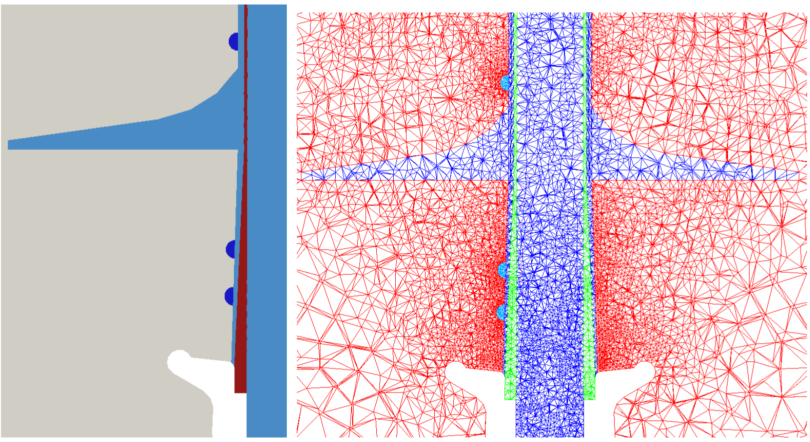


Abbildung 7.8: 2D-Schnitt und Mesh für das Modell Nr. 6, den Höchstspannungsisolator mit einer Mikrovaristorschicht von 1 mm und Wassertropfen (im linken Bild in rot).

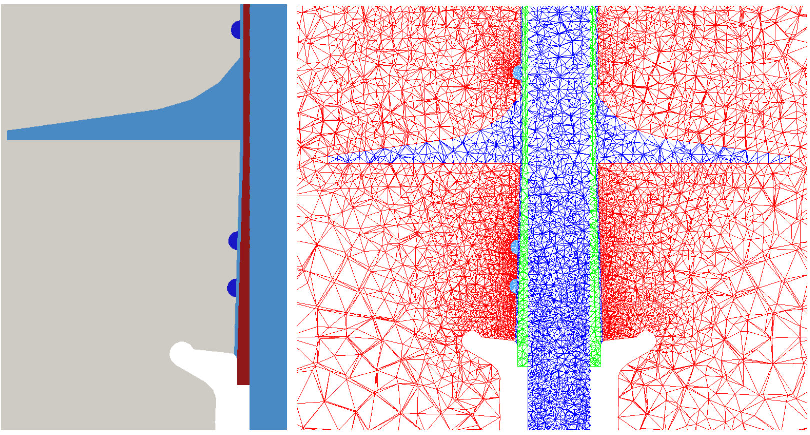


Abbildung 7.9: 2D-Schnitt und Mesh für das Modell Nr. 7, den Höchstspannungsisolator mit einer Mikrovaristorschicht von 2 mm und Wassertropfen (im linken Bild in rot).

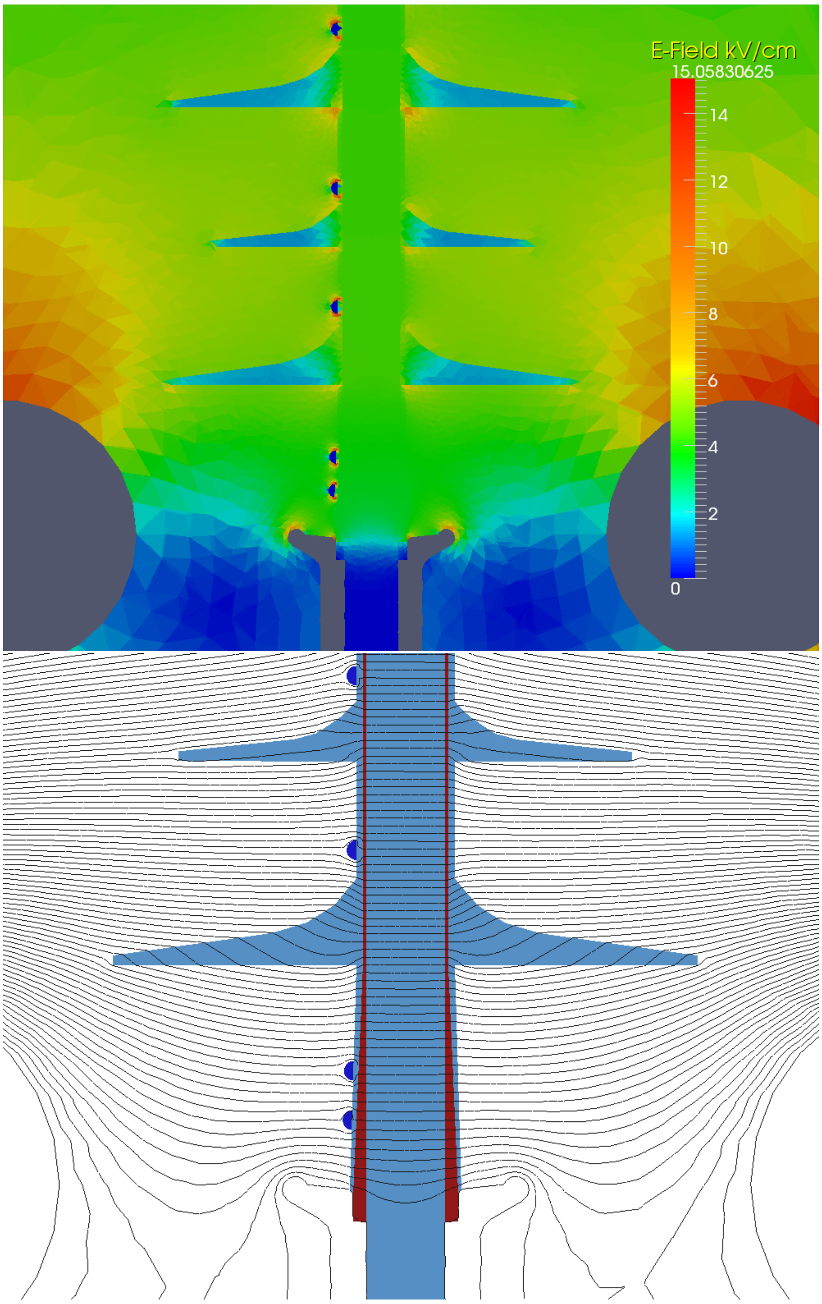


Abbildung 7.10: Elektrische Feldstärke und Äquipotentiallinien für das Modell Nr. 6, den Höchstspannungsisolator mit einer Mikrovaristorschicht von 1 mm und Wassertropfen.

| Material | relative Permittivität $\varepsilon_r$ | spezifische Leitfähigkeit $\kappa$ in S/m |
|---|---|---|
| Luft | 1 | 0 |
| GFK | 4 | 0 |
| Silikon | 4 | 0 |
| Wasser | 80 | $10^{-6}$ |
| Mikrovaristor-material | 12 | gem. Materialkurve Abb. 2.2 |

Tabelle 7.5: Materialparameter für die Modelle Nr. 3 bis Nr. 7

der 1 mm dicken Mikrovaristorschicht und $n_E = 15,96$ Mio Tetraedern für das Modell mit der 2 mm dicken Mikrovaristorschicht. Dies ergibt eine Summe von $n = 2,69$ Mio. bzw. $n = 2,65$ Mio. Unbekannten bei Ansatzfunktionen erster Ordnung und $n = 21,53$ Mio. bzw. $n = 21,21$ Mio. Unbekannten $n$ bei der Verwendung von Ansatzfunktionen zweiter Ordnung.

### 7.2.5 Höchstspannungsisolator mit Mikrovaristorfeldsteuerelement und Wassertropfen in erhöhter Auflösung

Das Modellproblem Nr. 3 ist das Problem mit den meisten Unbekannten, welches in dieser Arbeit betrachtet wird. Der Ausgangsstabisolator wird noch einmal deutlich feiner aufgelöst und mit einer 2 mm dicken Mikrovaristorschicht, sowie einzelnen Wassertropfen versehen. Das Modell stellt eine modifizierte Version des Modellproblems Nr. 7 dar. Es ist an den kritischen Stellen im Umfeld der aufgebrachten Tropfen noch weiter verfeinert. Mit dem Modell wird die maximale Dimension getestet, welches mit den zur Verfügung stehenden Streamingprozessoren und deren Speicher zu berechnen ist.

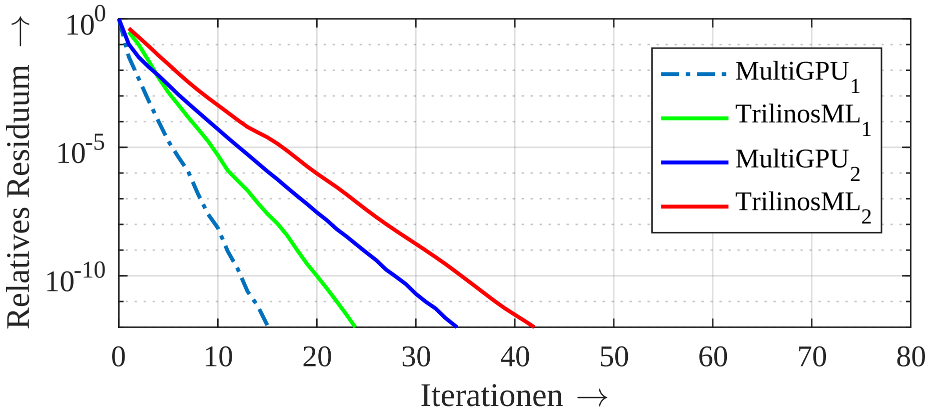


Abbildung 7.11: Konvergenzverhalten des Modells Nr.1 mit Ansatzfunktionen erster und zweiter Ordnung für den Multi-GPU-AMG-Vorkonditionierer und ein PCG-Verfahren mit Trilinos ML SA-AMG-Vorkonditionierer.

## 7.3 Vergleich des Multi-GPU-PCG-Lösers für lineare Gleichungssysteme

Zur Validierung der Ergebnisse werden die in Kap. 7.2 definierten Probleme mit dem in Kap. 5.1.3.4 vorgestellten Multi-GPU-AMG-PCG-Verfahren gelöst. In der Auswertung wird die Zeit betrachtet, welche benötigt wird, um das lineare Gleichungssystem zu lösen. Als Vergleichswert im Hinblick auf Geschwindigkeitsgewinne werden die erreichten Werte dem Standardsolver für lineare Gleichungssysteme des MEQSICO-Codes gegenübergestellt. Hierbei handelt es sich um einen PCG-Löser mit AMG-Vorkonditionierer der Bibliothek „Trilinos ML“ [101]. Der Trilinos ML-Löser ist unter Nutzung von OpenMP [60] voll parallelisiert und nutzt somit alle CPU-Rechenkerne des Hostsystems.

### 7.3.1 Konvergenzverhalten des Gleichungssystemlösers

Abb. 7.11 zeigt das Konvergenzverhalten der PCG-Löser für Beispielproblem Nr. 1 – „konventioneller Höchstspannungsisolator“. Es ist das Konvergenzverhalten des PCG-Lösers bei Nutzung von Ansatzfunktionen erster und zweiter

Ordnung für den Multi-GPU-Solver sowie den Trilinos ML-Vorkonditionierer dargestellt. Der Legendenindex gibt die Ordnung der Ansatzfunktionen an. Zum Zwecke der Vergleichbarkeit sind alle Multi-GPU-Darstellungen für Ansatzfunktionen erster Ordnung unter Verwendung von zwei GPUs berechnet. Für Ansatzfunktionen zweiter Ordnung wurden dazu jeweils vier GPUs verwendet. Das Abbruchkriterium für die Iteration ist in allen Fällen ein relatives Residuum

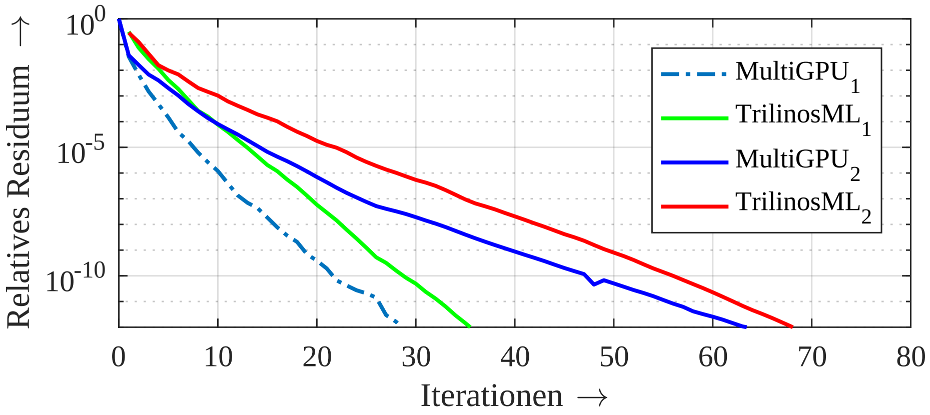
$$r_{rel} = \frac{||\mathbf{r}||}{||\mathbf{b}||} = \frac{||\mathbf{b} - \mathbf{Au}||}{||\mathbf{b}||} < \epsilon \, = \, 10^{-12}. \tag{7.1}$$

Diese hohen Genauigkeitsforderung ist aufgrund der Beziehung

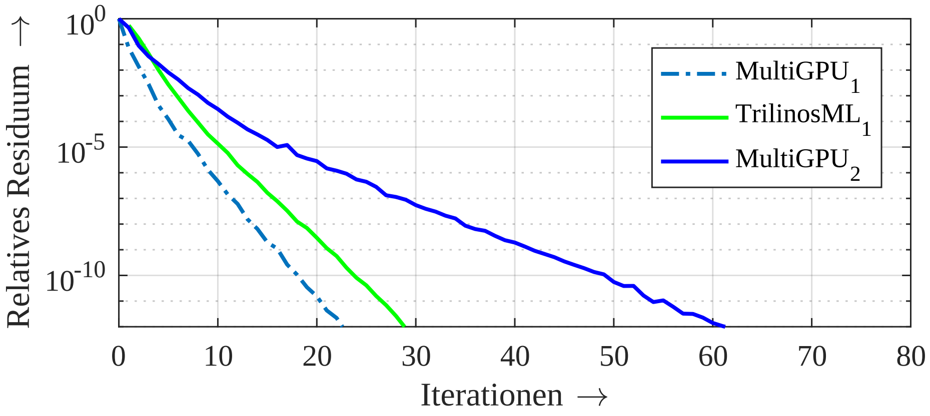
$$\frac{||\mathbf{e}_K||}{||\mathbf{e}_0||} \leq \varkappa(\mathbf{A}) \, \frac{||\mathbf{r}_K||}{||\mathbf{r}_0||} \tag{7.2}$$

erforderlich. Die betrachteten Probleme haben hohe Konditionszahlen $\varkappa(\mathbf{A})$, welche durch die kleine Größe der kleinsten Kantenlänge des FEM-Gitters von unter 1 mm entstehen (vgl. Kap. 4.1). Daher muss die Genauigkeitsforderung $\epsilon$ entsprechend hoch sein, um eine hinreichende Genauigkeit der Lösung zu erzielen.

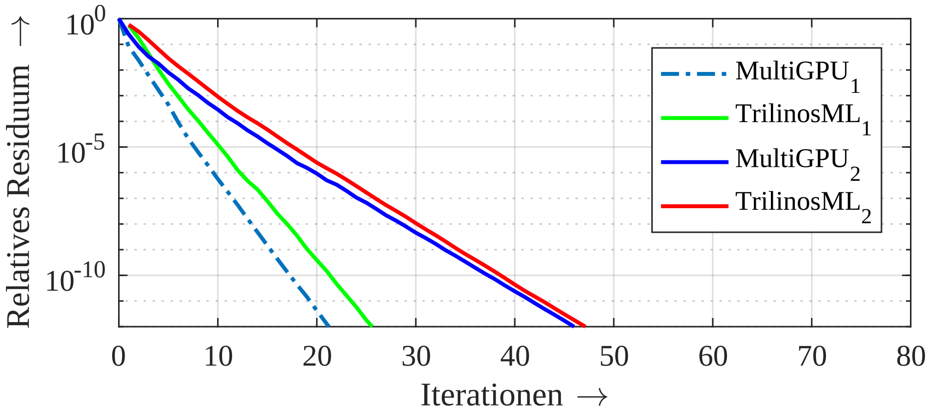
Aus Abb. 7.11 wird ersichtlich, dass der Multi-GPU-Ansatz das Residuum effektiver reduziert als der Trilinos ML PCG-Löser. In dieser Deutlichkeit ist dies jedoch nur bei Modell Nr. 1 der Fall. Auch wenn die Grundaussage der effektiveren Reduktion für alle Modellprobleme gilt, zeigen insbesondere Abb. 7.12a und Abb. 7.12c bei Ansatzfunktionen zweiter Ordnung einen geringeren Unterschied. Die Steigung des Multi-GPU-PCG-Lösers ist geringer, sodass sich die Kurven in der Verlängerung schneiden würden. In Abb. 7.13a tritt dieser Schnittpunkt bereits bei einem relativen Residuum von $r_{rel} \, = \, 10^{-11}$ auf. Die unterschiedliche Konvergenz der PCG-Verfahren ergibt sich aus dem unterschiedlichen Algorithmus zum Aufstellen des Vorkonditionierers. Trilinos ML wurde soweit möglich gleich dem in Kap. 4.2.1 vorgestellten Algorithmus konfiguriert. Dennoch ergeben sich Unterschiede, beispielsweise beim Aggregationsalgorithmus, welche die unterschiedliche Konvergenz begründen. Diese ist jedoch nicht auf den Einsatz von GPUs zurückzuführen.



(a) Modell Nr. 2 - Hochspannungsdurchführung.

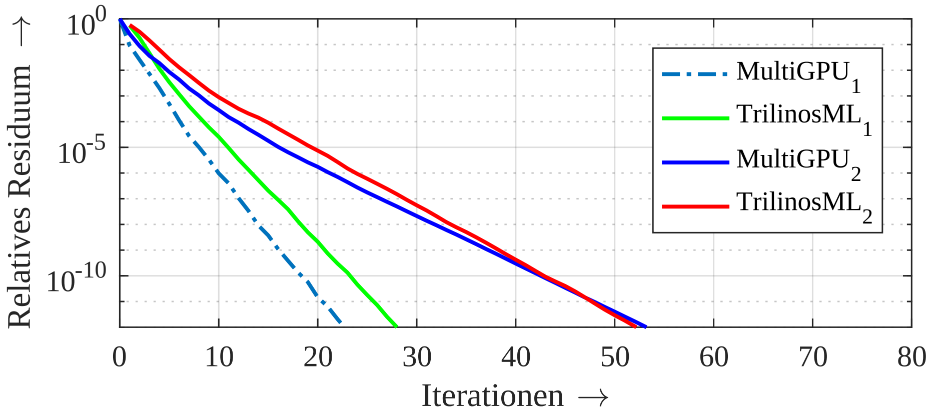


(b) Modell Nr. 3 - Langstabisolator mit maximaler Unbekanntenzahl.

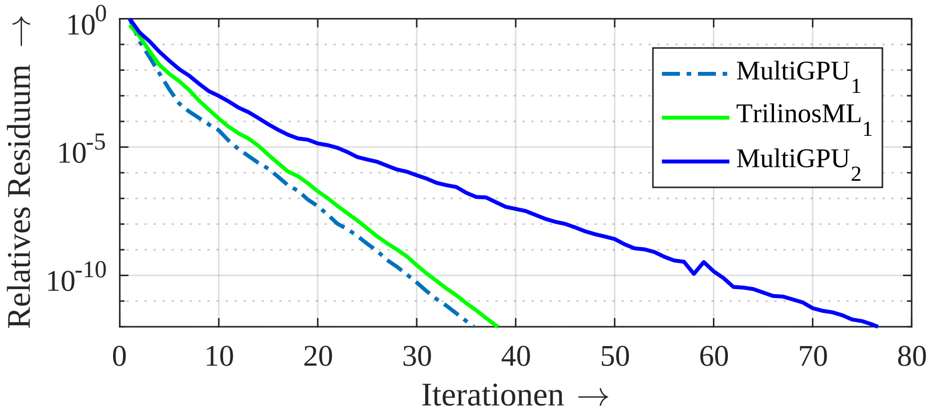


(c) Modell Nr. 4 - Langstabisolator mit 1 mm Mikrovaristorsteuerelement.

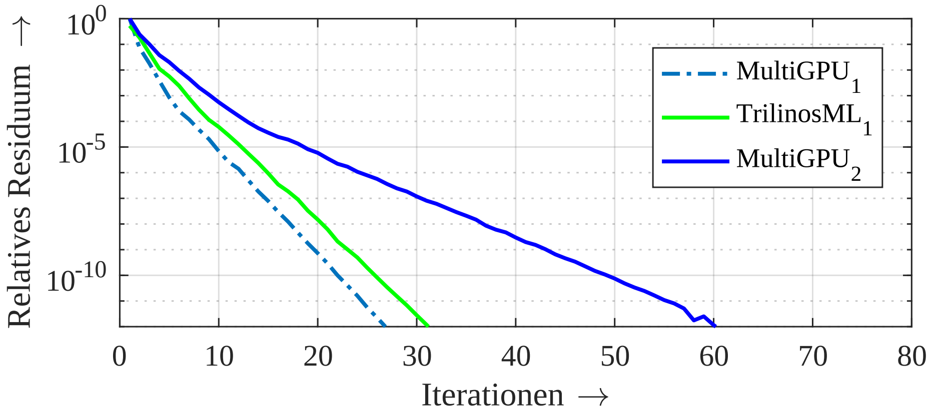
Abbildung 7.12: Konvergenzverhalten der Modelle Nr. 2 bis Nr. 4 mit Ansatzfunktionen erster und zweiter Ordnung für den Multi-GPU-AMG-Vorkonditionierer und ein CPU basiertes AMG-PCG-Verfahren mit Trilinos ML.



(a) Modell Nr. 5 - Langstabisolator mit 2 mm Mikrovaristorsteuerelement.



(b) Modell Nr. 6 - Langstabisolator mit 1 mm Feldsteuerelement und Tropfen.



(c) Modell Nr. 7 - Langstabisolator mit 2 mm Feldsteuerelement und Tropfen.

Abbildung 7.13: Konvergenzverhalten der Modelle Nr. 5 bis Nr. 7 mit Ansatzfunktionen erster und zweiter Ordnung für den Multi-GPU-AMG-Vorkonditionierer und ein CPU basiertes AMG-PCG-Verfahren mit Trilinos ML.

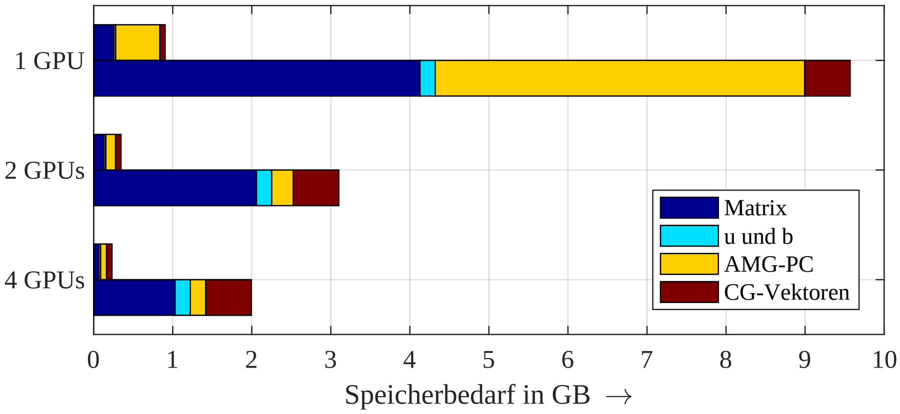


Abbildung 7.14: Speicherbedarf pro GPU des Modells Nr. 1 mit Ansatzfunktionen erster und zweiter Ordnung. „Matrix“ bezeichnet die Systemmatrix $\mathbf{A}$ und „$\mathbf{u}$ und $\mathbf{b}$“ die Vektoren des zu lösenden Gleichungssystems $\mathbf{Au} = \mathbf{b}$. „AMG-PC“ bezeichnet den AMG-Vorkonditionierer und „CG-Vektoren“ die intern Verwendeten Vektoren des PCG-Verfahrens.

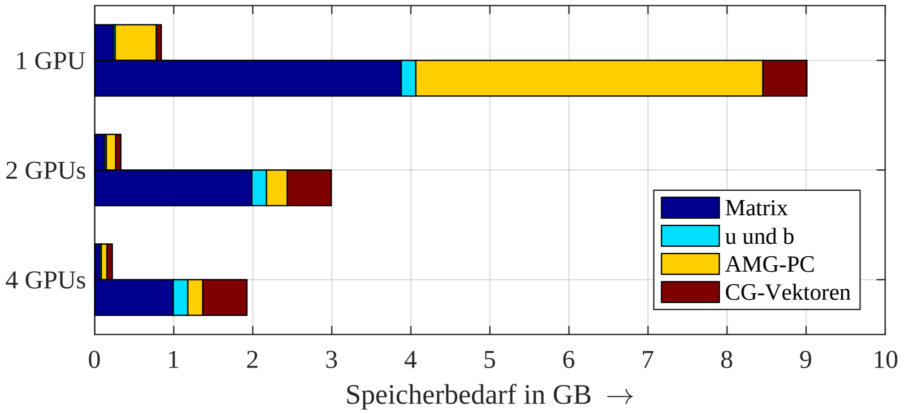
### 7.3.2 Speicherbedarf

In Abb. 7.14 ist für den „konventionellen Höchstspannungsisolator“ der Speicherbedarf dargestellt, welche auf pro GPU beim Lösen des LGS belegt wird. Diese ist unterteilt in die Systemmatrix $\mathbf{A}$, die Rechte-Seite- und Lösungsvektoren $\mathbf{b}, \mathbf{u}$, den AMG-Vorkonditionierer, sowie die Hilfsvektoren in der PCG-Iteration. Ferner ist der Graph vertikal nach der Anzahl der verwendeten GPUs unterteilt. Weiterhin ist jeder GPU-Bereich in zwei Balkendiagramme unterteilt, welche den benötigten Speicherplatz je GPU bei Verwendung von Ansatzfunktionen erster Ordnung (oberer Balken), sowie zweiter Ordnung (unterer Balken) angegeben. Zur Ermittlung des benötigten Speicherplatzes wurden eine sowie zwei GPUs der Nvidia Tesla K80 des Systems Nr. 2 mit der verwendet. Für die Angaben mit vier GPUs wurde das System Nr. 1 mit den Nvidia Tesla K20m Grafikkarten benutzt.

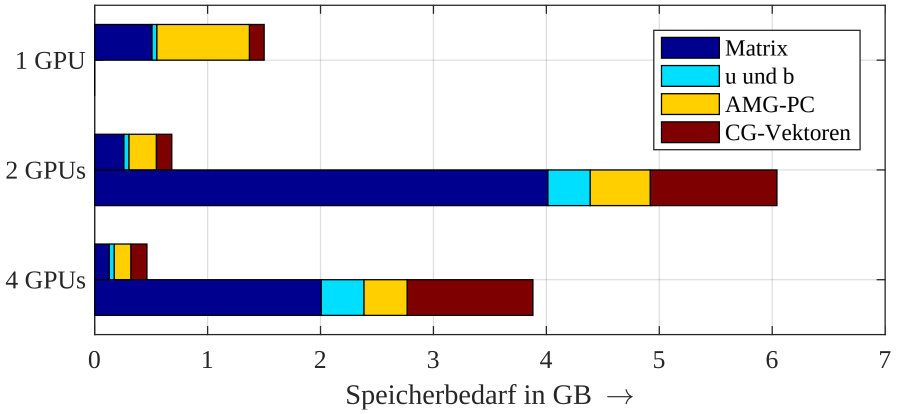
In Abb. 7.14 ist ersichtlich, dass bei Verwendung von Ansatzfunktionen zweiter Ordnung ein wesentlich höherer Speicherbedarf besteht als dies bei Ansatzfunktionen erster Ordnung der Fall ist. Der Mehrbedarf an Speicherplatz ist insbesondere bei den Matrizen klar zu erkennen. Als Gründe sind hier höhere Anzahl an Freiheitsgraden, aber auch die höhere Besetzungsdichte der Matrix zu nennen, welche auch Tab. 7.2 zu entnehmen sind. Bei steigender Anzahl an GPUs ist eine lineare Abnahme des Speicherbedarfs für die Matrix zu erkennen. Dies zeigt den Effekt der Aufteilung in gleich große Teilmatrizen (vgl. Kap. 5.1.3.1). Der Platzbedarf der Vektoren ist auf allen GPUs gleich groß, da diese auf jeder GPU vollständig vorhanden sein müssen. Die Abbildung 7.14 verdeutlicht den geringen Platzbedarf der Vektoren gegenüber der Matrix und macht deutlich, warum eine gleichmäßige Aufteilung der Matrix von entscheidender Bedeutung ist (vgl. Kap. 5.1.3.1).

Weiterhin ist erkennbar, dass der Vorkonditionierer bei Verwendung einer GPU – also bei Verwendung der unmodifizierten CUSP-Bibliothek – bzgl. des Speicherbedarfs größer ist, als dies bei den Multi-GPU-Implementationen der Fall ist. In der unmodifizierten CUSP-Bibliothek wird die Systemmatrix auch als unterste Ebene des AMG-Vorkonditionierers gespeichert. Diese Redundanz wurde für die in dieser Arbeit entwickelte Multi-GPU-Implementierung entfernt (vgl. Kap. 5.1). Der Speichergewinn hierdurch ist in Abb. 7.14 beim Vergleich der Balken für den Speicherbedarf von einer GPU und von mehreren GPUs ersichtlich. Ein wesentlicher Implementierungsgrund für den Multi-GPU Ansatz ist – neben der Beschleunigung des PCG-Lösers – Beschränkungen beim zur Verfügung stehenden Arbeitsspeicher ohne Geschwindigkeitsverlust zu umgehen. So ist keines der betrachteten Beispielprobleme auf einer einzigen Nvidia Tesla K20m mit Ansatzfunktionen zweiter Ordnung berechenbar, da sie jeweils den verfügbaren Speicherplatz von 5 GB übersteigen. Ihr Speicherbedarf ist in Abb. 7.15 und 7.16 dargestellt. Die in Abb. 7.15b bis 7.16a dargestellten Probleme sind auch auf einer GPU der Nvidia Tesla K80 nicht zu berechnen, daher sind hier keine Balken für diesen Fall angegeben.

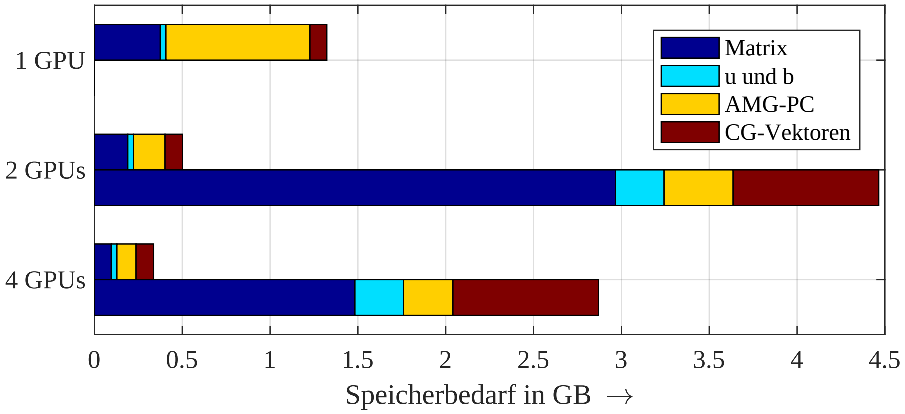
Der Vergleich des Speicherbedarfs der Beispielprobleme zeigt, dass dieser linear mit der Anzahl der Freiheitsgrade skaliert. Dies ist besonders gut am Platzbedarf der Teilmatrix in den Abb. 7.15a und 7.15b zu erkennen. Bei einer Anzahl an Unbekannten von 11,5 Mio. gegenüber 23,4 Mio. verdoppelt sich auch der Speicherbedarf für die Matrix. Dieser Umstand



(a) Modell Nr. 2 - Hochspannungsdurchführung.

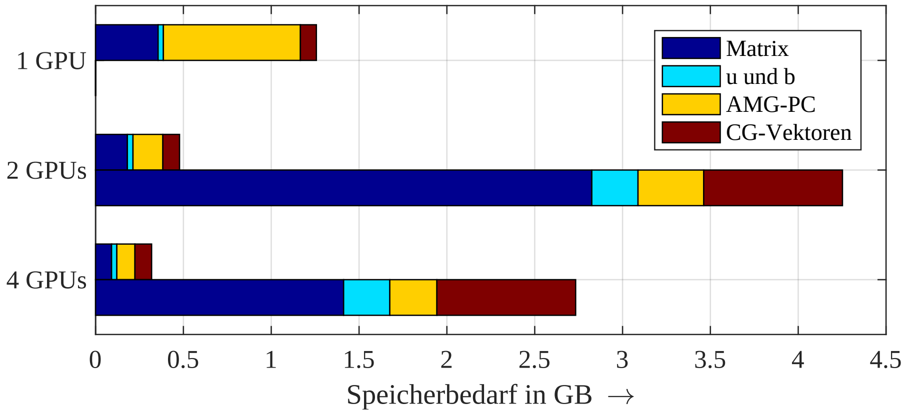


(b) Modell Nr. 3 - Langstabisolator mit maximaler Unbekanntenzahl.

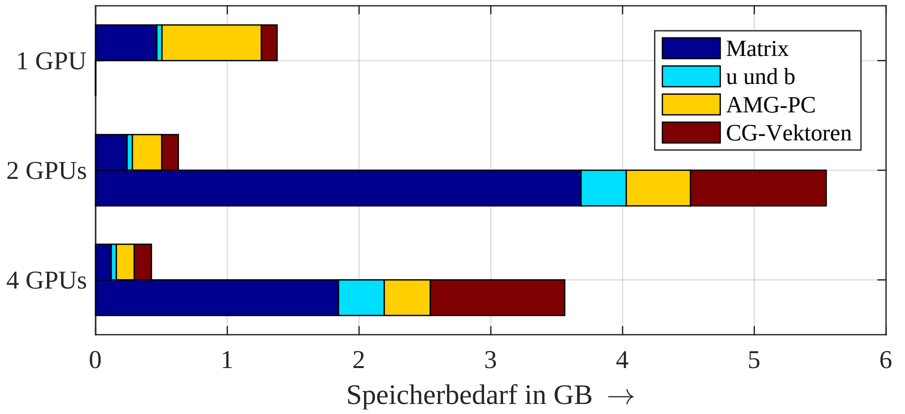


(c) Modell Nr. 4 - Langstabisolator mit 1 mm Mikrovaristorsteuerelement.

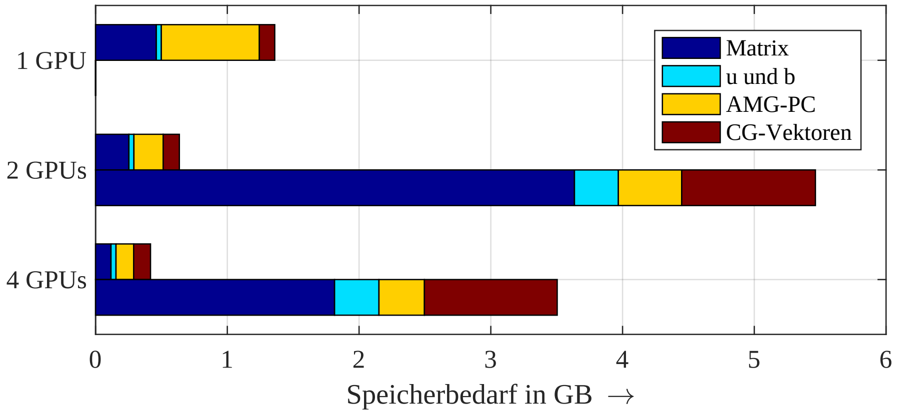
Abbildung 7.15: Speicherbedarf pro GPU der Modelle Nr. 2 bis Nr. 4 mit Ansatzfunktionen erster und zweiter Ordnung. „Matrix“ bezeichnet die Systemmatrix $\mathbf{A}$ und „$\mathbf{u}$ und $\mathbf{b}$“ die Vektoren des zu lösenden Gleichungssystems $\mathbf{Au} = \mathbf{b}$. „AMG-PC“ bezeichnet den AMG-Vorkonditionierer und „CG-Vektoren“ die intern Verwendeten Vektoren des PCG-Verfahrens.



(a) Modell Nr. 5 - Langstabisolator mit 2 mm Mikrovaristorsteuerelement.



(b) Modell Nr. 6 - Langstabisolator mit 1 mm Feldsteuerelement und Tropfen.



(c) Modell Nr. 7 - Langstabisolator mit 2 mm Feldsteuerelement und Tropfen.

Abbildung 7.16: Speicherbedarf pro GPU der Modelle Nr. 5 bis Nr. 7 mit Ansatzfunktionen erster und zweiter Ordnung. „Matrix“ bezeichnet die Systemmatrix **A** und „**u** und **b**“ die Vektoren des zu lösenden Gleichungssystems $\mathbf{Au} = \mathbf{b}$. „AMG-PC“ bezeichnet den AMG-Vorkonditionierer und „CG-Vektoren“ die intern Verwendeten Vektoren des PCG-Verfahrens.

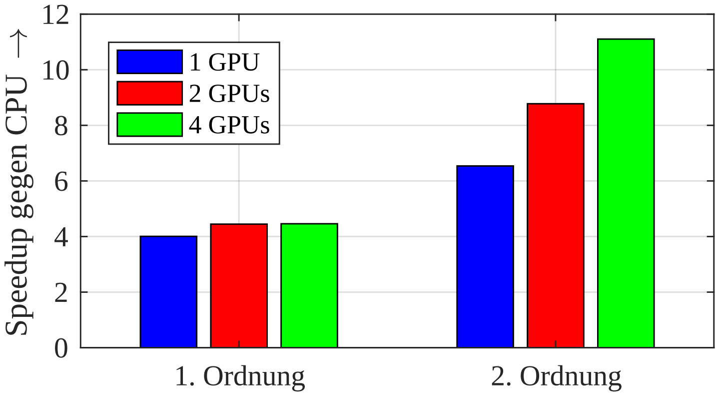


Abbildung 7.17: Speedup der Lösung des LGS des elektrostatischen Modells „konventioneller Höchstspannungsisolator“ (Modell Nr.1) mittels PCG-Verfahren mit Ansatzfunktionen erster und zweiter Ordnung für den Multi-GPU-AMG-Vorkonditionierer gegenüber einer CPU-basierten Version mit Trilinos ML.

ist auf alle untersuchten Beispiele übertragbar und entspricht damit den Erwartungen (vgl. Kap. 2.3).

Festzuhalten ist allerdings auch, dass eine GPU grundsätzlich einen gewissen Anteil an zusätzlichem Speicherplatz benötigt. So ist das Problem Nr. 3 auf System Nr. 1 mit vier Tesla K20m nur zu rechnen, wenn sämtliche anderen Zugriffe auf die GPUs unterbunden werden. Ein deutlicher Pufferraum von ca. 25 % des verfügbaren globalen Speichers sollte daher grundsätzlich eingerechnet werden.

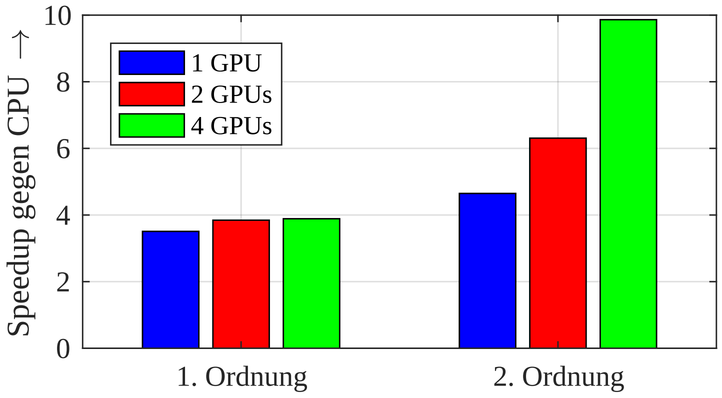
### 7.3.3 Rechengeschwindigkeit

Hauptgrund des Einsatzes von GPUs ist die Realisierung eines Geschwindigkeitsgewinns. Dieser wird für den Multi-GPU-Solver im Folgenden untersucht. Hierzu ist in Abb. 7.17 bis 7.19 der Geschwindigkeitsgewinn gegenüber dem CPU-basierten PCG-Verfahren mit Trilinos ML AMG-Vorkonditionierer angegeben.

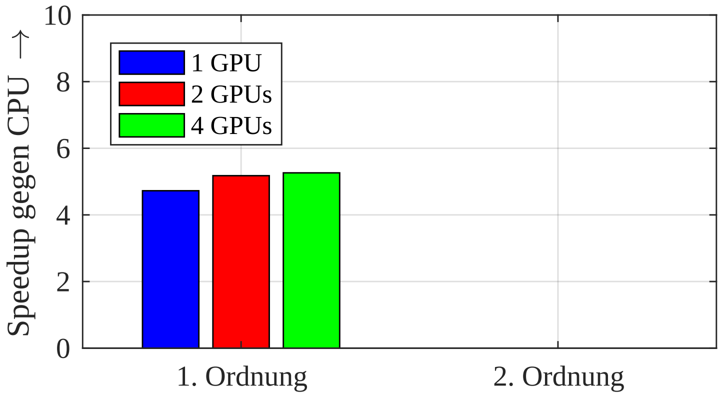
Der Geschwindigkeitsgewinn wird für Ansatzfunktionen erster und zweiter Ordnung untersucht. Dabei wird das LGS des Modellproblems jeweils mit der Trilinos ML-Implementierung, sowie dem Multi-GPU Solver mit einer, zwei und vier GPUs gelöst. Bei Nutzung einer GPU entspricht der AMG-PCG-Löser der unveränderten CUSP-Implementierung für GPUs [90]. Um verlässlich Geschwindigkeiten messen zu können, ist es wesentliche Voraussetzung, dasselbe Rechensystem zu nutzen. Daher wurden alle Rechnungen auf dem System Nr. 1 mit Tesla K20m GPUs gerechnet. In Abb. 7.18b sind keine Ergebnisse zweiter Ordnung dargestellt, da die Trilinos ML Referenzlösung nicht berechnet werden kann. In Abb. 7.18c und 7.19a ist das Modellproblem mit Ansatzfunktionen zweiter Ordnung nur unter Nutzung von vier Tesla K20m lösbar. Daher ist auch nur dieses Ergebnis dargestellt. Der Geschwindigkeitsgewinn ist bei allen untersuchten Fallbeispielen ähnlich.

Im Folgenden werden die in Abb. 7.17 dargestellten Ergebnisse diskutiert. Mit einer Tesla K20m lässt sich eine Beschleunigung der Rechnung um den Faktor 4 gegenüber der CPU-gestützten Lösung bei Ansatzfunktionen erster Ordnung erreichen. Bei Verwendung von zwei GPUs wird dieser Wert noch einmal um 10 % gesteigert, um dann bei der Verwendung von 4 GPUs mit einem Zugewinn von unter 1 % zu stagnieren. Hier ist ersichtlich, dass der Zugewinn an Rechenleistung und lokaler Speicherbandbreite durch die Kommunikation zwischen den Grafikkarten aufgezehrt wird.

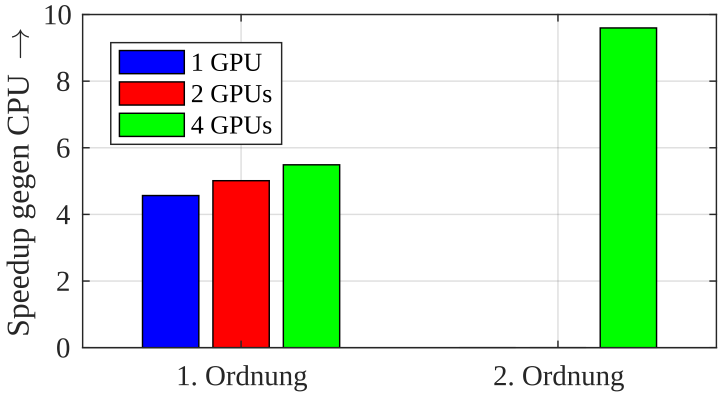
Bei der Lösung von Problemen zweiter Ordnung ist ein Geschwindigkeitsgewinn um den Faktor $6,4$ zu beobachten. Hier resultiert der Gewinn aus dem etwa doppelt so hohen Besetzungsgrad der Matrix (vgl. Tab. 7.2). Dadurch können mehr verwertbare Daten von globalen Speicher an die ALUs übertragen werden, wie in Kap. 3 erläutert wurde. Die effektive Bandbreite steigt und es wird so eine höhere Beschleunigung der Berechnung erreicht. Aufgrund des höheren Besetzungsgrades der Matrix steigt der Anteil an Berechnung relativ zur Kommunikation zwischen den GPUs. Verglichen mit den Ansatzfunktionen erster Ordnung kann hier bei Verwendung mehrerer GPUs noch einmal ein deutlicher Geschwindigkeitsgewinn verzeichnet werden. Dieser steigt auf einen Faktor von $7,8$ für zwei GPUs und bis zu einem Faktor von $11,1$ bei vier GPUs.



(a) Modell Nr. 2 - Hochspannungsdurchführung.

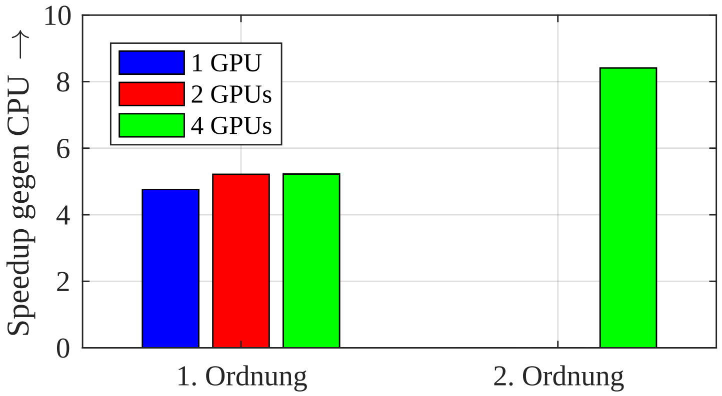


(b) Modell Nr. 3 - Langstabisolator mit maximaler Unbekanntenzahl.

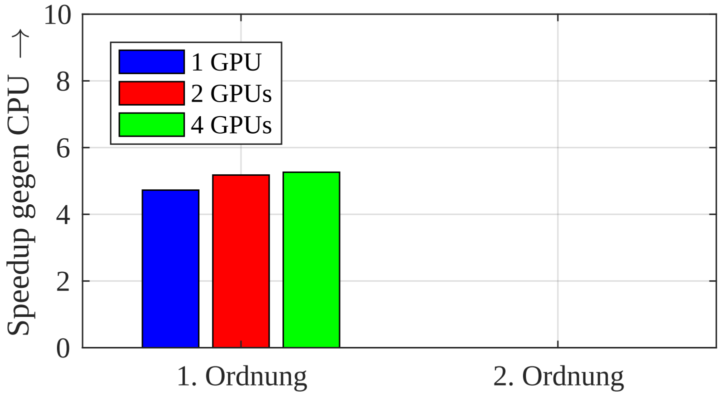


(c) Modell Nr. 4 - Langstabisolator mit 1 mm Mikrovaristorsteuerelement.

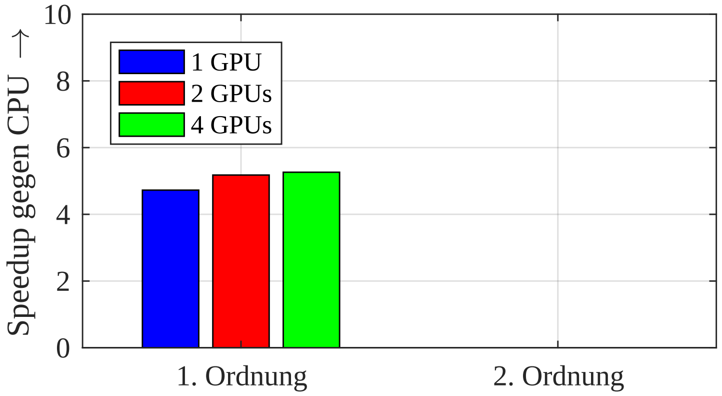
Abbildung 7.18: Speedup der Lösung des LGS für die Modell Nr. 2 bis Nr. 7 mit Ansatzfunktionen erster und zweiter Ordnung für den Multi-GPU-AMG-Vorkonditionierer gegenüber Trilinos ML.



(a) Modell Nr. 5 - Langstabisolator mit 2 mm Mikrovaristorsteuerelement.



(b) Modell Nr. 6 - Langstabisolator mit 1 mm Feldsteuerelement und Tropfen.



(c) Modell Nr. 7 - Langstabisolator mit 2 mm Feldsteuerelement und Tropfen.

Abbildung 7.19: Speedup der Lösung des LGS für die Modell Nr. 2 bis Nr. 7 mit Ansatzfunktionen erster und zweiter Ordnung für den Multi-GPU-AMG-Vorkonditionierer gegenüber Trilinos ML.

Für die Anwendungsbeispiele Nr. 2 bis Nr. 7 zeigen sich qualitativ vergleichbare Ergebnisse. Eine Beschleunigung um etwa Faktor 4 bei Ansatzfunktionen erster Ordnung ist bei allen in Abb. 7.19 dargestellten Problemen festzustellen. Daraus folgt insbesondere, dass der Speedup nicht von der Größe des Problems abhängt, sondern vom der Anzahl der Nicht-Null-Einträge *nnz* der Systemmatrix. Abb. 7.18a zeigt noch einmal das Ansteigen des Beschleunigungsfaktors mit der Anzahl an GPUs. Hierbei fällt jedoch auch die Differenz in der Beschleunigung gegenüber Abb. 7.17 auf. Bei einer ungleichen Verteilung von Nicht-Null Einträgen in der Matrix kann die Berechnung der einzelnen Teileinträge unterschiedlich lange dauern. Hinzu kommt, dass die Rechenoperation und Datenübertragungen nacheinander ausgelöst werden und unterschiedlich lange dauern. Hier kann es zu positiven oder negativen Effekten kommen. Ein positiver Effekt ist es, wenn die Datenübertragungen zwischen den GPUs, wie in Kap. 5.1.3.3 beschrieben, die gleich Zeitspanne benötigen. Ein negativer Effekt ist es, wenn nur eine GPU Berechnungen durchführt, während die anderen GPUs auf den Abschluss der Operation warten.

In Abb. 7.18b ist nur der Geschwindigkeitsgewinn für Probleme erster Ordnung dargestellt. Bei diesem Problem ist Trilinos ML aufgrund der diskreten Größe in der gegebenen Konfiguration nicht in der Lage, einen Vorkonditionierer aufzustellen. Hier wäre eine Clusterimplementation unter Nutzung von MPI [59] erforderlich. Auch bei Verwendung des Rechensystem Nr.1 belegt der Lösungsvorgang nahezu den gesamten verfügbaren Speichers, sodass die Berechnung mit dem Verweis auf nicht ausreichenden Arbeitsspeicherplatz abgebrochen wird. Dieses Problem tritt bei der wiederholten Lösung von linearen Gleichungssystemen auf, wie dies bei der elektroquasistatischen Simulation der Fall ist.

Die Anwendungsbeispiele in Abb. 7.18c und Abb. 7.19a zeigen qualitativ dieselben Beschleunigungswerte. Auf dem System Nr.1 ist das Problem bei Ansatzfunktionen zweiter Ordnung nur unter Verwendung von vier GPUs zu lösen. Qualitativ zeigt sich jedoch auch hier ein mit den anderen Ergebnissen der Abb. 7.17, 7.18 und 7.19 vergleichbarer Geschwindigkeitsgewinn um den Faktor $8,4$ (Abb. 7.18c) bzw. $9,6$ (Abb. 7.19a).

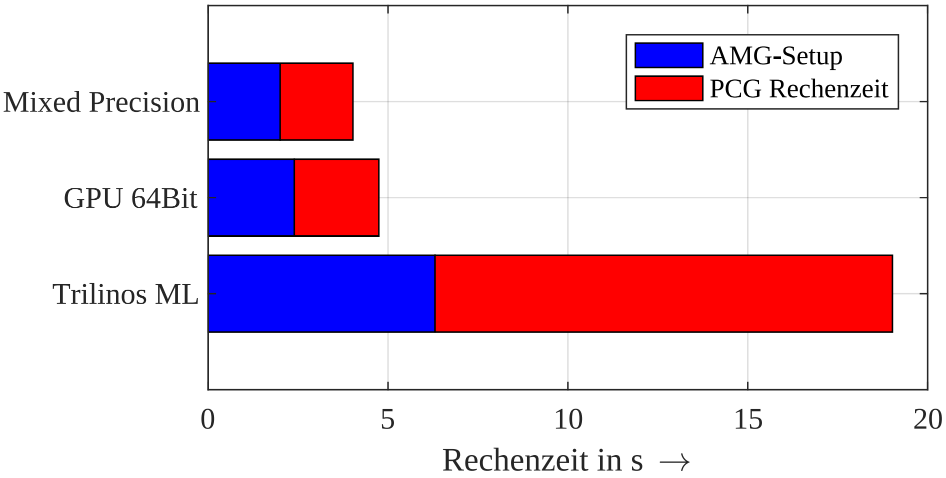


Abbildung 7.20: Vergleich der Zeiten für Setup-Phasen und Solve-Phasen für Mixed-Precision- und Double-Precision-AMG-Vorkonditionierer, sowie die CPU-basierte Referenz [B6].

### 7.3.4 Mixed-Precision-Ansatz

Abb. 7.20 zeigt den Zeitaufwand für das Aufstellen des Vorkonditionierers (die Setup-Phase) mit dem Grundverfahren, sowie mit dem Mixed-Precision-Ansatz. Als Testbeispiel wird das Modellproblem Nr. 1 mit Ansatzfunktionen erster Ordnung genutzt. Die Beispielrechnung erfolgt auf dem System Nr.2 und Nutzung einer Nvidia K20m.

In Abb. 7.20 erkennt man, dass der Zeitaufwand zum Erstellen des Mixed-Precision-Vorkonditionierers geringer ist als dies bei dem ursprünglichen DP-Vorkonditionierer der Fall ist. Die Zeit zur Konvertierung der Matrix ist hierbei farblich hervorgehoben. Für die genannte Kombination wird eine 17 % geringere Zeit gegenüber dem Ausgangsansatz benötigt. Dies deckt sich quantitativ mit Berechnungen auf anderen Rechensystemen und zeigt sich auch bei der Berechnung anderer Beispielmodelle.

In Abb. 7.21 ist der Speicherbedarf bei Verwendung der verschiedenen Vorkonditionierer-Versionen zu sehen. Durch den Mixed-Precision-Vorkonditionierer entsteht insbesondere im Vorkonditionierer ein erhöhter Speicherbedarf. Dieser ist begründet durch den doppelt vorhandenen Matrixeintrag-Vektor. Zusätzlich ist in Abb. 7.21 der Speicherbedarf bei Verwendung eines voll-

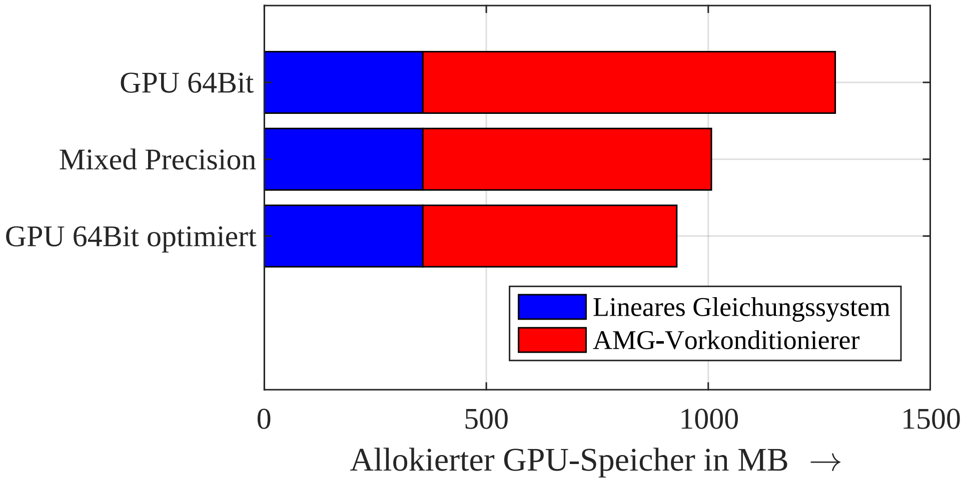


Abbildung 7.21: Vergleich des Speicherbedarfs für Mixed-Precision-AMG-Vorkonditionierer („Mixed Precision“) und DP-AMG-Vorkonditionierer („GPU 64bit“), sowie eines optimierten AMG-Vorkonditionierers mit DP-Zahlendarstellung („GPU 64bit optimiert“).

ständigen Vorkonditionierers dargestellt. Da hier die Systemmatrix auch doppelt gespeichert wird, ist der Speicherbedarf für diesen Fall deutlich höher. Durch den Mixed-Precision-Vorkonditionierer zusätzlich wird noch der Speicherbedarf reduziert [B6]. Durch eine weitere Optimierung im MEQSICO-CUSP-Code wurde die redundante Systemmatrix jedoch nach der Veröffentlichung [B6] entfernt. Das Ergebnis ist als „GPU optimiert“ in Abb. 7.21 dargestellt. Der für die Beschleunigung der Gesamtsimulation relevanteste Teil ist jedoch der Zeitgewinn in der CG-Iteration. Dieser ist in Abb. 7.20 für das Beispielmodell Nr. 1 dargestellt. Man erkennt, dass der gesamte Lösungsvorgang um 19 % beschleunigt wird. Da die Beschleunigung im AMG-Vorkonditionierer stattfindet, ist hier die Beschleunigung höher. Er wird um 39 % beschleunigt.

Abb. 7.22 zeigt das Residuum über die Iterationen der CG-Iteration als Konvergenzplot. Durch die Ergebnisse wird die Annahme bestätigt, dass aufgrund des approximativen Charakters eines Vorkonditionierers im PCG-Verfahren die geringere Genauigkeit des Mixed-Precision-Vorkonditionierers auch nur geringen Einfluss auf die Konvergenzrate des PCG-Verfahrens hat.

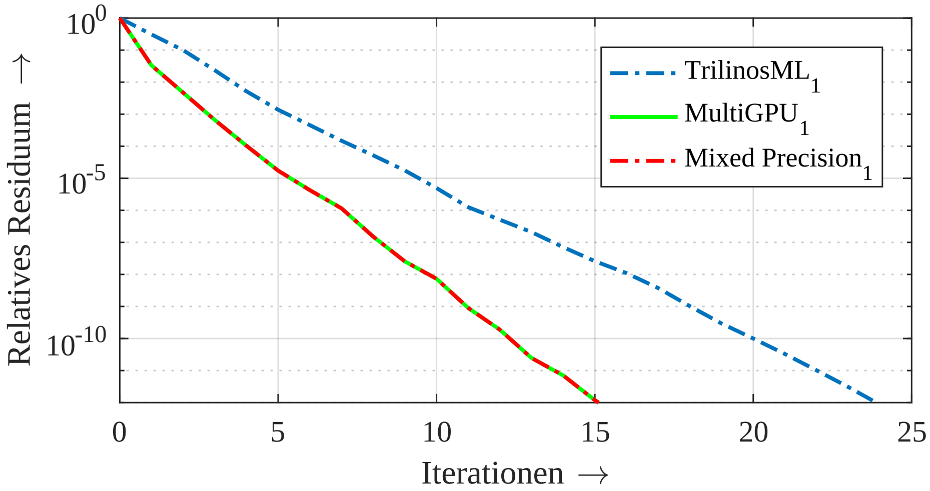


Abbildung 7.22: Konvergenzplots bei Nutzung des Mixed-Precision Vorkonditionierers gegenüber einem herkömmlichen Double-Precision-Vorkonditionierer, sowie der CPU-basierenden Referenzlösung [B6].

## 7.4 Validierung der GPU-basierten Assemblierung

Im Folgenden werden die in Kap 7.2 vorgestellten Beispielprobleme betrachtet und im Hinblick auf eine CPU-basierte Assemblierung gegenüber der GPU-basierten Assemblierungs- und SpMV-Kombination untersucht. Für die CPU-basierte Assemblierung wird auf die Betrachtung zwischen einem Kern und der maximal zur Verfügung stehenden Anzahl an Kernen des entsprechenden Rechensystems verzichtet, da das GPU-System gegen die schnellste zur Verfügung stehende CPU basierte Variante verglichen werden soll.

Der Vergleich zwischen einfacher Assemblierung und GPU-basierter Assemblierungs- und SpMV-Kombination ist relevant, da, wie in Kap. 6 beschrieben, in der impliziten Zeitintegration die nichtlineare Leitwertmatrix $\mathbf{K}_\kappa$ und ihre Jacobimatrix wiederholt assembliert werden (vgl. (6.1)). Bei der expliziten Zeitintegration ist die nichtlineare Leitwertmatrix $\mathbf{K}_\kappa$ Teil der rechten Seite (vgl. (6.6)), weshalb die Assemblierungs- und SpMV-Kombination genutzt werden kann. Daher werden die Beispielprobleme Nr. 2 bis Nr. 7

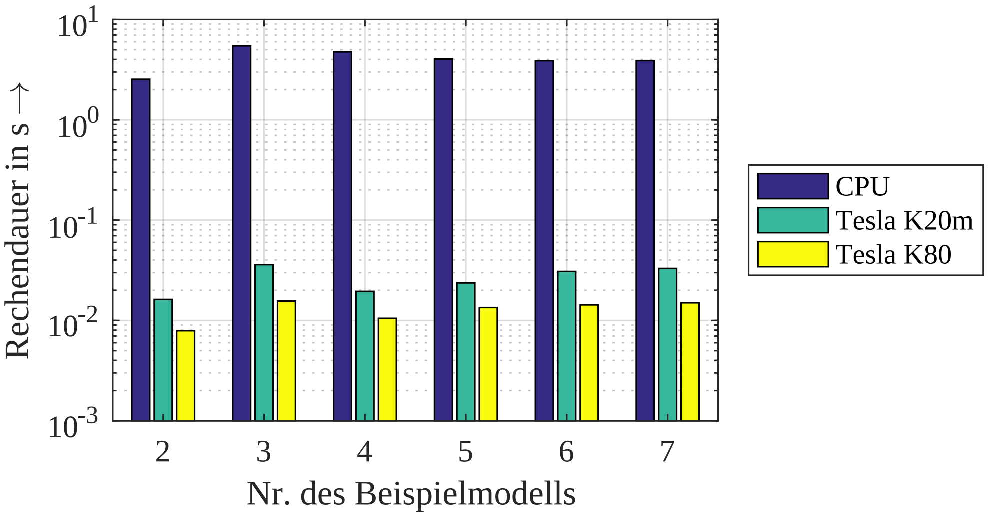


Abbildung 7.23: Zeiten für eine CPU-basierte Assemblierung und die Assemblierungs- und SpMV-Kombination auf GPUs für verschiedene Beispielmodelle mit Ansatzfunktionen erster Ordnung.

betrachtet, welche eine Mikrovaristorschicht und damit eine nichtlineare Leitwertmatrix $\mathbf{K}_\kappa$ enthalten.

In Abb. 7.23 ist die CPU basierte Assemblierung mit der vorgestellten Kombination bei Berechnung auf den Streamingprozessoren der beiden Referenzsysteme, einer Nvidia Tesla K20m und einer Nvidia Tesla K80, dargestellt. Die Rechendauer für die Durchführung der Operation ist logarithmisch dargestellt. Man erkennt, dass für alle Probleme die GPU basierte Kombination über zwei Größenordnungen schneller ist. Ferner ist ein Rechenzeitunterschied zwischen der Tesla K20m und der Tesla K80 zugunsten der Tesla K80 feststellbar. Dieser ist durch die höhere Bandbreite und die höhere Rechengeschwindigkeit bei Nutzung doppelt genauer Zahlen und die höhere Anzahl an CUDA-Cores bei der Tesla K80 zu erklären.

Abb. 7.24 stellt den Geschwindigkeitsgewinn $\frac{\text{Rechenzeit CPU}}{\text{Rechenzeit GPU}}$ (engl.: Speedup) für die beiden GPUs gegenüber der CPU-Implementierung dar. Man erkennt, dass mit einer Tesla K20m ein Geschwindigkeitsgewinn von Faktor 118 bis zu Faktor 244 erreicht werden kann. Für die Tesla K80 sind Geschwindigkeitsgewinne zwischen Faktor 260 und Faktor 453 aufzeigbar.

Die Assemblierung von FEM-Matrizen auf GPUs wurde in einer größe-

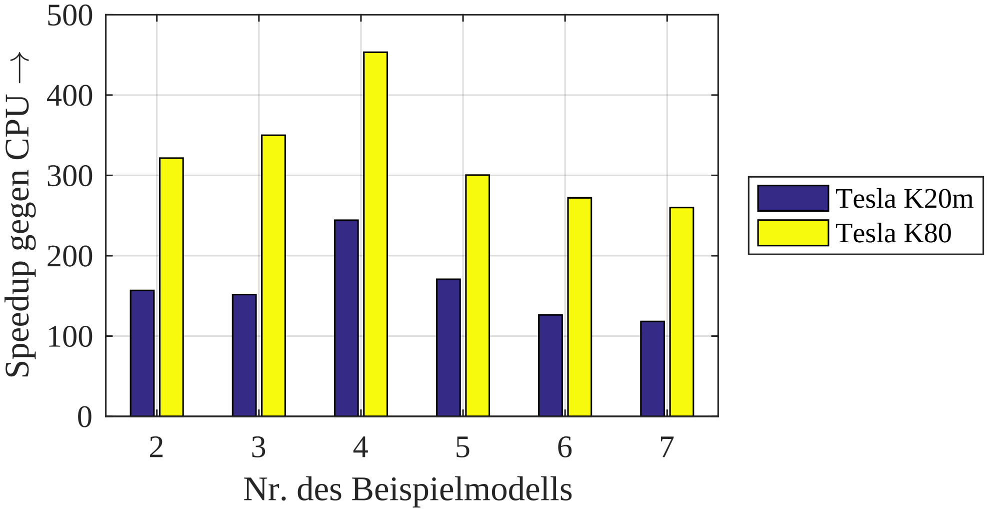


Abbildung 7.24: Geschwindigkeitsgewinn (engl: Speedup) der kombinierten Assemblierung und SpMV auf GPUs gegenüber der CPU-basierten Assemblierung für verschiedene Beispielmodelle mit Ansatzfunktionen erster Ordnung.

ren Zahl von Veröffentlichungen behandelt (vgl. Kap. 5.2). Der Vergleich der absoluten Werte fällt aufgrund der unterschiedlichen GPUs, welche verwendet wurden, schwer. 2011 wurde in [116] einen Beschleunigungsfaktor von 35 für die Assemblierung der lokalen Elemente in einfachgenauer Zahlendarstellung gezeigt. Dort wurde eine Nvidia GTX 8800 verwendet, welche über eine Speicherbandbreite von 86 Gb/s und eine Rechenleistung von 345,6 GFLOPS verfügt. [125] stellt einen kombinierten Assemblierungs- und SpMV-Kernel für 3D-Tetraeder vor, welcher die selben Ziele verfolgt, wie der in dieser Arbeit vorgestellte Kernel. Hier wurde eine Nvidia Q5000 mit einer Speicherbandbreite von 120 Gb/s und einer Rechenleistung von 722 GFLOPS verwendet. Es wurde bei einer Rechenzeit von 122 ms eine Beschleunigung um den Faktor 12 für Modelle mit $n > 10^6$ Freiheitsgrade erreicht. Das Modell aus [125] verfügt dabei über halb so viele Tetraeder, wie im Model Nr. 2 dieser Arbeit vorhanden sind. Die in dieser Arbeit vorgestellte Implementation ist damit auf der Nvidia Telsa K80 um einen Faktor 15 schneller, wobei die Bandbreite doppelt so groß ist und die Rechenleistung einem Faktor Vier entspricht. In [125] wird ausgeführt, dass der

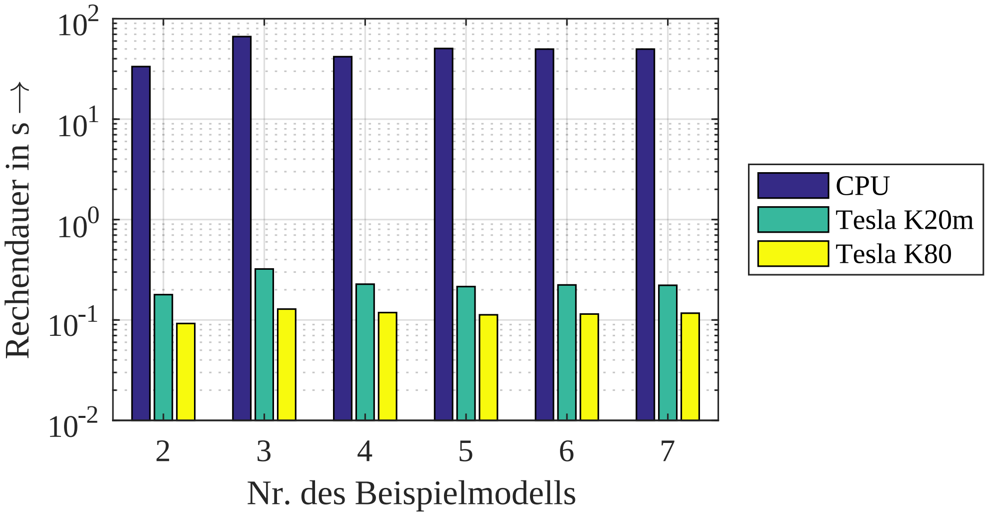


Abbildung 7.25: Zeiten für eine CPU-basierte Assemblierung und die kombinierte Assemblierung und SpMV auf GPUs für verschiedene Beispielmodelle mit Ansatzfunktionen zweiter Ordnung.

größte limitierende Faktor die Anzahl der zur Verfügung stehenden Register ist. Der verbleibende Speicherbedarf eines Threads wird, wenn die Register belegt sind, aus dem um eine Größenordnung langsameren globalen Speicher bedient. Diese Beschränkung wird mit dem in dieser Arbeit vorgestellten Ansatz und unter Nutzung neuerer GPUs umgangen.

Die gleiche Untersuchung wie mit Ansatzfunktionen erster Ordnung ist in Abb. 7.25 und Abb. 7.26 für die Probleme mit Ansatzfunktionen zweiter Ordnung dargestellt. Durch die höhere Anzahl an Integrationspunkten - zehn statt vier - sind hier mehr Multiplikationen pro übertragener Daten durchzuführen, was zu einer Veränderung des Rechenaufwandes pro Element und des Einflusses der massiven Parallelisierung durch die GPUs führt.

Der höhere Rechenaufwand pro Element ist in der Berechnungsdauer in Abb. 7.25 zu erkennen. Sowohl Assemblierung als auch die GPU-basierte Kombination benötigen etwa eine Größenordnung mehr Zeit zur Durchführung der Operation bei je Modell unveränderter Elementenanzahl. Das qualitative Verhalten bleibt jedoch unverändert: Durch Nutzung der kombinierten Assemblierung und SpMV kann die Rechenzeit deutlich reduziert werden.

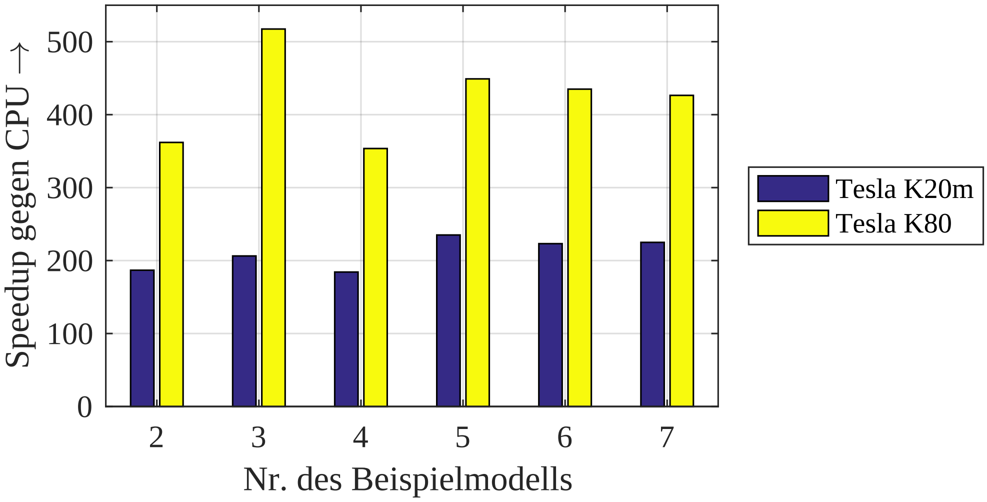


Abbildung 7.26: Geschwindigkeitsgewinn (engl: Speedup) der kombinierten Assemblierung und SpMV auf GPUs gegenüber der CPU-basierten Assemblierung für verschiedene Beispielmodelle mit Ansatzfunktionen zweiter Ordnung.

Auch die in Abb. 7.26 dargestellten Ergebnisse bestätigen diese Beobachtung. Der relative Geschwindigkeitsunterschied zwischen den beiden Streamingprozessoren Tesla K20m und Tesla K80 bleibt etwa gleich verglichen zur Verwendung von Ansatzfunktionen erster Ordnung (Abb. 7.24). Durch die gerade beschriebenen Unterschiede in der Ansatzfunktion sind jedoch noch mehr Berechnungen pro zu übertragendem Datenteil notwendig. Dies ermöglicht eine weitere Verbesserung der Beschleunigung. Für die Tesla K20m werden hierbei Beschleunigungswerte von 184 bis 235 und für die Tesla K80 von 353 bis 517 erreicht.

## 7.5 Untersuchung der Zeitintegratoren anhand der Anwendungsbeispiele

Die in Kap. 6 beschriebenen Ansätze zur Zeitintegration sollen im Folgenden auf Funktionsfähigkeit und Wirksamkeit geprüft werden. Hierzu werden die in Kap. 7.2 definierten Beispielprobleme mit den verschiedenen Ansätzen

| | Bezeichnung | Ansatz |
|---|---|---|
| **A** | TrilinosML | Implizites SDIRK3(2) mit CPU /Trilinos ML |
| **B** | implizit 1 | SDIRK3(2) mit Multi-GPU-PCG (vgl. Kap. 5.1) |
| **C** | implizit 2 | **B** mit adaptivem Vorkonditionierer (vgl. Kap. 6.1) |
| **D** | explizit RKC | Expliziter RKC-Zeitintegrator (vgl. Kap. 6.2) |
| **E** | explizit SPE | **D** mit SPE aus $\mathbf{f}_r,\ r = i - 10, .., i$ (vgl. Kap. 6.2.2) |
| **F** | explizit MOR | **D** mit MOR aus $\mathbf{f}_r,\ r = 1, .., 20$ (vgl. Kap. 6.2.3) |
| **G** | explizit MOR2 | **D** mit MOR mit $\mathbf{f}_r,\ r = i - 20, .., i$ (vgl. Kap. 6.2.3) |

Tabelle 7.6: Zu vergleichende Zeitintegrationsansätze. Ansätze **E** bis **G** berechnen einen Startwert für das PCG-Verfahren im Zeitschritt $t_i \rightarrow t_{i+1}$

gem. Tab. 7.6 als transient nichtlineares EQS-Problem in vorgegebenen Zeitintervallen gelöst.

Jedes Problem wird mit Ansatzfunktionen erster und zweiter Ordnung gelöst. Dabei wird das Problem zuerst mit dem CPU-basierenden impliziten Zeitintegrator simuliert (Tab. 7.6 – **A**), dessen AMG-PCG-Verfahren auf Trilinos ML basiert [101]. Die Parallelisierung der Berechnung erfolgt hier mit OpenMP [60]. Die Ergebnisse werden als Referenz für die beschriebenen Ansätze genutzt. Anschließend wird das jeweilige Problem unter Nutzung des in Kap. 5.1 beschriebenen Multi-GPU Lösers berechnet (Tab. 7.6 – **B**). Eine weiterer Ansatz (Tab. 7.6 – **C**) zur Beschleunigung der Rechenzeit wird durch das Wiederverwenden des AMG-Vorkonditionierers erreicht, wie es in Kap. 6.1 beschrieben ist. Diese Ansätze basieren auf dem impliziten SDIRK3(2) Zeitintegrator [48].

Dem gegenüber werden die entwickelten Ansätze auf Basis eines expliziten Zeitintegrators gestellt. Das jeweilige Problem wird hierzu mit dem in Kap. 6.2 vorgestellten RKC-Zeitintegrator unter Nutzung des Multi-GPU-Lösers für das lineare Gleichungssystem gelöst (Tab. 7.6 – **D**). Zur Startwertextrapolation für das PCG-Verfahren werden im RKC-Verfahren ein SPE-Ansatz (vgl. Kap. 6.2.2) und zwei MOR-Varianten (vgl. Kap. 6.2.3) genutzt. Der SPE-Ansatz (Tab. 7.6 – **E**) nutzt die Lösungen der zehn letzten Zeitschritte. Eine größere Zahl an Lösungen verbessern die Schätzung nicht. Der MOR-Ansatz (Tab. 7.6 – **F**) generiert die Projektionsmatrix einmalig aus den ersten 20 Lösungsvektoren. Der zweite MOR-Ansatz (Tab. 7.6 – **G**)

| Bezeichnung | Wert | Bemerkung |
|---|---|---|
| Relatives Residuum PCG-Verfahren | $\leqslant 10^{-12}$ | $r_{tol} = \frac{\|\|\mathbf{r}\|\|}{\|\|\mathbf{b}\|\|}$ |
| Geschätzter Zeitintegrationsfehler | $\leqslant 10^{-3}$ | $e_{rel} = \frac{\|\|\mathbf{e}\|\|}{\|\|\mathbf{u}\|\|}$ |
| Nichtlineares Residuum | $\leqslant 10^{-8}$ | Nur implizite Verfahren |
| $\Delta t$ für SDIRK3(2) | 0,5 ms | |
| $\Delta t$ für RKC | 0,25 ms | Anfangszeitschrittweite |

Tabelle 7.7: Toleranzwerte für die zu vergleichenden Verfahren gem. Tab. 7.6

| | Ansatzfunktionen | |
|---|---|---|
| Modell | Erster Ordnung | Zweiter Ordnung |
| Nr. 2 | Nr. 2 (Tesla K80) | Nr. 1 (4x Tesla K20m) |
| Nr. 3 | Nr. 2 (Tesla K80) | Nr. 2 (Tesla K80) |
| Nr. 4 | Nr. 2 (Tesla K80) | Nr. 1 (4x Tesla K20m) |
| Nr. 5 | Nr. 2 (Tesla K80) | Nr. 1 (4x Tesla K20m) |
| Nr. 6 | Nr. 2 (Tesla K80) | Nr. 2 (Tesla K80) |
| Nr. 7 | Nr. 2 (Tesla K80) | Nr. 2 (Tesla K80) |

Tabelle 7.8: Toleranzwerte für die zu vergleichenden Verfahren gem. Tab. 7.6

generiert hingegen alle 20 Zeitschritte eine neue Projektionsmatrix aus den 20 letzten Lösungen. Betrachtet man nun den Besetzungsgrad der Matrizen für die Beispielmodelle aus Tab. 7.2 wird deutlich, dass die Speicherung von 15 Snapshots den Speicherplatz einer Systemmatrix mit Ansatzfunktionen erster Ordnung und die Speicherung von 30 Snapshots den Speicherbedarf einer Systemmatrix mit Ansatzfunktionen zweiter Ordnung übersteigt. Dies verdeutlicht, dass es oft nur schwer möglich ist, die oft dreistellige Zahl von Snapshots zu speichern, welche nötig ist, um die volle Dynamik des Systems zu erfassen. Zudem bedeutet dies eine sehr lange Berechnungszeit für die SVD, welche die Anzahl der Snapshots $R$ mit $\mathcal{O}(R^3)$ skaliert [138]. Zur Berechnung aller Ansätze werden die Vorgaben gem. Tab. 7.7 festgelegt.

Die Probleme werden jeweils in der schnellsten ermittelten Hardwarekombination gerechnet, welche in Tab. 7.8 dargestellt ist. Die CPU-basierte Lösung wird jeweils auf dem Rechensystem Nr. 2 gerechnet. Die Probleme Nr. 3, 6 und 7 lassen sich aufgrund des Speicherplatzbedarfs durch die hohe Anzahl an Freiheitsgraden nur auf Rechensystem Nr. 2 berechnen.

| | CPU | GPU | | | | | |
|---|---|---|---|---|---|---|---|
| | Implizit | | | Explizit | | | |
| | **A** | **B** | **C** | **D** | **E** | **F** | **G** |
| Modell Nr. 2 | | | | | | | |
| Rechenzeit [s] | 10825 | 5733 | 2925 | 1876 | 676 | 874 | 992 |
| Beschleunigung | 1,00 | 1,89 | 3,70 | 5,77 | **16,01** | 12,39 | 10,90 |
| CG Iterationen | 13649 | 8870 | 8978 | 17372 | 5625 | 6204 | 4356 |

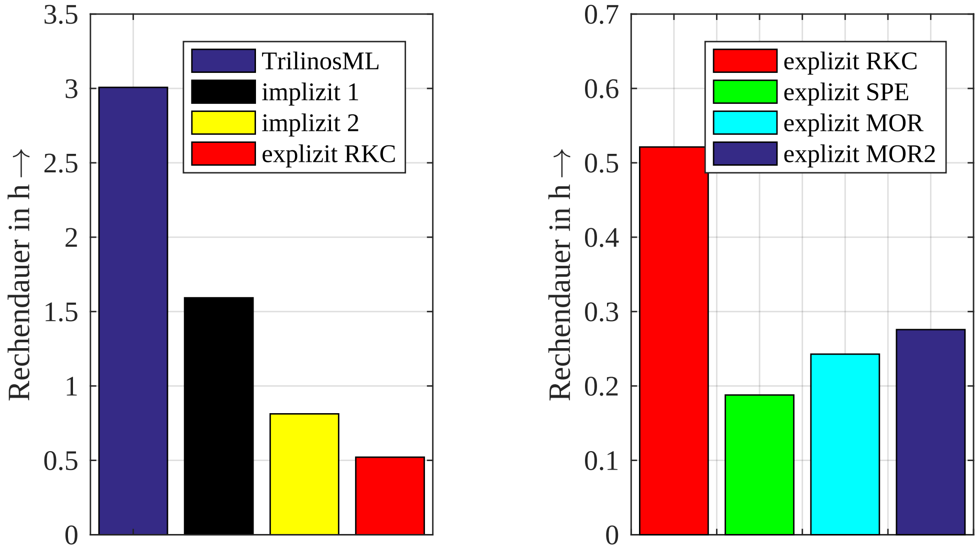
Tabelle 7.9: Übersicht der Rechenzeit, Beschleunigung und der Summe aller CG-Iterationen für die Zeitintegrationsvarianten gem. Tab. 7.6 und das Beispielmodell Nr. 2.

### 7.5.1 Beschleunigte Simulationen für Beispielmodelle mit Ansatzfunktionen erster Ordnung

Die in Kap. 7.2 beschriebenen Beispiele bei angenommener 50 Hz Sinusschwingung werden im Folgenden wegen der Nichtlinearität im Zeitbereich gelöst. Im Verlauf der ersten Halbwelle wird eine lineare Steigerung aufmultipliziert („ramp-up“), vgl. Kap. 6.2. Die Potentialverteilung über der Zeit wird über zwei Perioden berechnet.

Hierzu werden die verschiedenen beschriebenen Zeitintegrationsmethoden aus Tab. 7.6 genutzt. In diesem Kapitel erfolgt die Berechnung mit Ansatzfunktionen erster Ordnung. Post-Processing und initiale Berechnungen werden nicht betrachtet, da diese für alle Zeitintegrationsansätze gleich sind.

Zuerst wird die Rechendauer für die definierten Ansätze am Modell Nr. 2, der Hochspannungsdurchführung, dargestellt und analysiert. Die analogen Ergebnisse für die weiteren Modelle werden in grafischer Form dargestellt. Ein Vergleich erfolgt jedoch anschließend für alle Modelle anhand der Beschleunigungsfaktoren. Abschließend werden alle Ergebnisse in Tabellenform angegeben.



(a) Vergleich verschiedener Zeitintegratoren.

(b) Explizite Zeitintegration mit verschiedenen Startwertschätzern.

Abbildung 7.27: Lösungszeit der Lösung des EQS-Problems des Modells Nr. 2, der Hochspannungsdurchführung, mit Ansatzfunktionen erster Ordnung.

#### 7.5.1.1 Vergleich der Verfahren anhand eines Beispielmodells

In Abb. 7.27 und Tab. 7.9 sind die Rechenzeiten für das Modell Nr. 2 dargestellt. Die Simulationszeit für Modell Nr. 2 beträgt für die Referenzlösung (CPU-Systems mit Trilinos ML) 10825 s $\approx 3,0$h.

Durch die Verwendung des Multi-GPU beschleunigten Lösers – Abb. 7.27a „implizit 1“ – kann eine Beschleunigung der Gesamtsimulation um den Faktor 1,9 erreicht werden. Dies ist deutlich unter dem Faktor 4,0, den die Beschleunigung des PCG-Verfahrens für das Problem erreicht, vgl. Kap. 7.3.3. Grund hierfür sind weitere unbeschleunigte Berechnungen, welche in jedem Zeitschritt durchgeführt werden. Hierzu zählen die Assemblierung der nichtlinearen Leitwertmatrix, das Aufstellen des linearisierten Gleichungssystem sowie das Aufstellen des AMG-Vorkonditionieres, welcher aufgrund der sich ändernden Systemmatrix des LGS jedes Mal neu zu berechnen ist.

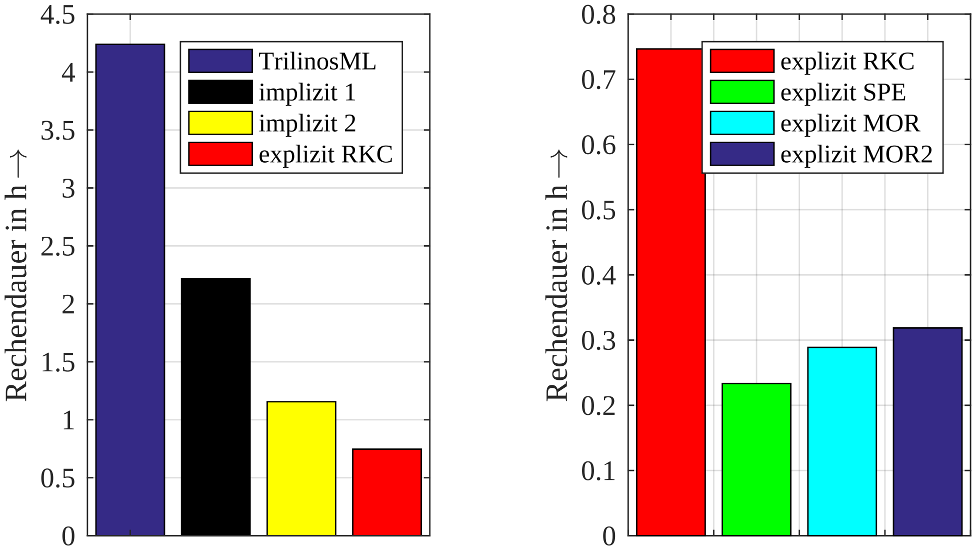
Eine weitere Beschleunigung um den Faktor 1,96 gegenüber dem Ansatz

„implizit 1“ wird durch den Ansatz „implizit 2“ erreicht, bei welchem der Vorkonditionierer wiederverwendet bzw. adaptiert wird, vgl. Kap. 6.1. Aufgrund der geringen Änderung der Systemmatrix bei der betrachteten Problemklasse der EQS-Simulation zeigt sich, dass dieser Ansatz als sehr effizient zu bewerten ist und eine weitere deutliche Reduktion der Simulationszeit erreicht wird.

Die Simulation mittels explizitem RKC-Verfahren ist um den Faktor 5,77 schneller, als bei Nutzung der impliziten CPU-basierten Zeitintegration auf Basis von Trilinos ML, aber auch als die anderen, in Abb. 7.27a dargestellten, impliziten Varianten. Es werden für dieses Modell die in Kap. 6.2 gemachten Annahmen bestätigt: Durch den höheren Parallelisierungsgrad und die Möglichkeit, eine größere Anzahl zu berechnender Abschnitte effizient auf die GPUs zu portieren, wird in Summe eine höhere Beschleunigung erreicht, obwohl das explizite Zeitintegrationserfahren deutlich mehr Zeitschritte benötigt – 8870 CG-Iterationen für „implizit 1“ gegenüber 17372 CG-Iterationen für das RKC-Verfahren. Neben dem Multi-GPU-PCG-Löser aus Kap. 5.1 wird die GPU-beschleunigte kombinierte Assemblierungs- und SpMV-Routine für $\mathbf{K}_\kappa \mathbf{u}$ verwendet. Dies ist hier möglich, da, wie in Kap. 2.4.2 beschrieben, die Leitwertmatrix in den Rechts-Seite-Vektor $\mathbf{b}$ des linearen Gleichungssystems eingeht und nicht Teil der Systemmatrix ist. Da die Systemmatrix konstant ist, ist ein Neuaufstellen des Vorkonditionierers nicht notwendig. Damit finden alle wesentlichen Berechnungsschritte auf der GPU statt.

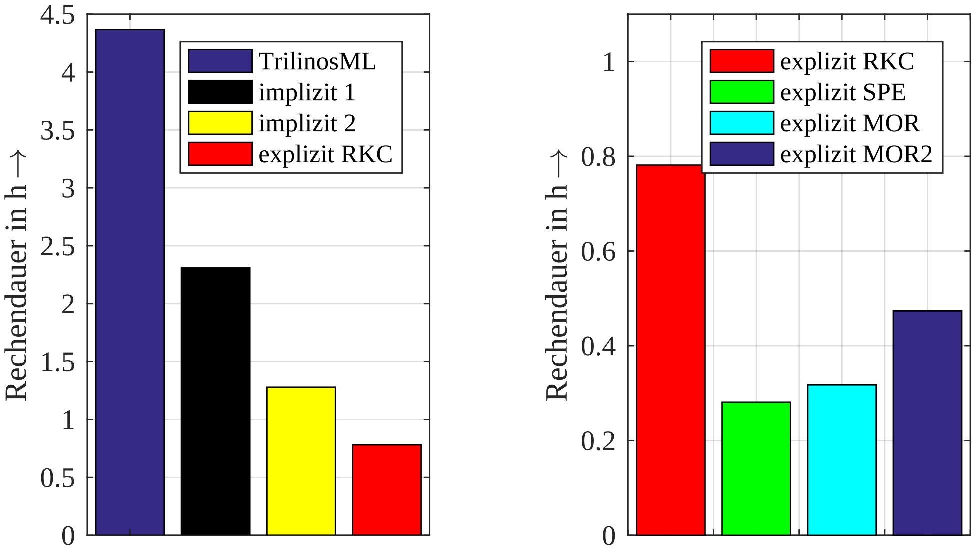
Der rote Balken in Abb. 7.27b stellt den Übergabepunkt aus Abb. 7.27a dar. Die Auswertung ist aus Gründen der Sichtbarkeit in zwei Abbildungen mit unterschiedlicher Skalierung aufgeteilt.

Man erkennt in Abb. 7.27b, dass für das gegebene Problem mit der SPE-Startwertschätzung die geringste Rechenzeit benötigt wird. Dies hängt hier mit dem geringen Rechenaufwand zur Ermittlung des Startlösungsvektors und der stärksten Reduktion an CG-Iterationen zusammen. Mit SPE-Startwertschätzer werden 5625 statt 17372 CG-Iterationen durchgeführt, wie es beim für das RKC-Verfahren ohne Startwertschätzer der Fall ist.



(a) Vergleich GPU-beschleunigter Zeitintegratoren mit Referenz.

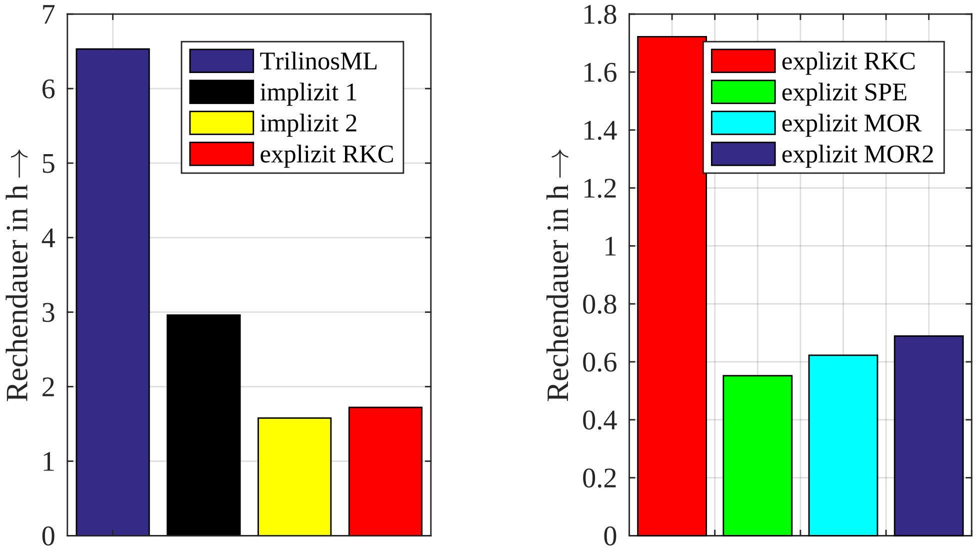
(b) Vergleich der Startwertschätzer für das RKC Verfahren.



(c) Vergleich GPU-beschleunigter Zeitintegratoren mit Referenz.

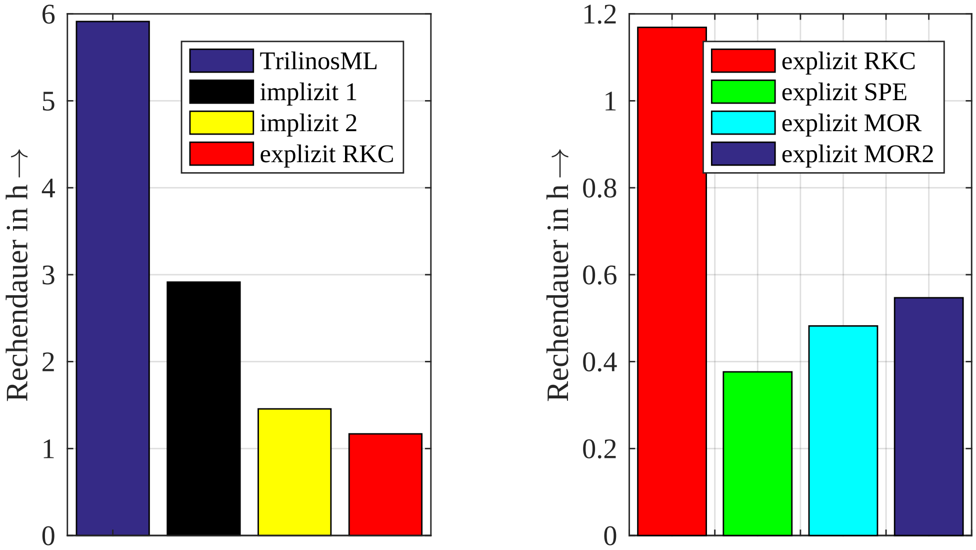
(d) Vergleich der Startwertschätzer für das RKC Verfahren.

Abbildung 7.28: Gesamtzeit zur Lösung des EQS-Problems „Langstabisolator mit 1 mm (Modell Nr. 4 – hier a) und b) ) bzw. 2 mm dickem Feldsteuerelement“ (Modell Nr. 5 – hier c) und d) ) mit Ansatzfunktionen erster Ordnung.



(a) Vergleich GPU-beschleunigter Zeitintegratoren mit Referenz.

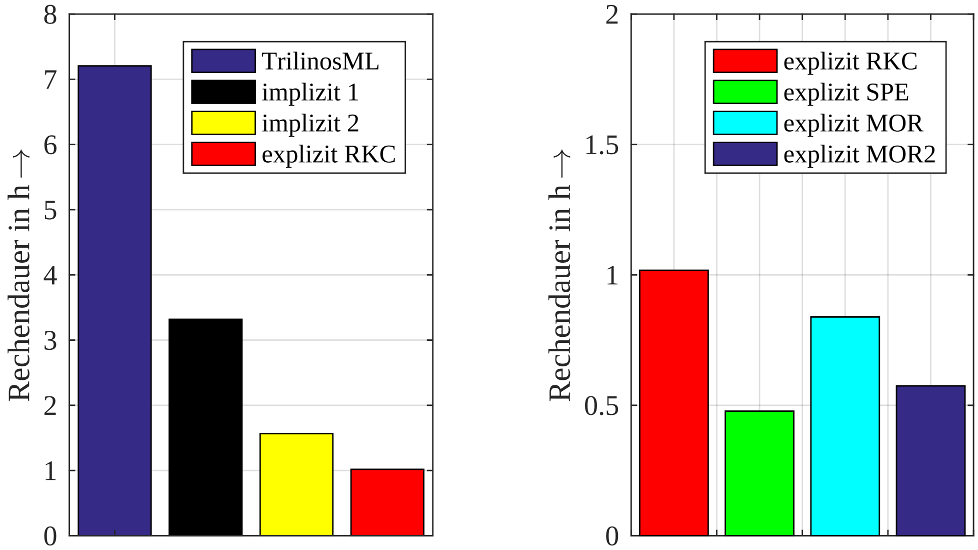
(b) Vergleich der Startwertschätzer für das RKC Verfahren.



(c) Vergleich GPU-beschleunigter Zeitintegratoren mit Referenz.

(d) Vergleich der Startwertschätzer für das RKC Verfahren.

Abbildung 7.29: Gesamtzeit zur Lösung des EQS-Problems „Langstabisolator mit 1 mm (Modell Nr. 6 – hier a) und b) ) bzw. 2 mm dickem Feldsteuerelement und Wassertropfen“ (Modell Nr. 7 – hier c) und d) ) mit Ansatzfunktionen erster Ordnung.



(a) Vergleich GPU-beschleunigter Zeitintegratoren mit Referenz.

(b) Vergleich der Startwertschätzer für das RKC Verfahren.

Abbildung 7.30: Gesamtzeit zur Lösung des EQS-Problems „Langstabisolator mit maximaler Unbekanntenzahl“ (Modell Nr. 3) mit Ansatzfunktionen erster Ordnung.

Für das hier betrachtete Modell mit Ansatzfunktionen erster Ordnung ist die Verwendung des MOR-Ansatzes „explizit MOR“ 198 s oder 29,2 % langsamer als der weniger rechenaufwendige SPE-Ansatz. Es wird jedoch ein deutlicher Geschwindigkeitsgewinn gegenüber dem Ansatz ohne Startwertermittlung erreicht wird, indem die Anzahl der Iterationen reduziert wird. Der Ansatz „explizit MOR2“, bei welchem der Startwert für $\mathbf{f}_{i+1}$ aus $\mathbf{X} = \{\mathbf{f}_{(i-R)}|...|\mathbf{f}_i\}$ ermittelt wird, ist aufgrund der hohen Anzahl an SVDs für das Problem Nr. 2 nicht effizienter als die anderen Startwertschätzer, auch wenn er die Anzahl der CG-Iteration – mit 4356 gegenüber 5625 für die SPE und 6204 für den anderen MOR-Ansatz – am effizientesten reduziert. Es ergibt sich das Abb. 7.27b in dunkelblauem Balken dargestellte Ergebnis für diesen Ansatz. Die absoluten Ergebnisse für die Modellprobleme Nr. 3 bis 7 sind in den Abb. 7.28 bis 7.30 dargestellt.

#### 7.5.1.2 Vergleich der Beschleunigungswerte aller Modellbeispiele

Zur vergleichenden Betrachtung ist die Beschleunigung der Gesamtsimulation vergleichend für alle Modelle in Abb. 7.31 dargestellt. Die farbigen Balken geben die unterschiedlichen Modelle Nr. 2 bis 7 an. Die Beschriftung der x-Achse gibt das Zeitintegrationsverfahren analog zu den zuvor besprochenen Abb. 7.27a und Abb. 7.27b wieder. Dabei korrespondieren die Buchstaben **A** bis **G** zu den Zeitintegrationsverfahren in Tab. 7.10.

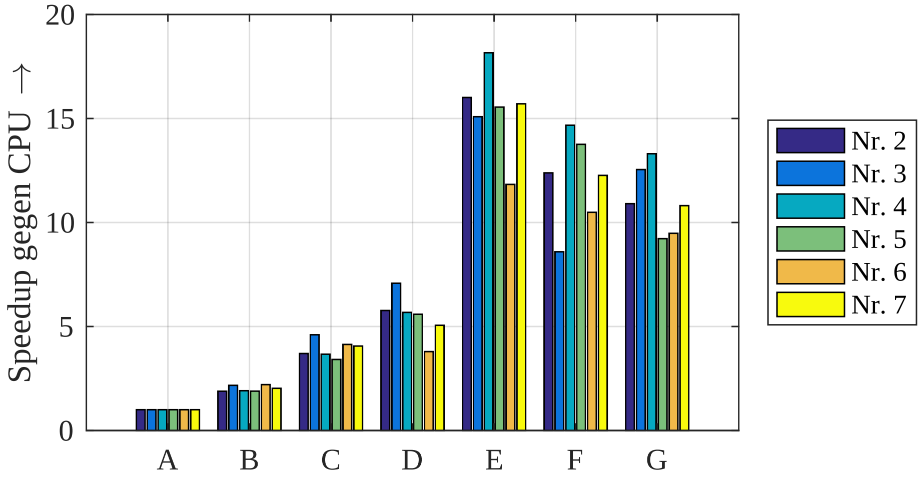


Abbildung 7.31: Geschwindigkeitsgewinn der Rechendauer der EQS-Probleme für die Beispielmodelle Nr. 2 bis 7 mit Ansatzfunktionen erster Ordnung für Zeitintegrationsverfahren gem. Tab. 7.10.

| | Bezeichnung | Ansatz |
|---|---|---|
| **A** | TrilinosML | Implizites SDIRK3(2) mit CPU /Trilinos ML |
| **B** | implizit 1 | SDIRK3(2) mit Multi-GPU-PCG (vgl. Kap. 5.1) |
| **C** | implizit 2 | **B** mit adaptivem Vorkonditionierer (vgl. Kap. 6.1) |
| **D** | explizit RKC | Expliziter RKC-Zeitintegrator (vgl. Kap. 6.2) |
| **E** | explizit SPE | **D** mit SPE aus $\mathbf{f}_r,\ r = i-10,..,i$ (vgl. Kap. 6.2.2) |
| **F** | explizit MOR | **D** mit MOR aus $\mathbf{f}_r,\ r = 1,..,20$ (vgl. Kap. 6.2.3) |
| **G** | explizit MOR2 | **D** mit MOR mit $\mathbf{f}_r,\ r = i-20,..,i$ (vgl. Kap. 6.2.3) |

Tabelle 7.10: Zu vergleichende Zeitintegrationsansätze. Ansätze **E** bis **G** berechnen einen Startwert für das PCG-Verfahren im Zeitschritt $t_i \to t_{i+1}$

Da sich die Rechenzeiten der Modelle aufgrund der unterschiedlichen Größe und Gitterkomplexität unterscheiden, erfolgt der Vergleich nicht in absoluten Zeiten, sondern basierend auf der erreichten Beschleunigung. Das CPU-basierte implizite Zeitintegrationsverfahren **A** bildet hierbei den Referenzpunkt mit dem Wert 1,00.

Mit dem Zeitintegrationsansatz **B** ist für alle Modelle ein etwa konstanter Geschwindigkeitsgewinn um einen Faktor Zwei bis Drei zu erzielen. Eine weitere Beschleunigung wird für alle Modelle mit Ansatz **C** erreicht, der die Rechenzeit für die Aufstellung des Vorkonditionierers reduziert. Hier ist erkennbar, dass insbesondere die größeren Modelle Nr. 3, 6 und 7 von der geringeren Anzahl von Vorkonditionierer-Setups profitieren. Dies ist maßgeblich mit dem Aufwand für die zwei Matrix-Multiplikationen innerhalb des Galerkin-Produktes zu erklären (vgl. Kap 4.2.1).

Ansatz **D**, die explizite Zeitintegration mit dem RKC-Verfahren, erreicht für alle Modelle mit Ausnahme von Modell Nr. 6 eine Beschleunigung gegenüber dem impliziten Zeitintegrationsansatz **C**, wie zuvor für Modell Nr. 2 beschrieben. Bei Modell Nr. 6 kann jedoch keine Beschleunigung gegenüber Zeitintegrationsansatz **C** verbucht werden. Grund hierfür ist die erhöhte Anzahl an CG-Iterationen von 21924 (RKC) gegenüber 11036 für die impliziten Ansätze. Das Konvergenzverhalten des PCG-Lösers wurde bereits in Kap. 7.3.1 untersucht und gezeigt, dass Modell Nr. 6 die meisten Iterationen zur Lösung seine resultierenden LGS benötigt.

Mit der expliziten Zeitintegration mit SPE-Startwertgeneration (Variante **E**) ist für alle Modelle die höchste Beschleunigung zu erreichen. Auffällig ist hier wieder Modell Nr. 6, welches deutlich geringere Beschleunigungswerte vorweist als die anderen Modelle, wobei die für Variante **D** gegebene Begründung hier ebenfalls gültig ist. Mittels der MOR-Ansätze **F** und **G** werden keine Beschleunigungen oberhalb des durch Ansatz **E** erreichten Niveaus möglich. Qualitativ ist das Verhalten aller Modelle ähnlich. Modell Nr. 6 bleibt aufgrund der höheren Anzahl an CG-Iterationen hinter den Beschleunigungswerten der anderen Modelle zurück. Weitere Auffälligkeit ist das Modell Nr. 3 mit Ansatz **F**. Hier bleibt der Beschleunigungswert deutlich hinter den Werten der anderen Modellprobleme zurück, da die Iterationszahl

| | CPU | GPU | | | | | |
|---|---|---|---|---|---|---|---|
| | Implizit | | | Explizit | | | |
| | **A** | **B** | **C** | **D** | **E** | **F** | **G** |
| Modell Nr. 2 | | | | | | | |
| Rechenzeit [s] | 10825 | 5733 | 2925 | 1876 | 676 | 874 | 992 |
| Beschleunigung | 1,00 | 1,89 | 3,70 | 5,77 | **16,01** | 12,39 | 10,90 |
| CG Iterationen | 13649 | 8870 | 8978 | 17372 | 5625 | 6204 | 4356 |
| Modell Nr. 3 | | | | | | | |
| Rechenzeit [s] | 25939 | 11948 | 5636 | 3663 | 1719 | 3019 | 2067 |
| Beschleunigung | 1,00 | 2,17 | 4,60 | 7,08 | **15,09** | 8,59 | 12,54 |
| CG Iterationen | 10702 | 8052 | 7691 | 14404 | 3871 | 5921 | 3532 |
| Modell Nr. 4 | | | | | | | |
| Rechenzeit [s] | 15260 | 7977 | 4160 | 2687 | 840 | 1039 | 1146 |
| Beschleunigung | 1,00 | 1,91 | 3,67 | 5,68 | **18,16** | 14,68 | 13,31 |
| CG Iterationen | 13649 | 8870 | 8978 | 17372 | 5625 | 6204 | 4356 |
| Modell Nr. 5 | | | | | | | |
| Rechenzeit [s] | 15721 | 8308 | 4603 | 2812 | 1011 | 1142 | 1704 |
| Beschleunigung | 1,00 | 1,89 | 3,41 | 5,59 | **15,54** | 13.757 | 9,22 |
| CG Iterationen | 9729 | 7611 | 7644 | 14406 | 4217 | 4017 | 3439 |
| Modell Nr. 6 | | | | | | | |
| Rechenzeit [s] | 23513 | 10659 | 5683 | 6198 | 1987 | 2241 | 2480 |
| Beschleunigung | 1,00 | 2,21 | 4,13 | 3,79 | **11,83** | 10,49 | 9,48 |
| CG Iterationen | 12101 | 10991 | 11036 | 21924 | 6116 | 6445 | 4347 |
| Modell Nr. 7 | | | | | | | |
| Rechenzeit [s] | 21284 | 10493 | 5242 | 4208 | 1355 | 1735 | 1968 |
| Beschleunigung | 1,00 | 2,03 | 4,06 | 5,06 | **15,71** | 12,26 | 10,81 |
| CG Iterationen | 10266 | 8329 | 7939 | 15226 | 4058 | 4792 | 3429 |

Tabelle 7.11: Übersicht der Rechenzeit, Beschleunigung und der Summe aller CG-Iterationen für die Zeitintegrationsvarianten gem. Tab. 7.10 und die Beispielmodelle Nr. 2 bis 7.

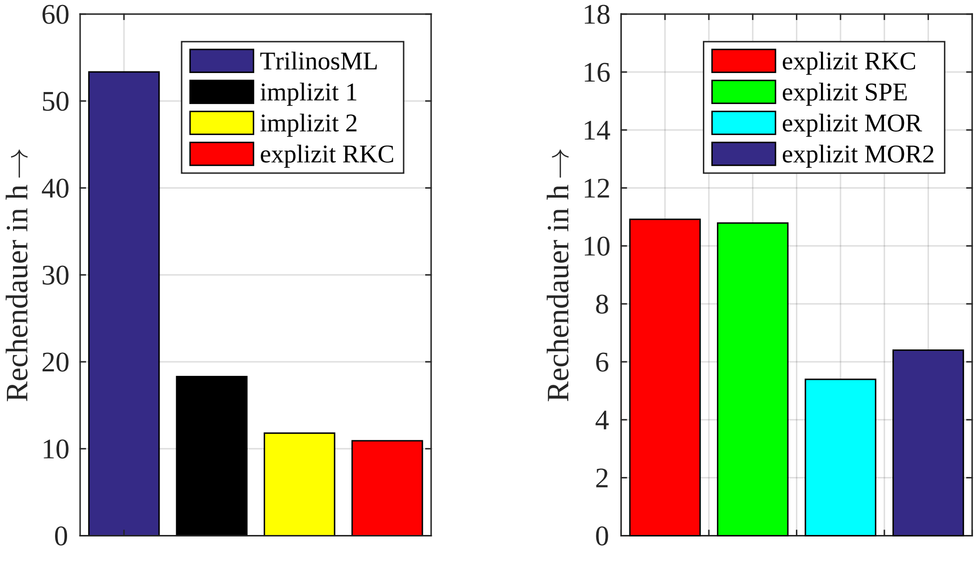
sehr viel schlechter reduziert wird als bei den anderen Modellen. Tab. 7.11 zeigt eine Übersicht der in diesem Abschnitt erhobenen Daten.

### 7.5.2 Beschleunigte Simulationen für Beispielmodelle mit Ansatzfunktionen zweiter Ordnung

Finite Elemente zweiter Ordnung erhöhen die Anzahl der Freiheitsgrade und damit die Größe des zu lösenden Systems deutlich. Zusätzlich erhöht sich auch die Anzahl der Verbindungen und direkten Abhängigkeiten einzelner Freiheitsgrade, welches in der Systemmatrix zu einer höheren Anzahl an Einträgen führt [40, 29]. Die gesteigerte Größe, aber auch die höhere Besetzungsdichte haben deutlichen Einfluss auf die Rechengeschwindigkeit und den Unterschied zwischen einer CPU- und einer GPU-Implementierung, vgl. Kap. 5.1. Im Folgenden soll der Einfluss einer Variation der Ansatzfunktionen auf eine elektroquasistatische Simulation überprüft werden. Die Beispielmodelle werden unter den in Kap. 7.5.1 definierten Rahmenbedingungen gelöst.

#### 7.5.2.1 Vergleich der Verfahren anhand eines Beispielmodells

Analog zu Kap. 7.5.1 erfolgt die Besprechung zuerst an Modell Nr. 2, der Hochspannungsdurchführung, um anschließend das Verhalten an einem Beschleunigungsvergleich aller Modelle zu verifizieren. Betrachtet man die Berechnungszeit unter Nutzung der CPU Referenzlösung aus Abb. 7.32a, so ist erkennbar, dass diese sich gegenüber der Verwendung von Ansatzfunktionen erster Ordnung, wie in Abb. 7.27a, deutlich steigert. Ferner ist die Beschleunigung, welche durch die Nutzung einer GPU erreicht wird, noch deutlicher ausgeprägt, als dies bei Ansatzfunktionen erster Ordnung der Fall ist. Die Hintergründe für die überproportionale Steigerung, etwa die bessere Nutzung der Parallelität durch dichtere Matrixbesetzung, wurden bereits in Kap. 5.1 erläutert und aufgezeigt. Dagegen fällt die Beschleunigung zwischen den Ansätzen „implizit 1“ und „implizit 2“ nicht so stark aus, wie dies mit Ansatzfunktionen erster Ordnung der Fall war. Auch wenn die absolute Zeitreduktion größer ist, fällt die relative Reduktion kleiner aus. Der wesentliche Unterschied zwischen den Ansätzen ist die Wiederverwendung und Adaption des AMG-Vorkonditionierers. Das Verhältnis von Aufstelldauer des AMG-Vorkonditionierer zur Lösungsdauer mit dem PCG-Verfahren



(a) Vergleich verschiedener Zeitintegratoren.

(b) Explizite Zeitintegration mit verschiedenen Startwertschätzern.

Abbildung 7.32: Lösungszeit der Lösung des EQS-Problems des Modells Nr. 2, der Hochspannungsdurchführung, mit Ansatzfunktionen zweiter Ordnung.

verschiebt sich zum PCG-Verfahren. Bei Ansatzfunktionen zweiter Ordnung ist der zeitliche Aufwand für den Lösungsvorgang des LGS deutlich höher, als die Zeit, welche für das Aufstellen des Vorkonditionierers benötigt wird. Zusätzlich fällt die Gesamtanzahl aller benötigten Iterationsschritte mit 38650 um 68 % höher aus, als dies mit 22958 für das implizite Ausgangsverfahren „implizit 1“ der Fall ist (vgl. Tab. 7.12). Demgegenüber steht ein Unterschied von 1,2 % zwischen den Ansätzen „implizit 1“ und „implizit 2“ bei Ansatzfunktionen erster Ordnung. Auch bei Verwendung des expliziten RKC-Zeitintegrationsansatzes ist keine so große Rechenzeitreduktion zu erreichen, wie dies mit Ansatzfunktionen erster Ordnung der Fall ist. Die Zeitreduktion liegt mit 7 % nur unwesentlich unter dem Wert des Ansatzes „implizit 2“. Auch hier ist dies mit dem benötigten zeitlichen Aufwand des PCG-Lösers zu erklären.

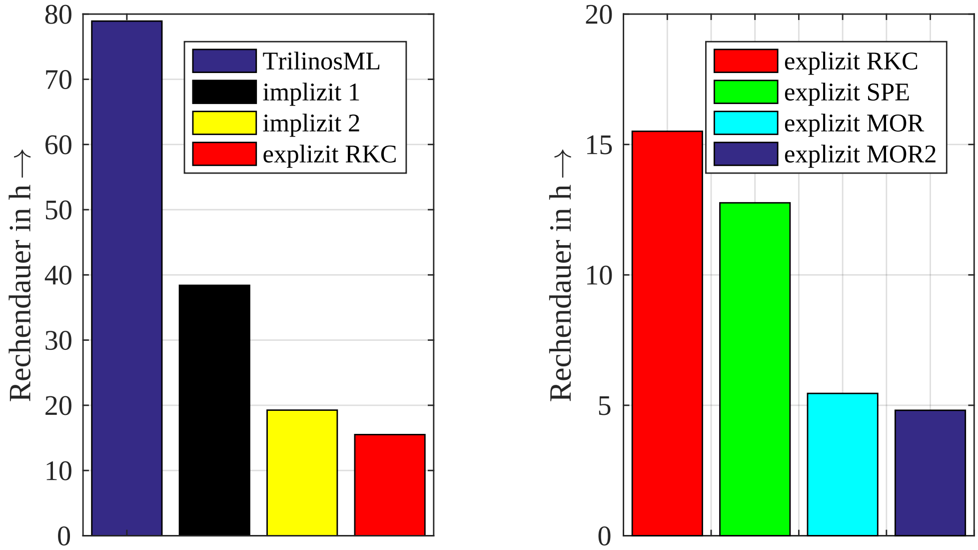
Wie in Tab. 7.12 ersichtlich sind im expliziten Verfahren in Summe über

| | CPU | GPU | | | | | |
|---|---|---|---|---|---|---|---|
| | | Implizit | | | Explizit | | |
| | **A** | **B** | **C** | **D** | **E** | **F** | **G** |
| Modell Nr. 2 | | | | | | | |
| Rechenzeit [$10^3$ s ] | 192,00 | 65,88 | 42,47 | 39,30 | 38,84 | 19,42 | 23,05 |
| Beschleunigung | 1,00 | 2,91 | 4,52 | 4,89 | 4,94 | **9,89** | 8,33 |
| CG Iterationen | 24709 | 22958 | 38650 | 78558 | 77556 | 12662 | 15224 |

Tabelle 7.12: Übersicht der Rechenzeit, Beschleunigung und der Summe aller CG-Iterationen für die Zeitintegrationsvarianten gem. Tab. 7.10 und das Beispielmodell Nr. 2.

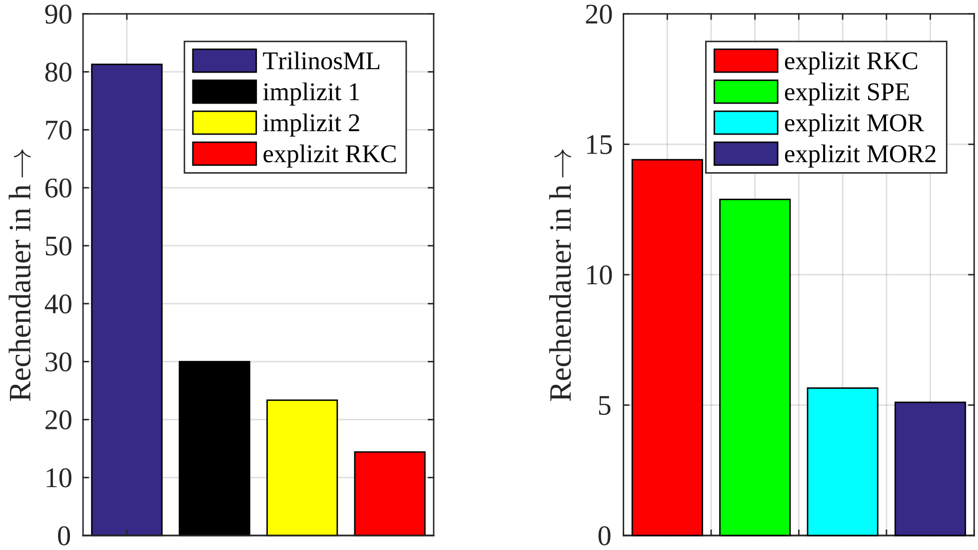
alle gelösten LGS 78558 CG-Iterationen notwendig. Verglichen mit dem impliziten Verfahren „implizit 2“ sind dies noch einmal 2,03 mal so viele Iterationen. Dies kann durch die in Kap. 5.2.2 beschriebene Kombination von Assemblierung und SpMV im RKC-Verfahren nicht kompensiert werden.

Die Anzahl der Iterationen konnte bei dem Modell Nr. 2 mit Ansatzfunktionen erster Ordnung durch Startwertschätzung mit der SPE (vgl. Kap. 6.2.2) deutlich reduziert werden. Dies ist bei Ansatzfunktionen zweiter Ordnung nicht der Fall. Mit 77556 CG-Iterationen mit SPE gegenüber 78558 CG-Iterationen ohne Startwertschätzer wird nur eine marginale Reduktion erreicht, wie Tab. 7.12 zu entnehmen ist. Dies spiegelt sich auch in der Rechenzeit in Abb. 7.32b wider. Die Beschleunigung bei Verwendung der SPE Startwertextrapolation beträgt gegenüber dem RKC-Verfahren vernachlässigbare 1,1 %. Wesentlich wirksamer ist hier die Nutzung des MOR-Ansatzes. Es wird ein Beschleunigungsfaktor von 2,02 erreicht. Dies ist damit zu begründen, dass die Startwertschätzung für Ansatzfunktionen zweiter Ordnung mit MOR effektiver funktioniert und die Anzahl der CG-Iterationen von 78558 ohne Starwertschätzer auf 12662 gesenkt wird (vgl. Tab. 7.12). Bei Verwendung des abschließend untersuchten Ansatzes MOR2, welcher den reduzierten Raum aus den letzten 20 Ergebnissen aufspannt, ist für das Modell Nr. 2 in Abb. 7.32b keine Reduzierung der Rechenzeit feststellbar. Die Rechenzeit für die höhere Anzahl an SVD’s kann nicht durch die erreichte Reduktion der CG-Iterationen und damit eine geringer Rechenzeit für den PCG-Löser kompensiert werden.



(a) Vergleich GPU-beschleunigter Zeitintegratoren mit Referenz.

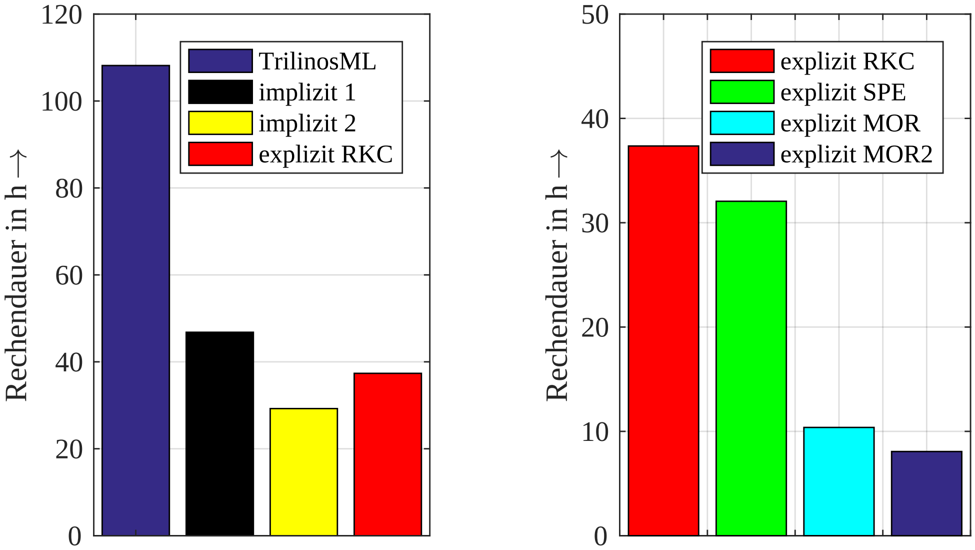
(b) Vergleich der Startwertschätzer für das RKC Verfahren.



(c) Vergleich GPU-beschleunigter Zeitintegratoren mit Referenz.

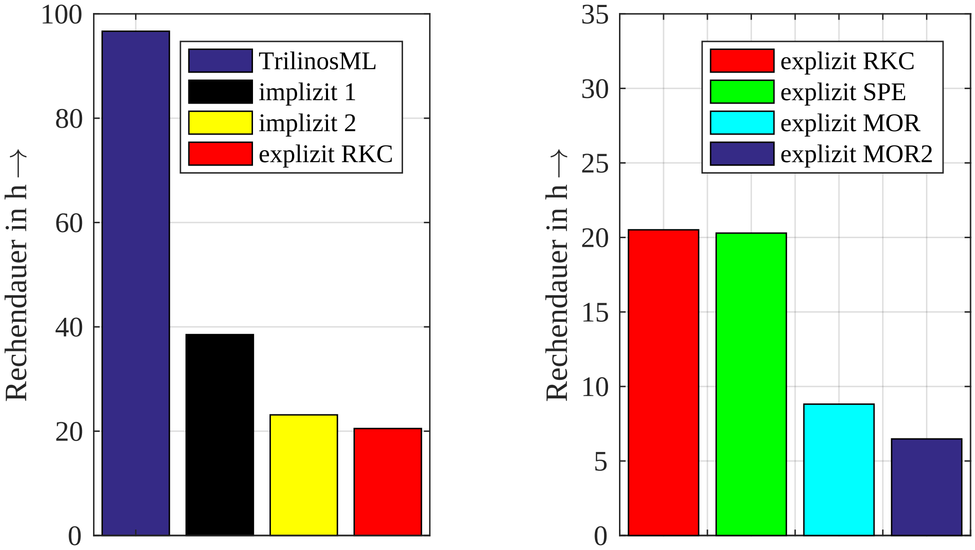
(d) Vergleich der Startwertschätzer für das RKC Verfahren.

Abbildung 7.33: Gesamtzeit zur Lösung des EQS-Problems „Langstabisolator mit 1 mm (Modell Nr. 4 – hier a) und b) ) bzw. 2 mm dickem Feldsteuerelement“ (Modell Nr. 5 – hier c) und d) ) mit Ansatzfunktionen zweiter Ordnung.



(a) Vergleich GPU-beschleunigter Zeitintegratoren mit Referenz.

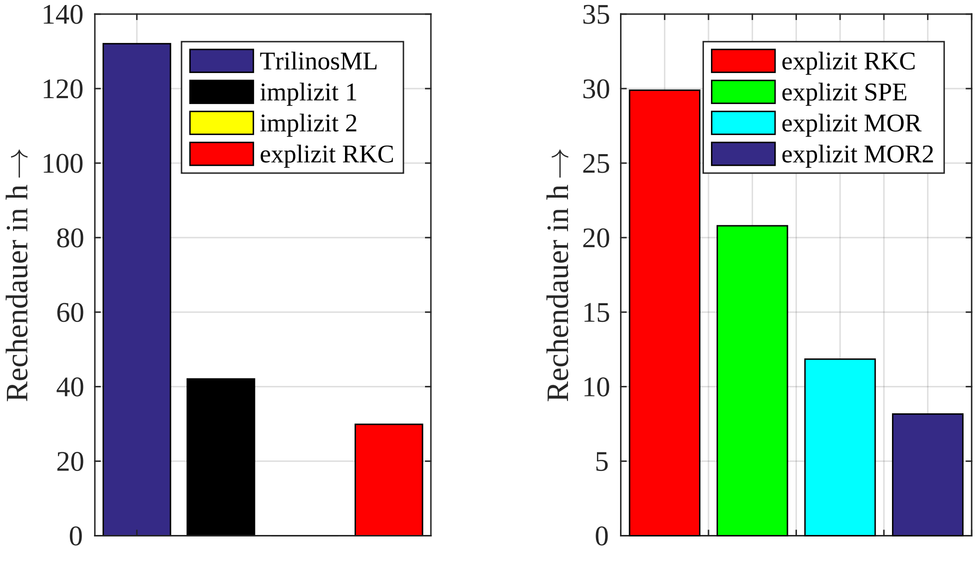
(b) Vergleich der Startwertschätzer für das RKC Verfahren.



(c) Vergleich GPU-beschleunigter Zeitintegratoren mit Referenz.

(d) Vergleich der Startwertschätzer für das RKC Verfahren.

Abbildung 7.34: Gesamtzeit zur Lösung des EQS-Problems „Langstabisolator mit 1 mm (Modell Nr. 6 – hier a) und b) ) bzw. 2 mm dickem Feldsteuerelement und Wassertropfen“ (Modell Nr. 7 – hier c) und d) ) mit Ansatzfunktionen zweiter Ordnung.



(a) Vergleich GPU-beschleunigter Zeitintegratoren mit Referenz.

(b) Vergleich der Startwertschätzer für das RKC Verfahren.

Abbildung 7.35: Gesamtzeit zur Lösung des EQS-Problems „Langstabisolator mit maximaler Unbekanntenzahl“ (Modell Nr. 3) mit Ansatzfunktionen zweiter Ordnung.

#### 7.5.2.2 Vergleich der Beschleunigungswerte aller Modellbeispiele

Abb. 7.33 bis 7.35 zeigen die Rechenzeiten der Modelle Nr. 3 bis 7. Die vergleichende Betrachtung erfolgt analog zur Betrachtung der Ergebnisse für Ansatzfunktionen erster Ordnung. Hierzu ist der relative Geschwindigkeitsgewinn der Gesamtsimulation in Abb. 7.36 dargestellt. Die farbigen Balken geben die unterschiedlichen Modelle Nr. 2 bis 7 an, welche in der Legende erläutert werden. Der Buchstaben auf der x-Achse gibt den Zeitintegrationsansatz gem. Tab. 7.13 an.

Die Tendenz zur höheren Beschleunigung bei Nutzung von Ansatzfunktionen zweiter Ordnung durch Verwendung einer GPU lässt sich für alle Modelle bei Vergleich der Integrationsmethoden **A** und **B** bestätigen. Grund hierfür ist die höhere Beschleunigung im Bereich des PCG-Lösers (vgl. Kap. 5.1).

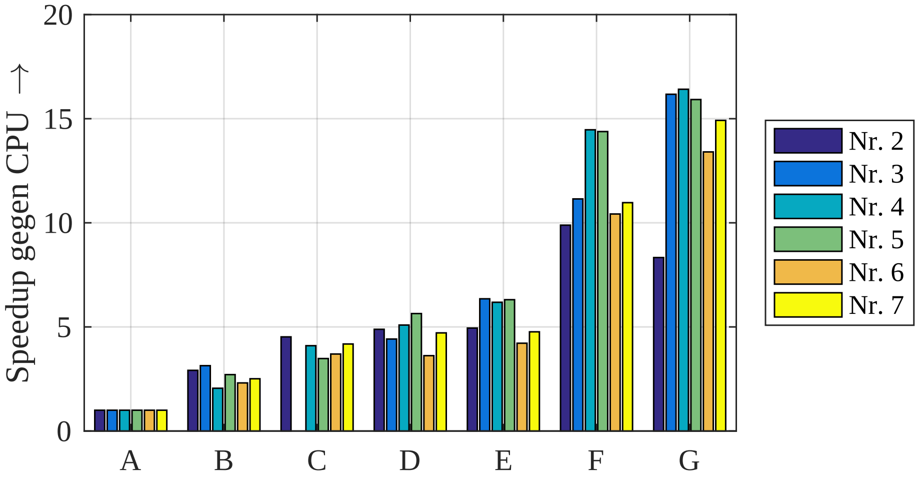


Abbildung 7.36: Beschleunigung der Rechendauer der EQS-Probleme für die Beispielmodelle Nr. 2 bis 7 mit Ansatzfunktionen zweiter Ordnung für Zeitintegrationsverfahren gem. Tab. 7.10.

| | Bezeichnung | Ansatz |
|---|---|---|
| **A** | TrilinosML | Implizites SDIRK3(2) mit CPU /Trilinos ML |
| **B** | implizit 1 | SDIRK3(2) mit Multi-GPU-PCG (vgl. Kap. 5.1) |
| **C** | implizit 2 | **B** mit adaptivem Vorkonditionierer (vgl. Kap. 6.1) |
| **D** | explizit RKC | Expliziter RKC-Zeitintegrator (vgl. Kap. 6.2) |
| **E** | explizit SPE | **D** mit SPE aus $\mathbf{f}_r,\ r = i - 10, .., i$ (vgl. Kap. 6.2.2) |
| **F** | explizit MOR | **D** mit MOR aus $\mathbf{f}_r,\ r = 1, .., 20$ (vgl. Kap. 6.2.3) |
| **G** | explizit MOR2 | **D** mit MOR mit $\mathbf{f}_r,\ r = i - 20, .., i$ (vgl. Kap. 6.2.3) |

Tabelle 7.13: Zu vergleichende Zeitintegrationsansätze. Ansätze **E** bis **G** berechnen einen Startwert für das PCG-Verfahren im Zeitschritt $t_i \to t_{i+1}$

Zudem wird ein höherer Anteil der Rechenzeit eines Zeitschrittes auf den PCG-Löser verwandt, als dies bei Ansatzfunktionen erster Ordnung der Fall ist.

Auch der weitere Geschwindigkeitsgewinn in Methode **C** lässt sich für alle Modelle bestätigen. Hier ist herauszustellen, das alle Modelle eine wesentlich erhöhte Anzahl an Iterationen aufweisen, was bei den Ansatzfunktionen erster Ordnung so nicht zu sehen war. Zusätzlich kann das Modell Nr. 3 mit der Methode nicht gelöst werden, da der PCG-Löser an einem späteren

Zeitschritt im Simulationsverlauf das Residuum nicht unter den geforderten Toleranzwert reduzieren kann.

Bei allen Modellen ist das Verhalten für das expliziten RKC-Verfahren ohne Startwertschätzer **D** und die Startwertextrapolation mit SPE (Ansatz **E**) qualitativ gleich. Mit der expliziten Zeitintegration ist nur ein geringer Geschwindigkeitsgewinn zu erreichen. Die SPE verursacht, analog zu Modell Nr. 2, nahezu keinen Geschwindigkeitsgewinn in den Modellen Nr. 6 und 7. Ein leichter Geschwindigkeitsgewinn ist für die Modelle Nr. 3 bis 5 bei Anwendung der SPE Startwertextrapolation feststellbar.

Der deutliche Geschwindigkeitsgewinn durch die Verwendung des MOR-Startwertschätzers, welcher für das Modell Nr. 2 beobachtet wurde, ist in Abb. 7.36 bei Methode **F** für alle Modelle zu bestätigen. Für die anderen Beispielmodelle werden noch höhere Beschleunigungswerte erreicht. Dies ist mit der höheren Anzahl an Freiheitsgraden zu erklären. Hierdurch steigt der relative Anteil des PCG-Lösers an der Gesamtrechenzeit und eine Reduzierung der Iterationszahl hat höhere Auswirkungen auf die Gesamtrechendauer.

Die Nutzung der MOR-Startwertextrapolation mit den Ergebnissen folgender Snapshotmatrix (**G**) führt zu einer noch weiter steigenden Beschleunigung. Mit einem Faktor von 16,41 werden hier die besten Beschleunigungswerte für eine Gesamtsimulation erreicht. Eine Ausnahme stellt das Modell Nr. 2 dar, das bereits in der Einzelbetrachtung erörtert wurde.

| | CPU | GPU | | | | | |
|---|---|---|---|---|---|---|---|
| | Implizit | | | Explizit | | | |
| | **A** | **B** | **C** | **D** | **E** | **F** | **G** |
| Modell Nr. 2 | | | | | | | |
| Rechenzeit [$10^3$ s ] | 192,00 | 65,88 | 42,47 | 39,30 | 38,84 | 19,42 | 23,05 |
| Beschleunigung | 1,00 | 2,91 | 4,52 | 4,89 | 4,94 | **9,89** | 8,33 |
| CG Iterationen | 24709 | 22958 | 38650 | 78558 | 77556 | 12662 | 15224 |
| Modell Nr. 3 | | | | | | | |
| Rechenzeit [$10^3$ s] | 475,40 | 151,45 | – | 107,60 | 74,86 | 42,65 | 29,39 |
| Beschleunigung | 1,00 | 3,14 | – | 4,42 | 6,35 | 11,15 | **16,41** |
| CG Iterationen | 17057 | 20001 | – | 45843 | 29364 | 15816 | 7688 |
| Modell Nr. 4 | | | | | | | |
| Rechenzeit [$10^3$ s] | 284,13 | 138,24 | 69,33 | 55,82 | 45,95 | 19,64 | 17,31 |
| Beschleunigung | 1,00 | 2,06 | 4,1 | 5,09 | 6,18 | 14,47 | **16,41** |
| CG Iterationen | 16875 | 17253 | 35862 | 66799 | 27487 | 10045 | 5651 |
| Modell Nr. 5 | | | | | | | |
| Rechenzeit [$10^3$ s] | 292,64 | 107,99 | 84,05 | 51,87 | 46,39 | 20,35 | 18,38 |
| Beschleunigung | 1,00 | 2,71 | 3,48 | 5,64 | 6,3 | 14,38 | **15,92** |
| CG Iterationen | 18301 | 18908 | 39249 | 35147 | 30153 | 11532 | 6825 |
| Modell Nr. 6 | | | | | | | |
| Rechenzeit [$10^3$ s] | 389,38 | 168,50 | 105,27 | 107,58 | 92,33 | 37,36 | 29,05 |
| Beschleunigung | 1,00 | 2,31 | 3,7 | 3,62 | 4,21 | 10,42 | **13,40** |
| CG Iterationen | 22544 | 26872 | 33431 | 59692 | 50400 | 17226 | 9457 |
| Modell Nr. 7 | | | | | | | |
| Rechenzeit [$10^3$ s] | 21284 | 10493 | 5242 | 4208 | 1355 | 1735 | 1968 |
| Beschleunigung | 1 | 2,51 | 4,18 | 4,71 | 4,76 | 10,97 | **14,92** |
| CG Iterationen | 17369 | 20541 | 37862 | 72428 | 71560 | 14946 | 6848 |

Tabelle 7.14: Übersicht der Rechenzeit, Beschleunigung und der Summe aller CG-Iterationen für die Zeitintegrationsvarianten gem. Tab. 7.10 und die Beispielmodelle Nr. 2 bis 7.

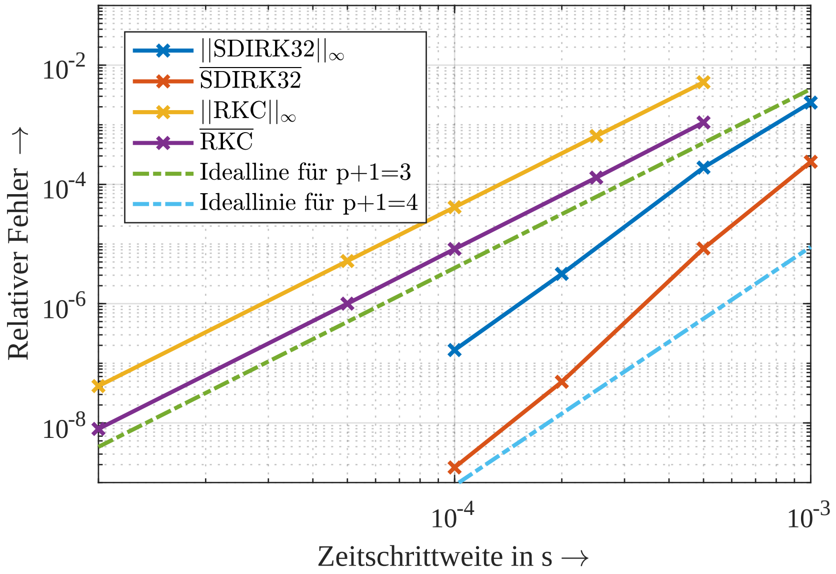
### 7.5.3 Zeitintegrationsfehler

Neben der betrachteten Rechendauer ist ein anderer Faktor numerischer Simulation die Genauigkeit. Ein Vergleich von Zeitintegrationsverfahren hinsichtlich Rechenzeit ist nur sinnvoll, wenn mit ihnen Ergebnisse vergleichbarer Genauigkeit erzielt werden. Der Fehler einer Simulation kann in einen Zeitdiskretisierungsfehler und einen Ortsdiskretisierungsfehler unterteilt werden. Der Ortsdiskrtisierungsfehler von MEQSICO wurde in [29] umfassend untersucht. Hier wird sich daher auf eine Analyse des Zeitintegrationsfehlers beschränkt, da in dieser Arbeit ein neues Zeitintegrationsschema vorgestellt wird, wohingegen die Ortsdiskretisierung unverändert für ein Modell beibehalten wird. Hauptziel dieser Arbeit ist die Beschleunigung der Berechnung elektroquasistatischer Probleme. Für die Zeitintegration wurde dies in den Abschnitten 7.5.1 und 7.5.2 betrachtet. Hinsichtlich der Genauigkeit wurde eine obere Schranke aufgestellt, welche kein Zeitschritt überschreiten durfte. Diese wurde mittels Fehlerschätzer während der Simulation kontinuierlich ermittelt. Eine genauere Untersuchung des Zeitintegrationsfehlers für das implizite SDIRK3(2)-Verfahren und das explizite RKC-Verfahren soll im Folgenden die Genauigkeit der Verfahren abhängig von der Zeitschrittweite aufzeigen.

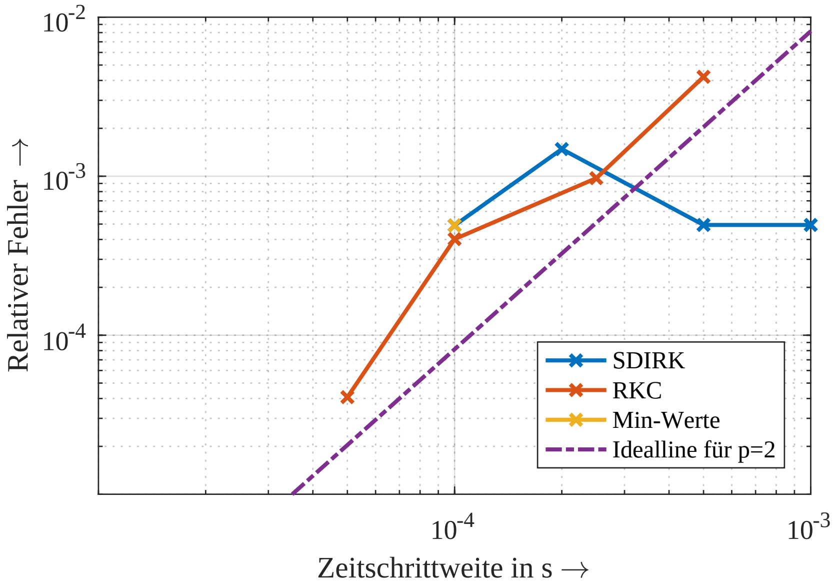
Hierzu werden die Anwendungsbeispiele mit variierender Zeitschrittweite berechnet. Für das SDIRK3(2)-Verfahren wird neben der in Kap. 7.5.1 und 7.5.2 verwendeten Zeitschrittweite von 0,5 ms die Berechnung mit Schrittweiten von 1 ms, 0,2 ms und 0,1 ms durchgeführt. Für das RKC-Verfahren wird neben der in Kap. 7.5.1 und 7.5.2 verwendeten Zeitschrittweite von 0,25 ms die Berechnung mit Schrittweiten von 0,5 ms, 0,1 ms, 0,05 ms und 0,01 ms durchgeführt. Der globale Zeitintegrationsfehler $e_g$ verhält sich bei einem Verfahren der Ordnung $p$ bei Veränderung der Zeitschrittweite $\Delta t$ um den Faktor $a$ mit $a\,\Delta t \sim a^p\, e_g$ [13]. Der lokale Fehler $e$ verhält mit $a\,\Delta t \sim a^{p+1}\, e$ [13] [55].

Analog zur Untersuchung der Rechendauer einer Simulation in Kap. 7.5.1 und 7.5.2 erfolgt eine detaillierte Diskussion am Anwendungsbeispiel der Hochspannungsdurchführung (Modell Nr. 2). Anschließend werden die Ergebnisse für die Modelle Nr. 3 bis 7 dargestellt. In Abb. 7.37a sind der maximale Wert und der Mittelwert des geschätzten lokalen Zeitintegrationsfehlers über der Zeitschrittweite aufgetragen. Das RKC-Verfahren ist ein Verfahren der Ordnung $p = 2$ (vgl. Kap. 2.4.2, [56, 43]), das SDIRK3(2)-Verfahren ist ein Verfahren der Ordnung $p = 3$. Dementsprechend sind Ideallinien p+1=3 und p+1=4 in Abb. 7.37a eingetragen.

Es ist erkennbar, dass der relative Fehler mit kleiner werdender Schrittweite parallel zur jeweiligen Ideallinie sinkt. Der mittlere geschätzte Fehler



(a) Lokaler Zeitintegrationfehler laut Fehlerschätzer in $||\cdot||_\infty$-Norm und Mittelwert für alle Zeitschritte $0.2\,\mathrm{s} \leq t \leq 0.4\,\mathrm{s}$ für SDIRK3(2)- und RKC-Verfahren



(b) Globaler Zeitintegrationsfehler am Zeitpunkt $t = 0,25\ s$. Die Lösung $\mathbf{u}_{\mathrm{RKC},min}$ mit $\Delta t = 10^{-5}\,\mathrm{s}$ wird als korrekte Lösung angenommen. SDIRK $= \frac{||\mathbf{u}_{\mathrm{SDIRK}} - \mathbf{u}_{\mathrm{SDIRK},min}||}{||\mathbf{u}_{\mathrm{SDIRK},min}||}$; RKC $= \frac{||\mathbf{u}_{\mathrm{RKC}} - \mathbf{u}_{\mathrm{RKC},min}||}{||\mathbf{u}_{\mathrm{RKC},min}||}$; Min-Werte $= \frac{||\mathbf{u}_{\mathrm{RKC},min} - \mathbf{u}_{\mathrm{SDIRK},min}||}{||\mathbf{u}_{\mathrm{RKC},min}||}$

Abbildung 7.37: Modell Nr. 2 - der Hochspannungsdurchführung nimmt schneller ab als der maximale Zeitschrittfehler. Ein weiteres Charakteristikum ist das Abflachen der Kurven des SDIRK3(2)-Verfahrens zwischen den Zeitschrittweiten 0,5 ms und 1 ms. Hier gewinnt der Fehler, welcher durch die Nichtlinearität verursacht wird, einen deutlichen Einfluss. So ist hier auch eine deutliche Steigerung der notwendigen Newton-Raphson-Iterationen zu beobachten. Dies führt zu einer längeren Berechnungszeit des impliziten SDIRK3(2)-Verfahrens für die Zeitschrittweite von 1 ms, als dies bei Zeitschritten von 0,5 ms der Fall ist.Wie zu erwarten sind für das explizite RKC-Verfahren zum Erreichen gleicher Genauigkeit eine kleinere Zeitschrittweite und damit deutlich mehr Zeitschritte erforderlich. Bei höheren Genauigkeitsanforderungen weitet sich dabei der Abstand der erforderlichen Zeitschrittweite auf, da das RKC-Verfahren eine geringere Ordnung $p$ hat als das implizite SDIRK3(2)-Verfahren. Weiterhin ist ein geringerer Abstand zwischen $||\cdot||_\infty$-Norm und Mittelwert des Zeitintegrationsfehlers für das RKC-Verfahren festzustellen. Dies ist insbesondere damit zu begründen, dass die Varianz zwischen Extremwerten und dem Großteil der Werte kleiner ist, der Zeitintegrationsfehler also relativ konstant gehalten wird. Im SDIRK3(2)-Verfahren steigt der Zeitintegrationsfehler dagegen insbesondere in dem Bereich an, in dem das nichtlineare Mikrovaristormaterial aussteuert, während er über weite Teile, in denen das nichtlineare Material praktisch nicht leitfähig ist, sehr gering ist.

Um die Qualität der Fehlerschätzung zu ermitteln, ist der globale Zeitintegrationsfehler auszuwerten. Hierzu wird das Ergebnis der jeweils kleinsten Schrittweite $\mathbf{u}_{min}$ eines Verfahrens als korrektes Ergebnis angenommen. Der globale Zeitintegrationsfehler ergibt sich aus

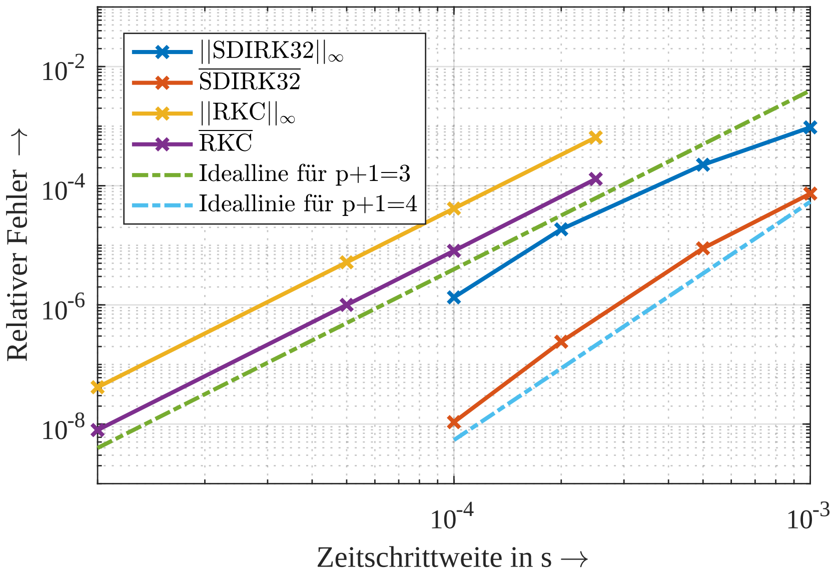
$$e_g(t) = \frac{||\mathbf{u}(t) - \mathbf{u}_{min}(t)||}{||\mathbf{u}_{min}(t)||}. \tag{7.3}$$

Es wird jeweils der Vektor $\mathbf{u}$ zum selben Zeitpunkt mit dem entspechenden Vektor $\mathbf{u}_{min}$ der kleinsten Schrittweite desselben Zeitintegrationsverfahrens verglichen. So zeigt sich, dass innerhalb des Verfahrens der Fehler bei kleiner werdendem $\Delta t$ reduziert wird. Konstante Abweichung durch unterschiedliche Verfahren werden so ausgeschlossen. Der Vergleich der Verfahren erfolgt über die Anwendung von (7.3) auf die Ergebnisse mit der jeweils kleinsten

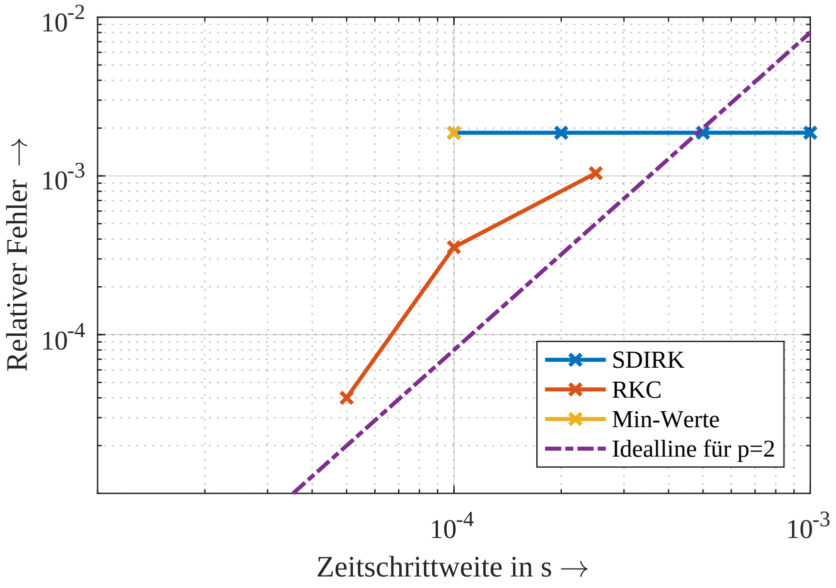
Zeitschrittweite. Bei vergleichbaren Ergebnissen setzt dieser Punkt dann die Linie des fehlerhafteren Verfahrens fort.

Abb. 7.37b zeigt den tatsächlichen globalen Zeitintegrationsfehler für das Beispielmodell Nr. 2. Für das RKC-Verfahren zeigt sich eine parallel zur Ideallinie für Verfahren der Ordnung $p = 2$ Linie. Es verhält sich also wie erwartet. Das SDIRK3(2)-Verfahren zeigt hingegen mit kleiner werdendem Zeitschritt keine Fehlerreduktion. Grund hierfür ist die Nichtlinearität und ihre Auswertung über die sukzessive Linearisierung mittels Newton-Raphson-Verfahren. Dies ist insofern bemerkenswert, als dass der Fehlerschätzer eine Reduktion des Zeitintegrationsfehlers angibt, welche so offensichtliche nicht stattfindet. Der konstante globale Fehler des SDIRK3(2)-Verfahrens wird durch die Auswertung von (7.3) bezüglich der beiden minimalen Schrittweiten der jeweiligen Zeitintegrationsverfahren fortgesetzt. Somit liefern die Verfahren vergleichbare Ergebnisse.

Die auftretenden lokalen und globalen Zeitintegrationsfehler sind in analoger Weise in Abb. 7.38 bis 7.42 für die anderen zu betrachtenden Beispiele Nr. 3 bis 7 dargestellt. Ein vergleichbares Verhalten, wie das gerade für Modell Nr. 2 dargestellte, ist auch für diese Modelle zu beobachten. Auch die nicht vorhandene Fehlerreduktion des globalen Fehlers für das SDIRK3(2)-Verfahren bei kleiner werdender Schrittweite kann bei allen Beispielmodellen beobachtet werden und ist somit qualitativ nicht modellabhängig.

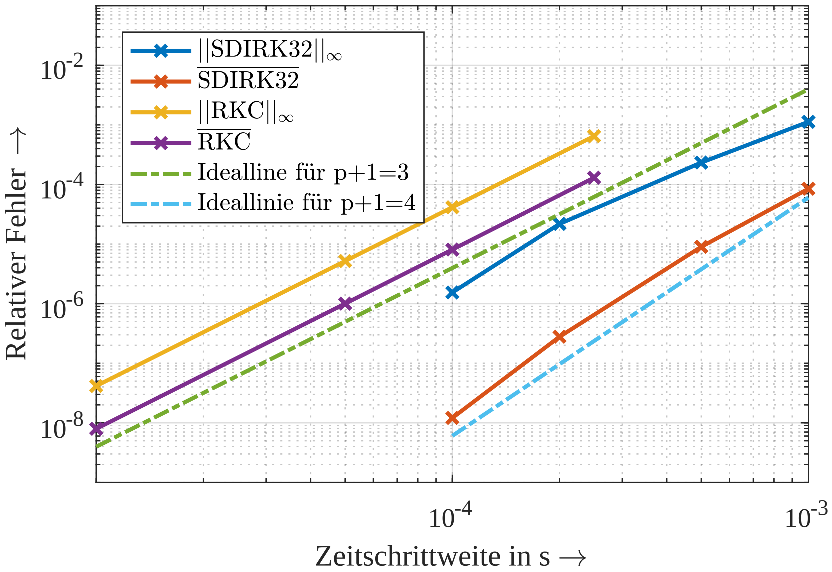


(a) Lokaler Zeitintegrationfehler laut Fehlerschätzer in $||\cdot||_\infty$-Norm und Mittelwert für alle Zeitschritte $0.2\,\text{s} \leq t \leq 0.4\,\text{s}$ für SDIRK3(2)- und RKC-Verfahren

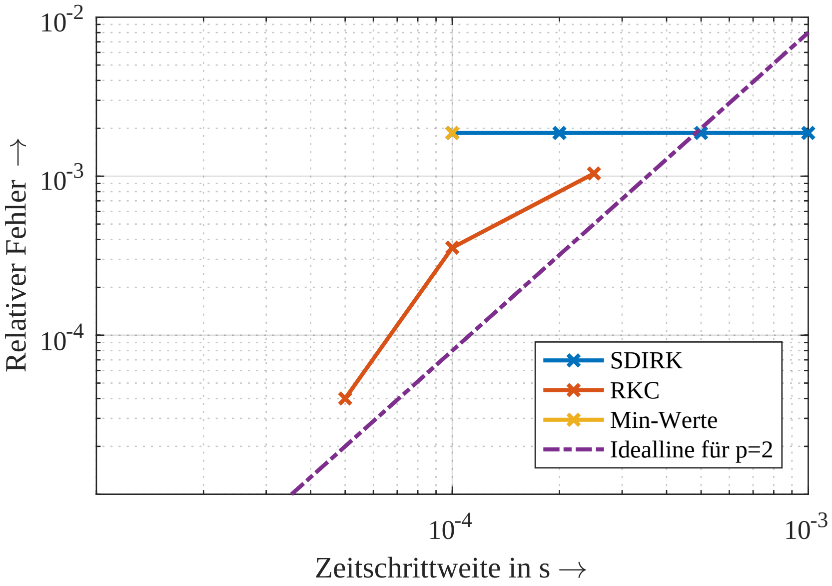


(b) Globaler Zeitintegrationsfehler am Zeitpunkt $t = 0,25\ s$. Die Lösung $\mathbf{u}_{\text{RKC},min}$ mit $\Delta t = 10^{-5}\,\text{s}$ wird als korrekte Lösung angenommen. SDIRK $= \frac{||\mathbf{u}_{\text{SDIRK}} - \mathbf{u}_{\text{SDIRK},min}||}{||\mathbf{u}_{\text{SDIRK},min}||}$; RKC $= \frac{||\mathbf{u}_{\text{RKC}} - \mathbf{u}_{\text{RKC},min}||}{||\mathbf{u}_{\text{RKC},min}||}$; Min-Werte $= \frac{||\mathbf{u}_{\text{RKC},min} - \mathbf{u}_{\text{SDIRK},min}||}{||\mathbf{u}_{\text{RKC},min}||}$

Abbildung 7.38: Modell Nr. 4 -Langstabisolator mit Mikrovaristorfeldsteuerelement.

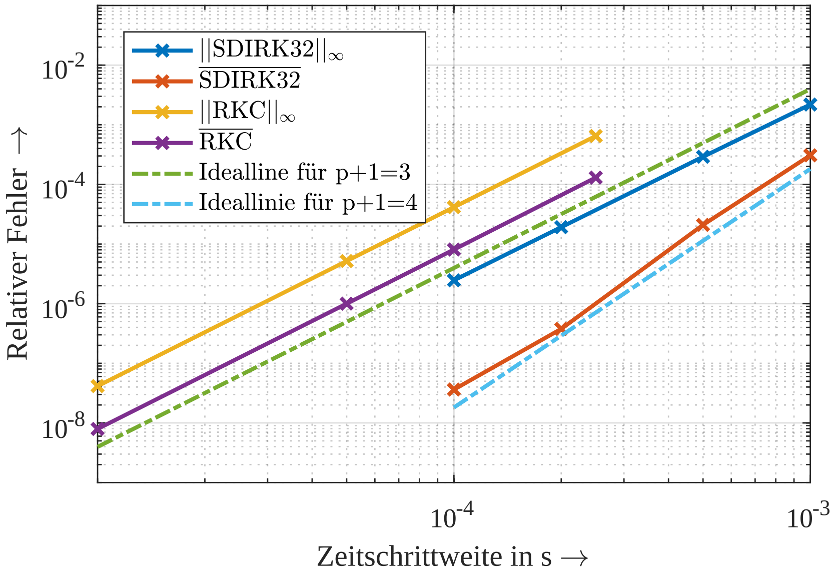


(a) Lokaler Zeitintegrationfehler laut Fehlerschätzer in $||\cdot||_\infty$-Norm und Mittelwert für alle Zeitschritte $0.2\,\mathrm{s} \leq t \leq 0.4\,\mathrm{s}$ für SDIRK3(2)- und RKC-Verfahren

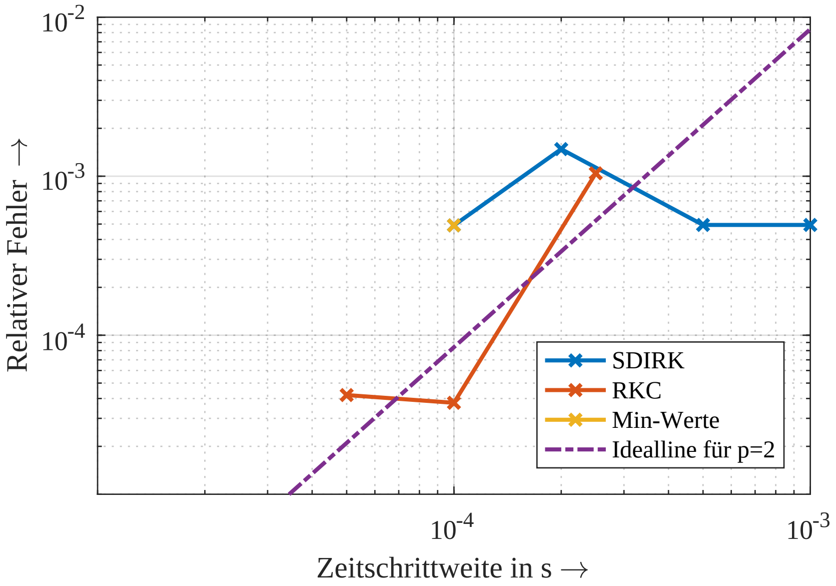


(b) Globaler Zeitintegrationsfehler am Zeitpunkt $t = 0,25\ s$. Die Lösung $\mathbf{u}_{\mathrm{RKC},min}$ mit $\Delta t = 10^{-5}\,\mathrm{s}$ wird als korrekte Lösung angenommen. SDIRK $= \frac{||\mathbf{u}_{\mathrm{SDIRK}} - \mathbf{u}_{\mathrm{SDIRK},min}||}{||\mathbf{u}_{\mathrm{SDIRK},min}||}$; RKC $= \frac{||\mathbf{u}_{\mathrm{RKC}} - \mathbf{u}_{\mathrm{RKC},min}||}{||\mathbf{u}_{\mathrm{RKC},min}||}$; Min-Werte $= \frac{||\mathbf{u}_{\mathrm{RKC},min} - \mathbf{u}_{\mathrm{SDIRK},min}||}{||\mathbf{u}_{\mathrm{RKC},min}||}$

Abbildung 7.39: Modell Nr. 5 - Langstabisolator mit Mikrovaristorfeldsteuerelement.

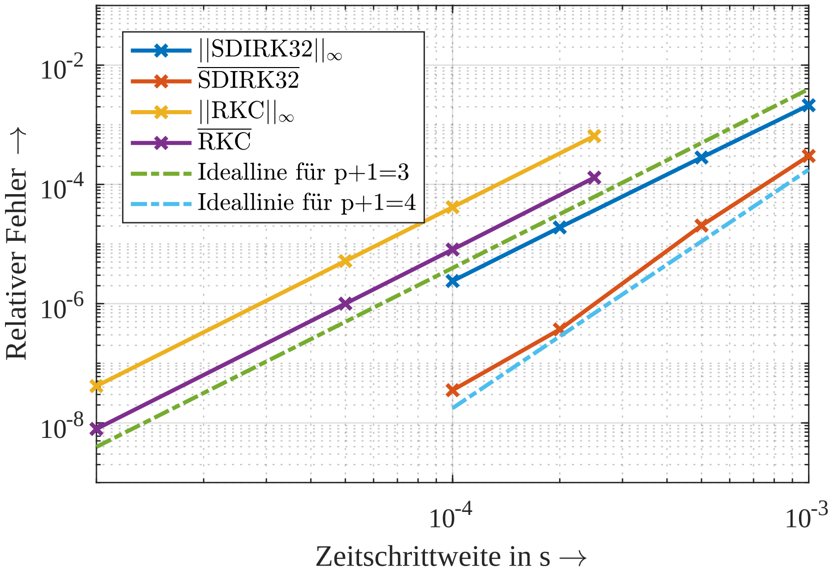


(a) Lokaler Zeitintegrationfehler laut Fehlerschätzer in $||\cdot||_\infty$-Norm und Mittelwert für alle Zeitschritte $0.2\,\text{s} \leq t \leq 0.4\,\text{s}$ für SDIRK3(2)- und RKC-Verfahren

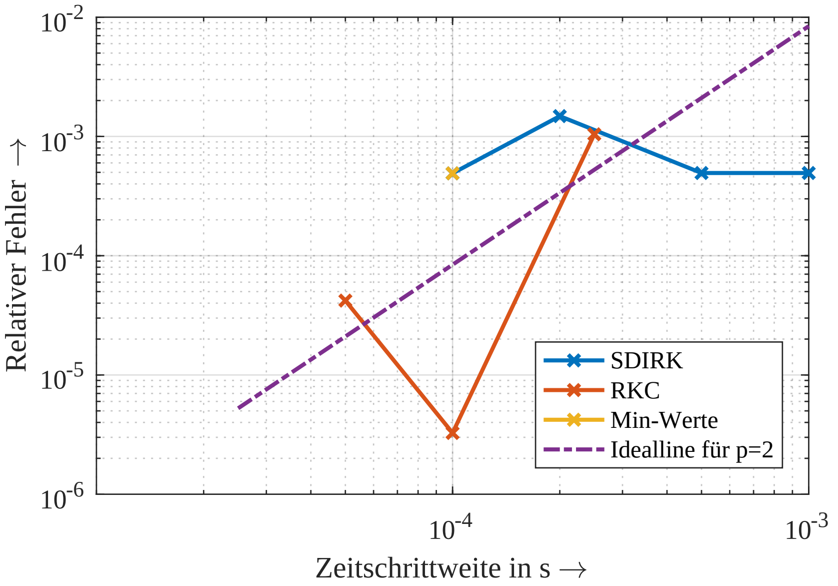


(b) Globaler Zeitintegrationsfehler am Zeitpunkt $t = 0,25\ s$. Die Lösung $\mathbf{u}_{\text{RKC},min}$ mit $\Delta t = 10^{-5}$ s wird als korrekte Lösung angenommen. SDIRK $= \frac{||\mathbf{u}_{\text{SDIRK}} - \mathbf{u}_{\text{SDIRK},min}||}{||\mathbf{u}_{\text{SDIRK},min}||}$; RKC $= \frac{||\mathbf{u}_{\text{RKC}} - \mathbf{u}_{\text{RKC},min}||}{||\mathbf{u}_{\text{RKC},min}||}$; Min-Werte $= \frac{||\mathbf{u}_{\text{RKC},min} - \mathbf{u}_{\text{SDIRK},min}||}{||\mathbf{u}_{\text{RKC},min}||}$

Abbildung 7.40: Modell Nr. 6 - Langstabisolator mit 1mm dickem Mikrovaristorfeldsteuerelement und Tropfen.

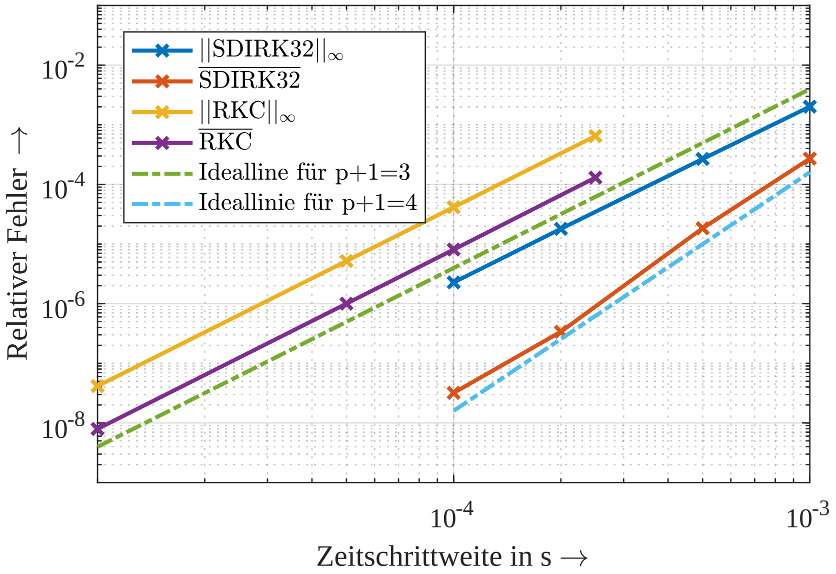


(a) Lokaler Zeitintegrationfehler laut Fehlerschätzer in $||\cdot||_\infty$-Norm und Mittelwert für alle Zeitschritte $0.2\,\text{s} \leq t \leq 0.4\,\text{s}$ für SDIRK3(2)- und RKC-Verfahren

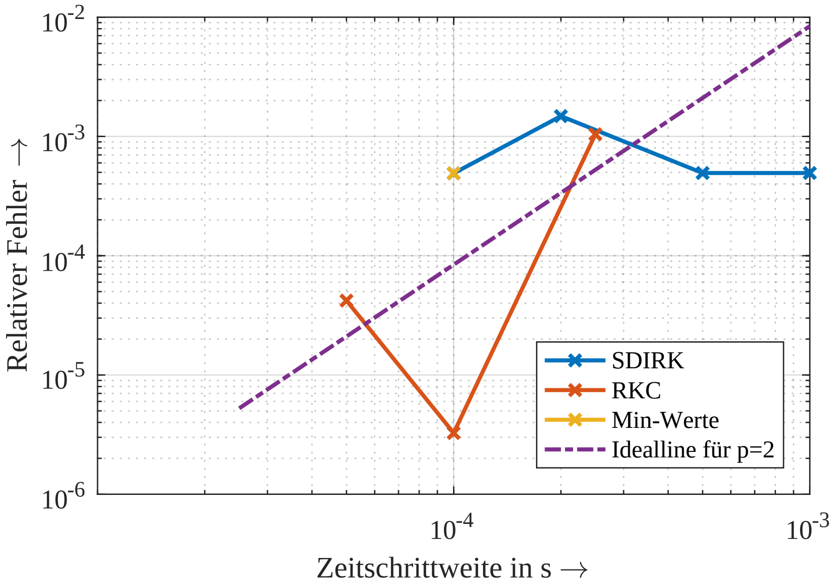


(b) Globaler Zeitintegrationsfehler am Zeitpunkt $t = 0,25\ s$. Die Lösung $\mathbf{u}_{\text{RKC},min}$ mit $\Delta t = 10^{-5}\,\text{s}$ wird als korrekte Lösung angenommen. SDIRK $= \frac{||\mathbf{u}_{\text{SDIRK}} - \mathbf{u}_{\text{SDIRK},min}||}{||\mathbf{u}_{\text{SDIRK},min}||}$; RKC $= \frac{||\mathbf{u}_{\text{RKC}} - \mathbf{u}_{\text{RKC},min}||}{||\mathbf{u}_{\text{RKC},min}||}$; Min-Werte $= \frac{||\mathbf{u}_{\text{RKC},min} - \mathbf{u}_{\text{SDIRK},min}||}{||\mathbf{u}_{\text{RKC},min}||}$

Abbildung 7.41: Modell Nr. 7 - Langstabisolator mit 2 mm dickem Mikrovaristorfeldsteuerelement und Tropfen.



(a) Lokaler Zeitintegrationfehler laut Fehlerschätzer in $||\cdot||_\infty$-Norm und Mittelwert für alle Zeitschritte $0.2\,\mathrm{s} \leq t \leq 0.4\,\mathrm{s}$ für SDIRK3(2)- und RKC-Verfahren



(b) Globaler Zeitintegrationsfehler am Zeitpunkt $t = 0,25$ $s$. Die Lösung $\mathbf{u}_{\mathrm{RKC},min}$ mit $\Delta t = 10^{-5}\,\mathrm{s}$ wird als korrekte Lösung angenommen. SDIRK $= \frac{||\mathbf{u}_{\mathrm{SDIRK}} - \mathbf{u}_{\mathrm{SDIRK},min}||}{||\mathbf{u}_{\mathrm{SDIRK},min}||}$; RKC $= \frac{||\mathbf{u}_{\mathrm{RKC}} - \mathbf{u}_{\mathrm{RKC},min}||}{||\mathbf{u}_{\mathrm{RKC},min}||}$; Min-Werte $= \frac{||\mathbf{u}_{\mathrm{RKC},min} - \mathbf{u}_{\mathrm{SDIRK},min}||}{||\mathbf{u}_{\mathrm{RKC},min}||}$

Abbildung 7.42: Modell Nr. 3 -Langstabisolator mit maximaler Unbekanntenzahl.

## 7.6 Zusammenfassung

Die in den Kap. 5 und Kap. 6 vorgestellten Ansätze werden anhand von Anwendungsbeispielen der Hochspannungstechnik verifiziert.

Anhand dieser Anwendungsbeispiele wird der Multi-GPU-beschleunigte CG-Gleichungssystemlöser hinsichtlich Konvergenzverhalten, Speicherbedarf und Rechengeschwindigkeit für eine variierende Anzahl an GPUs und gegenüber einer parallelisierten CPU-basierten Referenzimplementation überprüft. Hier zeigen sich Geschwindigkeitsgewinne um bis zu mehr als das elffache des Referenzlösungsverfahrens.

Anschließend wird die GPU-basierte Kombination aus Assemblierung und SpMV für die nichtlinearen Anwendungsbeispiele mit der ebenfalls parallelisierten CPU-Referenz verglichen. Hier sind Geschwindigkeitsgewinne um einen Faktor >500 aufzeigbar.

Abschließend werden die Gesamtsimulationszeiten für die nichtlinearen Anwendungsbeispiele untersucht. Im Einzelnen werden ein implizites SDIRK3(2)-Verfahren mit Multi-GPU beschleunigtem AMG-CG-Verfahren sowie mit zusätzlich adaptivem Vorkonditionierer gegenüber der rein CPU-basierten Referenzlösung verglichen. Diesem wird eine hinsichtlich Zeitintegrationsfehler vergleichbares semi-explizites RKC-Verfahren gegenübergestellt, welches in weiteren Varianten durch Startwertschätzer, basierend auf SPE und MOR, erweitert wird.

Mit FEM-Ansatzfunktionen erster Ordnung wird die Rechendauer um einem Faktor 18 verkürzt. Hierbei zeigt sich, dass ein RKC-Verfahren mit Startwertschätzung mittels SPE die geringsten Rechenzeiten benötigt. Für FEM-Ansatzfunktionen zweiter Ordnung wird die Simulationszeit um einen Faktor 16 gegenüber der Referenzlösung reduziert. Der Ansatz eines RKC-Verfahrens mit Startwertschätzung durch POD-MOR ist bei den meisten Beispielen das Verfahren, welches die geringste Rechenzeit benötigt. Der Zeitintegrationsfehler für das implizite und das explizite Zeitintegrationsverfahren wird bei variierender Zeitschrittweite untersucht.

# Kapitel 8

# Zusammenfassung

Im Rahmen dieser Arbeit wurde eine neuartige Kombination aus mathematischen Algorithmen, Softwareimplementierungen und Rechnerarchitekturen vorgeschlagen um die numerische Simulation nichtlinearer, elektroquasistatischer Probleme zu beschleunigen. So können Betriebsmittel der Hochspannungstechnik geometrisch optimiert werden, welche nichtlinear leitfähige Materialien zur Feldsteuerung einsetzen. Für diese Optimierung ist die Rechnung und Auswertung einer Reihe von Modellvarianten erforderlich.

Die erste, eingangs formulierte Forschungsfrage fordert die Untersuchung neuer technischer Möglichkeiten zur Beschleunigung des Rechenvorgangs einer elektroquasistatischen Simulation. Technisch wird eine Beschleunigung durch massive Parallelisierung rechenintensiver Prozesse mit Grafikkarten erreicht, da diese um eine vielfach höhere Anzahl an Recheneinheiten sowie eine breitbandigere Speicheranbindung verfügen als es bei CPUs der Fall ist. Portiert werden zum einen der Löser für das lineare Gleichungssystem und zum anderen die Matrixassemblierung.

Es werden das Matrixformat zur Speicherung der dünn-besetzten Matrizen und die Durchführung von Matrix-Vektor-Multiplikation mit dünnbesetzter Matrix für die Berechnung aus GPUs optimiert [B4]. Ein wesentlicher Faktor beim iterativen Lösen linearer Gleichungssysteme mit CG-Verfahren ist die Wahl und Implementierung des Vorkonditionierer. Dieser nutzt das algebraische Mehrgitterverfahren, welches auf GPU implementiert und für die wiederholte Lösung in elektroquasistatischen Problemen erweitert wird [B5]. Zusätzlich werden Mixed-Precision-Ansätze untersucht, welche den AMG-Vorkonditionierer in einfacher Zahlengenauigkeit ausführen. Dies reduziert den Speicherbedarf und erhöht die Berechnungsgeschwindigkeit [B6].

Es wird eine Implementation des iterativen Lösers für lineare Gleichungssysteme vorgestellt, welche die Berechnungen auf mehreren GPUs parallel ausführt. Damit wird die Speicherbeschränkung aufgrund des für große Probleme zu kleinen GPU-Speichers umgangen und eine weitere Beschleunigung erreicht. Hier werden auf mehrere GPUs verteilte Matrix- und Vektor-Klassen, Kommunikationsroutinen und Matrix-Vektor-Operationen entwickelt und im iterativen Löser sowie dem AMG-Vorkonditionierer eingesetzt [B7, B8, B9]. Durch Einsatz von GPU-basierter Parallelisierung kann mit den verwendeten Systemen eine Beschleunigung bis zu Faktor 25 gegenüber dem Löser für das lineare Gleichungssystem auf CPU-Basis erreicht werden [B11].

Aufgrund der nichtlinearen Leitfähigkeit der verwendeten Materialien muss die Leitwertmatrix häufig neu assembliert werden. Dieser „peinlich parallele“ Prozess wird daher auf die GPU portiert. Insbesondere durch Kombination der Matrix-Vektor-Multiplikation mit der Assemblierung kann eine hohe Beschleunigung dieses Teilprozesses auf einen Faktor >500 erreicht werden.

Die zweite Eingangs formulierte Forschungsfrage fordert die Untersuchung der Effekte auf die Simulationsteile, welche sich aus der Beschleunigung mittels Grafikkarten ergeben und die Evaluation neuer Methoden und Verfahren. Das implizite Zeitintegrationsverfahren SDIRK3(2) wird durch die Nutzung des GPU-beschleunigten linearen Lösers um einen Faktor zwei bis fünf beschleunigt. Durch einen vorgestellten Ansatz zur Wiederverwendung des AMG-Vorkonditionierung mit Anpassung an die Variation der Systemmatrix aufgrund nichtlinearer Leitfähigkeit kann die Rechenzeit zusätzlich erneut halbiert werden [B10].

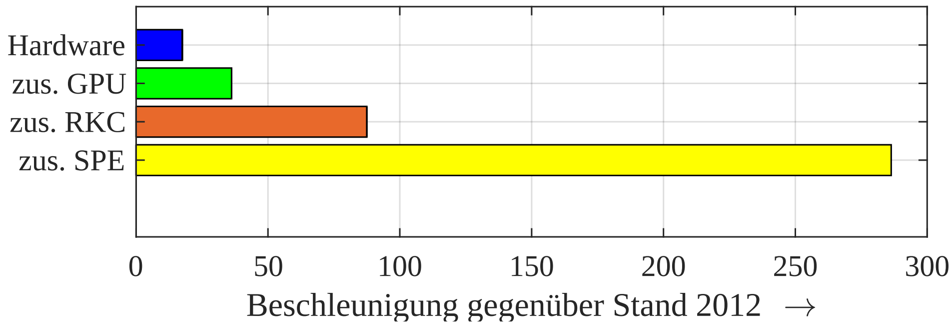
Umformulierung des diskreten transienten elektroquasistatischen Modells ergibt ein System gewöhlicher Differentialgleichungen, sodass die Anwendung von (semi-)expliziten Zeitintegratoren erstmalig untersucht wurde. Durch die schnelle Lösung des linearen Gleichungssystems sind (semi-)explizite

Zeitintegrationsverfahren für elektroquasistatische Probleme rentabel. Es wird ein neuer expliziter Zeitintegrationsansatz für elektroquasistaische Probleme auf Basis des Runge-Kutta-Chebychev-Verfahrens vorgestellt. Auch wenn explizite Zeitintegrationsverfahren wegen des Courant-Friedrich-Lewy-Stabilitätskriteriums eine deutlich höhere Anzahl an Zeitschritten benötigen, wird dieser Aufwand durch eine geringere Rechenzeit pro Zeitschritt kompensiert, so dass eine Beschleunigung um einen Faktor zwei bis vier erreicht wird. Die Form des Problems bei expliziter Zeitintegration als Multiple-Rechte-Seite-Problem macht es zudem möglich, Startwerte für die iterative Lösung des linearen Gleichungssystems zu extrapolieren. Hierzu werden Startwertschätzer basierend auf Subspace Projection Extrapolation und Proper-Orthogonal-Decomposition Modelordnungsreduktion genutzt. Durch diese Ansätze kann noch einmal ein Geschwindigkeitsgewinn um einen Faktor zwei bis vier erreicht werden [B1, B12, B13].

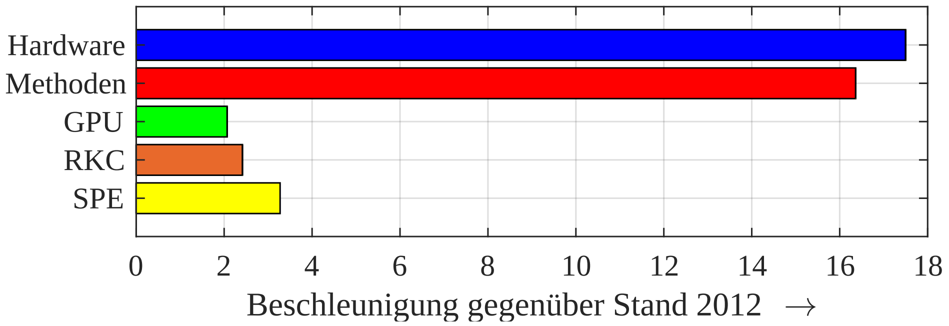
In der Summe kann so eine Beschleunigung einer transienten EQS-Simulation um einen Faktor 10 bis 30 gegenüber der vollparallelisierten EQS-Variante unter Verwendung von CPUs erreicht werden. Es werden insbesondere aufwendige Simulationen mit sehr hochdimensionalen Auflösungen beschleunigt, die sehr lange Rechenzeiten verursachen.

Zusätzlich zu dem methodenbasierten Fortschritt dieser Arbeit hat sich auch die verfügbare Rechenleistung erhöht. In der Einleitung wurde daher auf ein Modell verwiesen, welches zu Beginn der Bearbeitungszeit mit Ansatzfunktionen erster Ordnung 105 Stunden zur Lösung mit der damals zur Verfügung stehenden Hardware benötigte. Das gleiche Problem konnte unter Einsatz der in dieser Arbeit entwickelten Techniken zuletzt in 22 Minuten gelöst, was einer Beschleunigung der Gesamtrechenzeit um den Faktor 290 entspricht. Die Beschleunigungsanteile hierzu sind in Abb. 8.1 dargestellt.

Das Ziel dieser Arbeit war es laut der dritten, in der Einleitung formulierten Forschungsfrage, numerische Simulationen von Anwendungen der Hochspannungstechnik mit nichtlinear leitfähigen Materialien soweit zu beschleunigen, dass auch für große Modelle die Rechnung mehrerer Modellvariationen in akzeptabler Zeitdauer möglich wird. Dies ermöglicht die Optimierung dieser Modelle in Form und Funktion. Das Ziel einer Simulati-



(a) Geschwindigkeitsgewinn durch Kombination der Maßnahmen von oben nach unten. „zus.“ = zusätzlich.



(b) Geschwindigkeitsgewinn durch technischen und methodischen Fortschritt. „Methoden“ ergibt sich aus der Multiplikation des Geschwindigkeitsgewinns der darunter aufgeführten Einzelmaßnahmen.

Abbildung 8.1: Durch technischen Fortschritt und entwickelte Methoden erreichte Beschleunigung in der Bearbeitungszeit dieser Arbeit.

onsdauer von unter einer Stunde wird für alle betrachteten Modelle erreicht, welche jeweils über mehr als eine Millionen Freiheitsgrade verfügen. Mit den hier vorgestellten Verbesserungen ist es nun möglich, mehrere zehn bis hundert Modellvarianten dieser Größenordnung in einem Zeitraum zu berechnen, der zu Beginn dieser Arbeit für eine Variante erforderlich gewesen wäre.

# Ausblick

Die Ergebnisse dieser Arbeit eignen sich als Ausgangspunkt für eine weitere Beschleunigung numerischer Simulationen von elektroquasistatischen Feldverteilungen. Die Entwicklung von für allgemeine Berechnungen optimierte Grafikprozessoren mit bedeutend höheren Leistungsdaten schreitet voran. Insbesondere durch den Bedarf im Bereich der Künstliche-Intelligenz-Forschung kommt diesem Bereich in der Entwicklung von Computer-Hardware zur Zeit besondere Aufmerksamkeit zu Gute. Durch den im Oktober 2018 schnellsten Grafikprozessor für allgemeine Berechnungen, die Nvidia Tesla V100 [70], ist auf Basis der höheren Speicherbandbreite eine 3,75-fache Beschleunigung der aus die Grafikkarte ausgelagerten Prozesse zu erwarten.

Im Bereich der methodischen Entwicklung sind aktuelle Forschungsthemen eine weitere Parallelisierung von Berechnungen. In der Raumdiskretisierung sind hier Domain Decomposition Methoden zu nennen [140]. Bei der Parallelisierung in der Zeit sind Parareal-Ansätzen verstärkt Thema aktueller Forschung [141, 142, 143].

Wie diese Arbeit gezeigt hat, kann der lineare Gleichungssystemlöser signifikant durch Startwertgenerierung beschleunigt werden. Die notwendige Singulärwertzerlegung benötigt jedoch einen signifikanten Teil der Rechenzeit, insbesondere bei einer hohen Anzahl von auszuwertenden Lösungsvektoren. Hier versuchen Verfahren zur Snapshot-Auswahl und singulärwertzerlegungsfreie Verfahren diese Problematik zu mindern und letztendlich ganz zu umgehen [144, 145].

# Kapitel 9

# Anhang

## 9.1 Pseudocodes der verwendeten Funktionen

```
Lese Eingabedaten                ▷ Optionen, FEM-Gitterdaten, Materialdaten
Assembliere $\mathbf{M}_\varepsilon$
Setze Dirichlet-Randbedingungen
while t < $T$ do                 ▷ $T$ ist der Endzeitpunkt, Schritt $\mathbf{u}_i \rightarrow \mathbf{u}_{i+1}$
    for j = 1,..,4 do
        SDIRK_Stufe($\mathbf{u}_j$,$\mathbf{u}_{j+1}$,$j$,$t$,$\Delta t$)        ▷ siehe Alg. 9.2
    end for
    Berechne $\mathbf{u}_{i+1}$ aus Linearkombination (6.2)
    Berechne $e_{i+1}$ aus $||\mathbf{u}_{i+1} - \tilde{\mathbf{u}}_{i+1}||$
end while
Post-processing
```

Abbildung 9.1: Algorithmus von MEQSICO mit SDIRK3(2)-Zeitintegrator als Pseudocode.

```
function SDIRK-Stufe($\mathbf{u}_j$,$\mathbf{u}_{j+1}$,$j$,$t$,$\Delta t$)        ▷ Aus $\mathbf{u}_j$ wird $\mathbf{u}_{j+1}$ errechnet
    Assemblieren der nichtlinearen Matrix $\mathbf{K}_\kappa(\mathbf{u}_j)$
    Aufstellen der rechten Seite $\mathbf{rhs} = \mathbf{b}_{j+1} + \frac{1}{a_d\,\Delta t}\mathbf{M}_\varepsilon\,\mathbf{u}_j$:
    Aufstellen der Systemmatrix $\mathbf{A} = \left(\frac{1}{a_d\,\Delta t}\mathbf{M}_\varepsilon + \mathbf{K}_\kappa(\mathbf{u}_j)\right)$
    Setze $k = 0$, $r_{NL} = 1$        ▷ $k$: Newton-Iteration; $r_{NL}$: NL Residuum
    while $r_{NL} > tol_{NL}$ do
        Setup_AMG($\mathbf{A}$)        ▷ Wird in Kap. 6.1 nur einmal durchgeführt
        $\mathbf{u}_{j+1,k+1} \leftarrow$ CG-Verfahren($\mathbf{A}, \mathbf{rhs}, \mathbf{u}_{j+1,k}, AMG$)        ▷ siehe Alg. 9.7
        Assemblieren der Matrizen $\mathbf{K}_\kappa(\mathbf{u}_{j+1,k+1})$, $\vec{J}(\mathbf{u}_{j+1,k+1})$
        Aufstellen der rechten Seite $\mathbf{rhs} = \mathbf{b}_{j+1} + \frac{1}{a_d\,\Delta t}\mathbf{M}_\varepsilon\,\mathbf{u}_j$:
        Update der Systemmatrix $\mathbf{A} = \left(\frac{1}{a_d\,\Delta t}\mathbf{M}_\varepsilon + \mathbf{K}_\kappa(\mathbf{u}_{j+1,k+1})\right)$
        Ermitteln von $r_{NL} = ||\mathbf{A}\,\mathbf{u}_{j+1,k+1} - \mathbf{rhs}||$
        Update der rechten Seite: $\mathbf{rhs} = \mathbf{rhs} + \vec{J}(\mathbf{u}_{j+1,k+1})\,\mathbf{u}_{j+1,\underline{k}}$
        Update Systemmatrix: $\mathbf{A} = \mathbf{A} + \vec{J}(\mathbf{u}_{j+1,k+1})\,\mathbf{u}_{j+1,\underline{k+1}}$
        $k = k + 1$
    end while
    Berechne SDIRK-Stufenableitungsvektor mit (6.3)
end function
```

Abbildung 9.2: Algorithmus für einen Zeitschritt im SDIRK3(2)-Verfahren als Pseudocode.

```
Lese Eingabedaten        ▷ Optionen, FEM-Gitterdaten, Materialdaten
Assembliere $\mathbf{M}_\varepsilon$
Setze Dirichlet-Randbedingungen
Bestimme initale Stufenzahl $s$ des RKC-Verfahrens
Setup_AMG($\mathbf{M}_\varepsilon$)
while t < $T$ do        ▷ $T$ ist der Endzeitpunkt
    for j = 1,..,s do
        RKC_Schritt($\mathbf{u}_0$,$\mathbf{u}$,$j$,$t$,$\Delta t$)        ▷ siehe Alg. 9.4
        $\mathbf{u}_0 \leftarrow \mathbf{u}$
    end for
    Adaptiere_Stufenzahl(s)        ▷ siehe Alg. 9.6
end while
Post-processing
```

Abbildung 9.3: Algorithmus von MEQSICO mit RKC-Zeitintegrator als Pseudocode.

```
function RKC_SCHRITT($\mathbf{u}_0$,$\mathbf{u}$,$j$,$t$,$\Delta t$)    ▷ Aus $\mathbf{u}_0$ wird $\mathbf{u}$ errechnet
    Initialisiere Koeffizienten $\tilde{\mu}_1$
    $\mathbf{f0} \leftarrow f(t, \mathbf{u}_0)$    ▷ siehe Alg. 9.5
    $\mathbf{y1} \leftarrow \mathbf{u}_0 + \tilde{\mu}_1 \Delta t\, \mathbf{f0}$
    $\mathbf{ym1} \leftarrow \mathbf{y1}$   ;   $\mathbf{ym2} \leftarrow \mathbf{u}_0$
    for j=2,...,s do
        Aktualisiere Koeffizienten $\mu_j$, $\nu_j$, $\tilde{\mu}_j$, $\tilde{\gamma}_j$    ▷ siehe Kap. 2.4.2
        $\mathbf{fn} \leftarrow f(t + c_j\, \Delta t, \mathbf{ym1})$    ▷ siehe Alg. 9.5
        $\mathbf{y} \leftarrow (1 - \mu_j - \nu_j)\mathbf{u}_0 + \mu_j \mathbf{ym1} + \nu_j \mathbf{ym2} + \tilde{\mu}_j\, \Delta t\, \mathbf{fn} + \tilde{\gamma}_j\, \Delta t\, \mathbf{f0}$
        Verschiebe Vektoren: $\mathbf{ym2} \leftarrow \mathbf{ym1}$ ; $\mathbf{ym1} \leftarrow \mathbf{y}$
    end for
    $\mathbf{u} \leftarrow \mathbf{y}$
end function
```

Abbildung 9.4: Algorithmus für einen Zeitschritt im RKC-Verfahren als Pseudocode.

```
function $f(t,\mathbf{u})$    ▷ Löst $\mathbf{f}(t, \mathbf{u}) = \mathbf{M}_\varepsilon{}^{-1}\big(\mathbf{b}(t) - \mathbf{K}_\kappa(\mathbf{u})\mathbf{u}\big)$
    Setzte Dirichlet-Ränder $\mathbf{b}$
    $\mathbf{rhs} = \big(\mathbf{b}(t) - \mathbf{K}_\kappa(\mathbf{u})\mathbf{u}\big)$    ▷ GPU-Assemblierung nach Kap. 5.2
    $\mathbf{f}_0 \leftarrow$SPE / POD_MOR    ▷ Startwertextrapolation nach Kap. 6.2.2 f.
    $\mathbf{f} \leftarrow$ CG-Verfahren($\mathbf{M}_\varepsilon, \mathbf{rhs}, \mathbf{f}_0, AMG$)    ▷ siehe Alg. 9.7
    return f    ▷ Rückgabevektor f
end function
```

Abbildung 9.5: Algorithmus für die Zeitableitung der elektroquasistatischen Gleichung im RKC-Code als Pseudocode.

```
function ADAPTIERE_STUFENZAHL(s)
    Schätze Zeitintegrationsfehler $e$
    if $e > tol$ then                                  ▷ Zeitschritt wird verworfen
        Erhöhe Stufenzahl $s$
        if Zweiter verworfener Zeitschritt hintereinander then
            $t \leftarrow t - \Delta t$                ▷ Gehe zum vorherigen Zeitschritt
            Bestimme/Ändere Zeitschrittweite $\Delta t$
            Passe um1 aus gespeicherten Daten an
        end if
    else                                               ▷ Zeitschritt wird akzeptiert
        um1 ← u
        $t \leftarrow t + \Delta t$                    ▷ Gehe zum nächsten Zeitschritt
        Speichere u für Postprocessing
        if 10 akzeptierte Zeitschritte then
            Reduziere Stufenzahl $s$
        end if
        Füge fn zur Snapshotmatrix für SPE / POD-MOR hinzu
    end if
end function
```

Abbildung 9.6: Algorithmus zur heuristischen Anpassung der Stufenzahl im RKC-Code als Pseudocode

```
function CG-VERFAHREN($\mathbf{A}, \mathbf{b}, \mathbf{x}_0, AMG$)
    $\mathbf{x} \leftarrow \mathbf{x}_0$
    $\mathbf{r} \leftarrow \mathbf{b} - \mathbf{A}\,\mathbf{x}$
    $AMG(\mathbf{r}, \mathbf{z}, 1)$            ▷ Funktionsbeschreibung AMG-Lösungsphase s.u.
    $\mathbf{p} \leftarrow \mathbf{z}$
    $rr \leftarrow < \mathbf{r}, \mathbf{z} >$
    while (Residuum zu groß) do
        gather-nn ($\mathbf{p}$)
        $y \leftarrow \mathbf{A} * \mathbf{p}$
        $\alpha \leftarrow < \mathbf{r}, \mathbf{z} > / < \mathbf{y}, \mathbf{p} >$          ▷ Multi-GPU inneres Produkt
        $\mathbf{x} \leftarrow \mathbf{x} + \alpha * \mathbf{p}$
        $\mathbf{r} \leftarrow \mathbf{r} - \alpha * \mathbf{y}$
        gather-nn-async ($\mathbf{r}$)                    ▷ parallel zum AMG
        function AMG($\mathbf{r}$,$\mathbf{z}$,h=1)        ▷ h gibt den aktuellen AMG-Level an
            if $h = H$ then                 ▷ H gibt den maximalen AMG-Level an
                gather-nn ($\mathbf{r}_h$)
                $\mathbf{r}\_host \leftarrow \mathbf{r}_h[GPU0]$
                solve($\mathbf{r}\_host, \mathbf{z}\_host$)                ▷ Löse direkt
                copy-1n ($\mathbf{z}_h \leftarrow \mathbf{z}\_host$)
            else
                presmooth($\mathbf{z}_h$)
                gather-nn ($\mathbf{z}_h$)
                $\mathbf{res}_h \leftarrow \mathbf{r}_h - \mathbf{A}_h * \mathbf{z}_h$
                gather-nn ($\mathbf{res}_h$)
                $\mathbf{r}_{h+1} \leftarrow \mathbf{R}^h_{h+1} * \mathbf{res}_h$
                AMG($\mathbf{r}_{h+1}, \mathbf{z}_{h+1}, h + 1$)
                gather-nn ($\mathbf{z}_{h+1}$)
                $\mathbf{z}_h \leftarrow \mathbf{P}^{h+1}_h * \mathbf{z}_{h+1} + \mathbf{z}_h$
                gather-nn ($\mathbf{z}_h$)
                postsmooth($\mathbf{z}_h$)
            end if
        end function
        $rz_{alt} \leftarrow rz$
        $rz \leftarrow < \mathbf{r}, \mathbf{z} >$
        $\beta \leftarrow rz / rz_{alt}$
        $\mathbf{p} \leftarrow \mathbf{r} + \beta * \mathbf{p}$
    end while
    return $\mathbf{x}$                                ▷ Rückgabevektor $\mathbf{x}$
end function
```

Abbildung 9.7: Das Multi-GPU-AMG-CG-Verfahren mit Kommunikationsroutinen in Rot als Pseudocode.

# Literaturverzeichnis

[1] A. Küchler, *Hochspannungstechnik.* Berlin, Heidelberg: Springer, 2009.

[2] V. Hinrichsen, *Metalloxid-Ableiter in Hochspannungsnetzen*, 3. Aufl. Erlangen: Siemens AG, 2012.

[3] H. Ye, M. Clemens, und J. Seifert, „Electroquasistatic Field Simulation for the Layout Improvement of Outdoor Insulators Using Microvaristor Material," *IEEE Transactions on Magnetics*, Vol. 49, Nr. 5, S. 1709–1712, 2013.

[4] D. A. Reed, „Computational Science: Ensuring America's Competitiveness," President's Information Technology Advisory Committee, Tech. Rep., 2005.

[5] G. E. Moore, „Cramming More Components Onto Integrated Circuits," *Electronics*, Vol. 38, Nr. 8, S. 82–85, 1965.

[6] J. C. Maxwell, „A Dynamical Theory of the Electromagnetic Field," *Philosophical Transactions of the Royal Society of London*, Vol. 155, S. 459–512, 1865.

[7] H. Klingbeil, *Elektromagnetische Feldtheorie*, 2. Aufl. Wiesbaden: Vieweg+Teubner Verlag, 2011.

[8] J. D. Jackson, *Classical Electrodynamics*, 3. Aufl. New York: Wiley and Sons, 1998.

[9] M. Clemens, M. Wilke, G. Benderskaya, H. DeGersem, W. Koch, und T. Weiland, „Transient Electro-Quasistatic Adaptive Simulation Schemes," *IEEE Transactions on Magnetics*, Vol. 40, Nr. 2, S. 1294–1297, 2004.

[10] T. Steinmetz, M. Helias, G. Wimmer, L. Fichte, und M. Clemens, „Electro-quasistatic field simulations based on a discrete electromagnetism formulation,“ *IEEE Transactions on Magnetics*, Vol. 42, Nr. 4, S. 755–758, 2006.

[11] U. van Rienen, *Numerical Methods in Computational Electrodynamics*, Serie Lecture Notes in Computational Science and Engineering. Berlin, Heidelberg: Springer, 2001, Vol. 12.

[12] H. Haus und J. R. Melcher, *Electromagnetic Fields and Energy.* Englewood Cliffs, NJ: Prentice Hall, 1989.

[13] E. Hairer, S. P. Nørsett, und G. Wanner, *Solving Ordinary Differential Equations I: Nonstiff Problems*, 2. Aufl., Serie Springer Series in Computational Mathematics. Berlin: Springer, 2000.

[14] K. Yee, „Numerical solution of initial boundary value problems involving maxwell's equations in isotropic media,“ *IEEE Transactions on Antennas and Propagation*, Vol. 14, Nr. 3, S. 302–307, 1966.

[15] T. Weiland, „Time Domain Electromagnectic Field Computation with Finite Difference Methods,“ *International Journal of Numerical Modelling: Electronic Networks, Devices and Fields*, Vol. 9, Nr. 4, S. 295–319, 1996.

[16] P. Silvester und R. L. Ferrari, *Finite Elements for Electrical Engineers*, 2. Aufl. Cambridge UP, 1996.

[17] R. Einzinger, „Metal Oxide Varistors,“ *Annual Review of Materials Science*, Vol. 17, Nr. 1, S. 299–321, 1987.

[18] K. Eda, „Zinc oxide varistors,“ *IEEE Electrical Insulation Magazine*, Vol. 5, Nr. 6, S. 28–30, 1989.

[19] F. Greuter, M. Siegrist, P. Kluge-Weiss, R. Kessler, L. Donzel, R. Loitzl, und H. J. Gramespacher, „Microvaristors: Functional Fillers for Novel Electroceramic Composites,“ *Journal of Electroceramics*, Vol. 13, Nr. 1-3, S. 739–744, 2004.

[20] F. Greuter, Y. Dirix, P. Kluge-Weiss, W. Schmidt, und R. Kessler, „Polymer compound with nonlinear current-voltage characteristic and process for producing a polymer compound,“ *US Patent 7,320,762*, 2008.

[21] T. Christen, L. Donzel, und F. Greuter, „Nonlinear resistive electric field grading part 1: Theory and simulation," *IEEE Electrical Insulation Magazine*, Vol. 26, Nr. 6, S. 47–59, 2010.

[22] L. Donzel, F. Greuter, und T. Christen, „Nonlinear resistive electric field grading part 2: Materials and applications," *IEEE Electrical Insulation Magazine*, Vol. 27, Nr. 2, S. 18–29, 2011.

[23] J. Stolz, „Untersuchung der Einsatzbereiche und Einsatzmöglichkeiten mikrovaristorgefüllter Silikonelastomere im Bereich des Überspannungsschutzes für Niederspannungsanwendungen," Dissertation, TU Kaiserslautern, 2009.

[24] J.-O. Debus, „Untersuchung der Anwendungsmöglichkeiten mikrovaristorgefüllter Feldsteuerelemente in der elektrischen Energietechnik," Dissertation, TU Darmstadt, 2014.

[25] S. Blatt, „Untersuchungen zu einem möglichen Einsatz von Mikrovaristoren in der Isolation umrichtergespeister Antriebe," Dissertation, TU Darmstadt, 2015.

[26] Wacker Chemie Deutschland, „Datenblatt SLM 79049 zum mikrovaristorgesteuerten Feldsteuermaterial."

[27] G. D. Knott, *Interpolating Cubic Splines.* Boston, MA: Birkhäuser Boston, 2000.

[28] W. Schiesser, *The Numerical Method of Lines: Integration of partial differential Equations.* San Diego: Elsevier Academic Press, 1991.

[29] D. Weida, „Optimierung von Applikationen aus der Hochspannungstechnik mit dünnen Schichten aus mikrovaristorgefüllten Polymeren mithilfe von nichtlinearen transienten 3D Simulationen," Dissertation, Bergische Universiät Wuppertal, 2011.

[30] K. Eriksson, C. Johnson, und D. Estep, „FEM für Randwertprobleme in R2 und R3," *Angewandte Mathematik: Body and Soul: Analysis in mehreren Dimensionen*, S. 1099–1127, 2005.

[31] D. Braess, *Finite Elemente: Theorie, schnelle Löser und Anwendungen in der Elastizitätstheorie.* Berlin, Heidelberg: Springer, 2007.

[32] S. C. Brenner und L. R. Scott, *The Mathematical Theory of Finite Element Methods*, 3. Aufl., Serie Texts in Applied Mathematics. New York, NY: Springer New York, 2008, Vol. 15.

[33] C.-D. Munz und T. Westermann, *Numerische Behandlung gewöhnlicher und partieller Differenzialgleichungen*, 3. Aufl. Berlin, Heidelberg: Springer, 2012.

[34] M. G. Larson und F. Bengzon, *The Finite Element Method: Theory, Implementation, and Applications*, Serie Texts in Computational Science and Engineering. Berlin, Heidelberg: Springer, 2013, Vol. 10.

[35] P. Steinke, *Finite-Elemente-Methode*, 2. Aufl. Berlin: Springer, 2007.

[36] R. Hiptmair, „Finite elements in computational electromagnetism," *Acta Numerica*, Vol. 11, S. 237–339, 2002.

[37] I. Babuška, „The finite element method with Lagrangian multipliers," *Numerische Mathematik*, Vol. 20, Nr. 3, S. 179–192, 1973.

[38] B. Klein, *FEM*, 7. Aufl. Wiesbaden: Vieweg, 2007.

[39] V. Thomée, *Galerkin Finite Element Methods for Parabolic Problems*, Serie Springer Series in Computational Mathematics. Berlin, Heidelberg: Springer, 2006, Vol. 25.

[40] O. C. Zienkiewicz und R. L. Taylor, *The Finite Element Method, The Basis*, 5. Aufl. Oxford: Butterworth Heinemann, 2001, Vol. 1.

[41] R. Courant, K. Friedrichs, und H. Lewy, „Über die partiellen Differenzengleichungen der mathematischen Physik," *Mathematische Annalen*, Vol. 100, Nr. 1, S. 32—-74, 1928.

[42] A. Abdulle, „Explicit Stabilized Runge-Kutta Methods," in *Encyclopedia of Applied and Computational Mathematics*, B. Engquist, Hrsg. Berlin, Heidelberg: Springer, 2015, S. 460 – 468.

[43] E. Hairer, S. P. Nørsett, und G. Wanner, *Solving Ordinary Differential Equations II: Stiff and Differential-Algebraic Problems*, 2. Aufl., Serie Springer Series in Computational Mathematics. Berlin: Springer, 2002.

[44] C. F. Curtiss und J. O. Hirschfelder, „Integration of Stiff Equations," *Proceedings of the National Academy of Sciences*, Vol. 38, Nr. 3, S. 235–243, mar 1952.

[45] C. W. Gear, *Numerical Initial Value Problems in Ordinary Differential Equations.* Upper Saddle River, NJ: Prentice Hall PTR, 1971.

[46] P. Deuflhard, *Newton methods for nonlinear problems: affine invariance and adaptive algorithms.* Berlin: Springer, 2004.

[47] G. G. Dahlquist, „A special stability problem for linear multistep methods," *BIT*, Vol. 3, Nr. 1, S. 27–43, 1963.

[48] F. Cameron, „A Class of low order DIRK methods for a class of DAEs," *Applied Numerical Mathematics*, Vol. 31, Nr. 1, S. 1–16, 1999.

[49] M. Bollhöfer und V. Mehrmann, *Numerische Mathematik.* Wiesbaden: Vieweg+Teubner, 2004.

[50] J. Verwer, „Explicit Runge-Kutta methods for parabolic partial differential equations," *Applied Numerical Mathematics*, Vol. 22, Nr. 1-3, S. 359–379, 1996.

[51] V. Saul'ev, „Integration of parabolic type equations with the method of nets," *Fizmatgiz*, 1960.

[52] A. Guillou und B. Lago, „Domaine de stabilité associé aux formules d'intégration numérique d'equations différentielles, à pas séparés et à pas liés." *Recherche de formules à grand rayon de stabilité*, S. 43–56, 1960.

[53] W. Gentzsch und A. Schlüter, „Über ein Einschrittverfahren mit zylischer Schrittweitenänderung zur Lösung parabolischer Differentialgleichungen," *Zeitschrift für Angewandte Mathematik und Mechanik*, Vol. 58, S. 415–416, 1978.

[54] P. J. van Der Houwen und B. P. Sommeijer, „On the Internal Stability of Explicit,m-Stage Runge-Kutta Methods for Largem-Values," *ZAMM - Zeitschrift für Angewandte Mathematik und Mechanik*, Vol. 60, Nr. 10, S. 479–485, 1980.

[55] B. P. Sommeijer und J. G. Verwer, „A performance evaluation of a class of Runge-Kutta-Chebyshev methods for solving semi-discrete parabolic differential equations," Mathematisch Centrum, Amsterdam, Tech. Rep., 1980.

[56] B. Sommeijer, L. Shampine, und J. Verwer, „RKC: An explicit solver for parabolic PDEs,” *Journal of Computational and Applied Mathematics*, Vol. 88, Nr. 2, S. 315–326, 1998.

[57] J. G. Verwer, W. H. Hundsdorfer, und B. P. Sommeijer, „Convergence properties of the Runge-Kutta-Chebyshev method,” *Numerische Mathematik*, Vol. 57, Nr. 1, S. 157–178, 1990.

[58] B. P. Sommeijer, „RKC, a nearly-stiff ODE solver,” 1991. [Online]. Available: https://netlib.sandia.gov/ode/rkc.f

[59] D. Fey, *Grid computing.* Heidelberg: Springer, 2010.

[60] B. Chapman, G. Jost, und R. van der Pas, *Using OpenMP: Portable Shared Memory Parallel Programming*, 1. Aufl. Cambridge: The MIT Press, 2008.

[61] J. Reinders, *Intel threading building blocks : outfitting C++ for multi-core processor parallelism.* O’Reilly, 2007.

[62] D. Blythe, „The Direct3D 10 system,” *ACM Transactions on Graphics*, Vol. 25, Nr. 3, S. 724, 2006.

[63] ——, „Rise of the Graphics Processor,” *Proceedings of the IEEE*, Vol. 96, Nr. 5, S. 761–778, 2008.

[64] J. Sanders und E. Kandrot, *CUDA by example: an introduction to general-purpose GPU programming*, 1. Aufl. Addison-Wesley Professional, 2010.

[65] J. D. Owens, M. Houston, D. Luebke, S. Green, J. E. Stone, und J. C. Phillips, „GPU Computing,” *Proceedings of the IEEE*, Vol. 96, S. 879–899, 2008.

[66] J. Bolz, I. Farmer, E. Grinspun, und P. Schröder, „Sparse matrix solvers on the GPU,” *ACM Transactions on Graphics*, Vol. 22, Nr. 3, S. 917–924, 2003.

[67] Nvidia, „CUDA Programming Guide, Version 7.5,” NVIDIA Corporation, Tech. Rep., 2015.

[68] J. E. Stone, D. Gohara, und G. Shi, „OpenCL: A Parallel Programming Standard for Heterogeneous Computing Systems,” *Computing in Science & Engineering*, Vol. 12, Nr. 3, S. 66–73, 2010.

[69] Nvidia, „Nvidia Tesla C1060 Fact Sheet," Nvidia Corporation, Tech. Rep., 2009. [Online]. Available: https://www.nvidia.co.uk/object/tesla_c1060_uk.html

[70] ——, „Nvidia Tesla V100 Fact Sheet," Nvidia Corporation, Tech. Rep., 2018. [Online]. Available: https://www.nvidia.com/content/PDF/Volta-Datasheet.pdf

[71] ——, „Nvidia Tesla Kepler Fact Sheet," Nvidia Corporation, Tech. Rep., 2013. [Online]. Available: https://www.nvidia.de/content/tesla/pdf/NVIDIA-Tesla-Kepler-Family-Datasheet.pdf

[72] ——, „Tesla K80 GPU Accelerator Board Specifications," Nvidia Corporation, Tech. Rep., 2015. [Online]. Available: http://images.nvidia.com/content/pdf/kepler/Tesla-K80-BoardSpec-07317-001-v05.pdf

[73] Y. Saad, *Iterative Methods for Sparse Linear Systems*, 2. Aufl. Boston: SIAM, 2003.

[74] N. Bell und M. Garland, „Efficient Sparse Matrix-Vector Multiplication on CUDA," NVIDIA Corporation, NVIDIA Technical Report NVR-2008-004, 2008.

[75] M. M. Dehnavi, D. M. Fernandez, und D. Giannacopoulos, „Finite-Element Sparse Matrix Vector Multiplication on Graphic Processing Units," *IEEE Transactions on Magnetics*, Vol. 46, Nr. 8, S. 2982–2985, 2010.

[76] ——, „Enhancing the Performance of Conjugate Gradient Solvers on Graphic Processing Units," *IEEE Transactions on Magnetics*, Vol. 47, Nr. 5, S. 1162–1165, 2011.

[77] R. Barrett, M. Berry, T. F. Chan, J. Demmel, J. Donato, J. Dongarra, V. Eijkhout, R. Pozo, C. Romine, und H. van der Vorst, *Templates for the Solution of Linear Systems: Building Blocks for Iterative Methods.* Society for Industrial and Applied Mathematics, 1994.

[78] W. Hackbusch, *Iterative Lösung großer schwachbesetzter Gleichungssysteme.* Wiesbaden: Vieweg+Teubner Verlag, 1993.

[79] A. Meister, *Numerik linearer Gleichungssysteme.* Wiesbaden: Vieweg+Teubner, 2011.

[80] S. Dalton, N. Bell, L. Olson, und M. Garland, „Cusp: Generic Parallel Algorithms for Sparse Matrix and Graph Computations," 2014. [Online]. Available: http://cusplibrary.github.io/

[81] A. F. P. de Camargos, V. C. Silva, J.-M. Guichon, und G. Munier, „Efficient Parallel Preconditioned Conjugate Gradient Solver on GPU for FE Modeling of Electromagnetic Fields in Highly Dissipative Media," *IEEE Transactions on Magnetics*, Vol. 50, Nr. 2, S. 569–572, 2014.

[82] M. Naumov, „Incomplete-LU and Cholesky preconditioned iterative methods using CUSPARSE and CUBLAS," *Nvidia white paper*, 2011.

[83] M. Lukash, K. Rupp, und S. Selberherr, „Sparse approximate inverse preconditioners for iterative solvers on GPUs," in *Proceedings of the 2012 Symposium on High Performance Computing.* Society for Computer Simulation International, 2012, S. 13.

[84] M. M. Dehnavi, D. M. Fernandez, J. L. Gaudiot, und D. D. Giannacopoulos, „Parallel sparse approximate inverse preconditioning on graphic processing units," *IEEE Transactions on Parallel and Distributed Systems*, Vol. 24, Nr. 9, S. 1852–1862, 2013.

[85] R. Li und Y. Saad, „GPU-accelerated preconditioned iterative linear solvers," *The Journal of Supercomputing*, Vol. 63, Nr. 2, S. 443–466, 2013.

[86] K. Rupp, J. Weinbub, A. Jüngel, und T. Grasser, „Pipelined iterative solvers with kernel fusion for graphics processing units," *ACM Transactions on Mathematical Software (TOMS)*, Vol. 43, Nr. 2, S. 11, 2016.

[87] U. Trottenberg, C. Oosterlee, und A. Schüuller, *Multigrid.* San Diego: Elsevier Academic Press, 2001.

[88] K. Stüben, „Algebraic multigrid (AMG): an introduction with applications," GMD, Report November, 1999.

[89] D. M. Young, *Iterative Solution of Large Linear Systems.* Orlando, FL, USA: Elsevier Academic Press, 1971.

[90] N. Bell, S. Dalton, und L. N. Olson, „Exposing Fine-Grained Parallelism in Algebraic Multigrid Methods," *SIAM Journal on Scientific Computing*, Vol. 34, Nr. 4, S. C123–C152, 2012.

[91] K. Stüben, „An Introduction to Algebraic Multigrid," in *Multigrid.* San Diego: Elsevier Academic Press, 2001, ch. Appendix A, S. 413 – 532.

[92] P. Wesseling, *An Introduction to Multigrid Methods.* New York: John Wiley & Sons, Ltd., 1991.

[93] D. Göddeke, „Fast and accurate finite-element multigrid solvers for PDE simulations on GPU clusters," Dissertation, Technische Universität Dortmund, 2010.

[94] K. Stüben, „A review of algebraic multigrid," *Journal of Computational and Applied Mathematics*, Vol. 128, Nr. 1-2, S. 281–309, 2001.

[95] P. Vaněk, J. Mandel, und M. Brezina, „Algebraic Multigrid On Unstructured Meshes," University of Colorado at Denver, Denver, Tech. Rep., 1994.

[96] ——, „Algebraic multigrid by smoothed aggregation for second and fourth order elliptic problems," *Computing*, Vol. 56, Nr. 3, S. 179–196, 1996.

[97] P. Hemker, „A note on defect correction processes with an approximate inverse of deficient rank," *Journal of Computational and Applied Mathematics*, Vol. 8, Nr. 2, S. 137–139, 1982.

[98] S. Balay, S. Abhyankar, M. Adams, J. Brown, P. Brune, K. Buschelman, L. Dalcin, V. Eijkhout, W. Gropp, D. Kaushik, M. Knepley, L. C. McInnes, K. Rupp, B. Smith, S. Zampini, H. Zhang, und H. Zhang, „PETSc Users Manual," Argonne National Laboratory, Tech. Rep. ANL-95/11 - Revision 3.7, 2016. [Online]. Available: http://www.mcs.anl.gov/petsc

[99] S. Balay, W. D. Gropp, L. C. McInnes, und B. F. Smith, „Efficient Management of Parallelism in Object Oriented Numerical Software Libraries," in *Modern Software Tools in Scientific Computing*, 1997, S. 163–202.

[100] S. Ashby und R. D. Falgout, „A Parallel Multigrid Preconditioned Conjugate Gradient Algorithm for Groundwater Flow Simulations,," *Nuclear Science and Engineering*, Vol. 124, S. 145 – 159, 1996.

[101] M. W. Gee, C. M. Siefert, J. J. Hu, R. S. Tuminaro, und M. G. Sala, „ML 5.0 Smoothed Aggregation User's Guide," Sandia National Laboratories, Tech. Rep. SAND2006-2649.

[102] L. Buatois, G. Caumon, und B. Lévy, „Concurrent Number Cruncher: An Efficient Sparse Linear Solver on the GPU," in *High Performance Computing and Communications*, 2007, S. 358–371.

[103] S. Sengupta, M. Harris, Y. Zhang, und J. D. Owens, „Scan primitives for GPU computing," *Graphics hardware*, S. 97–106, 2007.

[104] W. Wiggers, V. Bakker, A. Kokkeler, und G. Smit, „Implementing the conjugate gradient algorithm on multi-core systems," in *2007 International Symposium on System-on-Chip.* IEEE, 2007, S. 1–4.

[105] L. Buatois, G. Caumon, und B. Lévy, „Concurrent number cruncher: a GPU implementation of a general sparse linear solver," *International Journal of Parallel, Emergent and Distributed Systems*, Vol. 24, Nr. 3, S. 205–223, 2009.

[106] J. L. Greathouse und M. Daga, „Efficient Sparse Matrix-Vector Multiplication on GPUs Using the CSR Storage Format," in *SC14: International Conference for High Performance Computing, Networking, Storage and Analysis.* IEEE, 2015, S. 769–780.

[107] Lifeng Liu, Meilin Liu, Chongjun Wang, und Jun Wang, „LSRB-CSR: A Low Overhead Storage Format for SpMV on the GPU Systems," in *2015 IEEE 21st International Conference on Parallel and Distributed Systems (ICPADS).* IEEE, 2015, S. 733–741.

[108] J. Wong, E. Kuhl, und E. Darve, „A new sparse matrix vector multiplication graphics processing unit algorithm designed for finite element problems," *International Journal for Numerical Methods in Engineering*, Vol. 102, Nr. 12, S. 1784–1814, 2015.

[109] D. Göddeke, R. Strzodka, J. Mohd-Yusof, P. McCormick, H. Wobker, C. Becker, und S. Turek, „Using GPUs to improve multigrid solver performance on a cluster," *International Journal of Computational Science and Engineering*, Vol. 4, Nr. 1, S. 36–55, 2008.

[110] G. Haase, M. Liebmann, C. C. Douglas, und G. Plank, „A Parallel Algebraic Multigrid Solver on Graphics Processing Units," in *High Performance Computing and Applications: Second International Conference, HPCA 2009, Shanghai, China, August 10-12, 2009, Revised Selected Papers*, W. Zhang, Z. Chen, C. C. Douglas, und W. Tong, Hrsgg. Berlin, Heidelberg: Springer, 2010, S. 38–47.

[111] G. Haase, M. Liebmann, C. Douglas, und G. Plank, „Parallel Algebraic Multigrid on General Purpose GPUs," in *Proceeedings of the 3rd Austrian Grid Symposium, Linz, 2009*, 2010, S. 28–37.

[112] R. S. Tuminaro, „Parallel smoothed aggregation multigrid: Aggregation strategies on massively parallel machines," in *Proceedings of the 2000 ACM/IEEE conference on Supercomputing.* IEEE Computer Society, 2000, S. 5.

[113] M. Naumov, M. Arsaev, P. Castonguay, J. Cohen, J. Demouth, J. Eaton, S. Layton, N. Markovskiy, I. Reguly, N. Sakharnykh, V. Sellappan, und R. Strzodka, „AmgX: A Library for GPU Accelerated Algebraic Multigrid and Preconditioned Iterative Methods," *SIAM Journal on Scientific Computing*, Vol. 37, Nr. 5, S. S602–S626, 2015.

[114] D. Merrill und A. Grimshaw, „High performance and scalable radix sorting: A case study of implementing dynamic parallelism for GPU computing," *Parallel Processing Letters*, Vol. 21, Nr. 02, S. 245–272, 2011.

[115] G. R. Markall, A. Slemmer, D. A. Ham, P. H. J. Kelly, C. D. Cantwell, und S. J. Sherwin, „Finite element assembly strategies on multi-core and many-core architectures," *International Journal for Numerical Methods in Fluids*, Vol. 71, Nr. 1, S. 80–97, 2013.

[116] C. Cecka, A. J. Lew, und E. Darve, „Assembly of finite element methods on graphics processors," *International Journal for Numerical Methods in Engineering*, Vol. 85, Nr. 5, S. 640–669, 2011.

[117] A. Dziekonski, P. Sypek, A. Lamecki, und M. Mrozowski, „Finite element matrix generation on a GPU," *Progress In Electromagnetics Research*, Vol. 128, S. 249–265, 2012.

[118] J. Martínez-Frutos und D. Herrero-Pérez, „Efficient matrix-free GPU implementation of Fixed Grid Finite Element Analysis," *Finite Elements in Analysis and Design*, Vol. 104, S. 61–71, 2015.

[119] K. Banaś, F. Krużel, und J. Bielański, „Finite element numerical integration for first order approximations on multi- and many-core architectures," *Computer Methods in Applied Mechanics and Engineering*, Vol. 305, S. 827–848, 2016.

[120] J. P. A. Bastos und N. Sadowski, *Magnetic Materials and 3D Finite Element Modeling.* Boca Raton: CRC Press, 2013.

[121] D. Komatitsch, D. Michéa, und G. Erlebacher, „Porting a high-order finite-element earthquake modeling application to NVIDIA graphics cards using CUDA," *Journal of Parallel and Distributed Computing*, Vol. 69, Nr. 5, S. 451–460, 2009.

[122] D. Göddeke, R. Strzodka, J. Mohd-Yusof, P. McCormick, S. H. Buijssen, M. Grajewski, und S. Turek, „Exploring weak scalability for FEM calculations on a GPU-enhanced cluster," *Parallel Computing*, Vol. 33, Nr. 10-11, S. 685–699, 2007.

[123] J. Martínez-Frutos, P. J. Martínez-Castejón, und D. Herrero-Pérez, „Fine-grained GPU implementation of assembly-free iterative solver for finite element problems," *Computers & Structures*, Vol. 157, S. 9–18, 2015.

[124] S. Ikuno und T. Itoh, „GPU acceleration of variable preconditioned Krylov subspace method for linear system obtained by eXtended element-free Galerkin method," *2017 IEEE International Conference on Computational Electromagnetics, ICCEM 2017*, S. 344–345, 2017.

[125] A. Akbariyeh, B. Dennis, B. Lawrence, B. Wang, und K. Lawrence, „Comparison of GPU-Based Parallel Assembly and Assembly-Free Sparse Matrix Vector Multiplication for Finite Element Analysis of Three-Dimensional Structures," in *Proceedings of the Fifteenth International Conference on Civil, Structural and Environmental Engineering Computing*, 2015, S. 1–4.

[126] M. Clemens, M. Wilke, und T. Weiland, „Extrapolation strategies in numerical schemes for transient magnetic field simulations," *IEEE Transactions on Magnetics*, Vol. 39, Nr. 3, S. 1171–1174, 2003.

[127] M. Clemens, M. Wilke, R. Schuhmann, und T. Weiland, „Subspace Projection Extrapolation Scheme for Transient Field Simulations," *IEEE Transactions on Magnetics*, Vol. 40, Nr. 2, S. 934–937, 2004.

[128] P. Benner, S. Gugercin, und K. Willcox, „A Survey of Projection-Based Model Reduction Methods for Parametric Dynamical Systems," *SIAM Review*, Vol. 57, Nr. 4, S. 483–531, 2015.

[129] G. Strang, *Introduction to linear algebra*, 5. Aufl. Wellesley, MA: Wellesley - Cambridge Press, 2016.

[130] Y. Sato, F. Campelo, und H. Igarashi, „Fast Shape Optimization of Antennas Using Model Order Reduction," *IEEE Transactions on Magnetics*, Vol. 51, Nr. 3, S. 1–4, 2015.

[131] T. Henneron und S. Clenet, „Model Order Reduction of Non-Linear Magnetostatic Problems Based on POD and DEI Methods," *IEEE Transactions on Magnetics*, Vol. 50, Nr. 2, S. 33–36, 2014.

[132] D. Schmidthäusler und M. Clemens, „Low-Order Electroquasistatic Field Simulations Based on Proper Orthogonal Decomposition," *IEEE Transactions on Magnetics*, Vol. 48, Nr. 2, S. 567–570, 2012.

[133] D. Schmidthäusler, S. Schöps, und M. Clemens, „Reduction of Linear Subdomains for Non-Linear Electro-Quasistatic Field Simulations," *IEEE Transactions on Magnetics*, Vol. 49, Nr. 5, S. 1669–1672, 2013.

[134] ——, „Linear Subspace Reduction for Quasistatic Field Simulations to Accelerate Repeated Computations," *IEEE Transactions on Magnetics*, Vol. 50, Nr. 2, S. 421–424, 2014.

[135] L. Sirovich, „Turbulence and the dynamics of coherent structures. I. Coherent structures," *Quarterly of applied mathematics*, Vol. 45, Nr. 3, S. 561–571, 1987.

[136] L. N. Trefethen und D. Bau, *Numerical linear algebra.* Philadelphia: Society for Industrial and Applied Mathematics, 1997.

[137] A. C. Antoulas, *Approximation of Large-Scale Dynamical Systems*, 6. Aufl. Philadelphia: Society for Industrial and Applied Mathematics, 2005.

[138] G. Guennebaud, B. Jacob, und Others, „Eigen v3," 2010. [Online]. Available: http://eigen.tuxfamily.org

[139] F. Fritzen, B. Haasdonk, D. Ryckelynck, und S. Schöps, „An algorithmic comparison of the Hyper-Reduction and the Discrete Empirical Interpolation Method for a nonlinear thermal problem," *Mathematical and Computational Applications*, Vol. 23, Nr. 1, 2018.

[140] B. Smith, P. Bjorstad, und W. Gropp, *Domain Decomposition: Parallel Multilevel Methods for Elliptic Partial Differential Equations.* Cambridge University Press, 2004.

[141] M. J. Gander, „50 Years of Time Parallel Time Integration," in *Multiple Shooting and Time Domain Decomposition Methods*, T. Carraro, M. Geiger, S. Körkel, und R. Rannacher, Hrsgg. Heidelberg: Springer, 2015.

[142] J.-L. Lions, Y. Maday, und G. Turinici, „A p̈ararealïn time discretization of PDE's," *Comptes Rendus de l'Académie des Sciences - Series I - Mathematics*, Vol. 332, Nr. 2, S. 661–668, 2001.

[143] S. Schöps, I. Niyonzima, und M. Clemens, „Parallel-In-Time Simulation of Eddy Current Problems Using Parareal," *IEEE Transactions on Magnetics*, Vol. 54, Nr. 3, 2018.

[144] F. Kasolis und M. Clemens, „Entropy-Based Model Reduction for Strongly Nonlinear Electro-Quasistatic Field Problems," *11th International Symposium on Electric and Magnetic Fields (EMF 2018), 10.-12.04.2018, Darmstadt, Germany. Abstract accepted*, 2017.

[145] ——, „Snapshot Selection Criteria for Model Order Reduction," *Eighteenth Biennial IEEE Conference on Electromagnetic Field Computation (CEFC 2018), 28.-31.10.2018, Hangzhou, VRC. Abstract submitted*, 2018.

# Eigene Veröffentlichungen

[B1] C. Richter, S. Schöps, und M. Clemens, „GPU Accelerated Explicit Time Integration Methods for Electro-Quasistatic Fields," *IEEE Transactions on Magnetics*, Vol. 53, Nr. 6, S. 1–4, 2017.

[B2] D. Weida, C. Richter, und M. Clemens, „Design of cable accessories using ZnO microvaristor material based on FEM simulations," in *2010 IEEE International Power Modulator and High Voltage Conference*. IEEE, 2010, S. 94–97.

[B3] ——, „Design of ZnO microvaristor material stress-cone for cable accessories," *IEEE Transactions on Dielectrics and Electrical Insulation*, Vol. 18, Nr. 4, S. 1262–1267, 2011.

[B4] C. Richter, S. Schöps, und M. Clemens, „GPU Acceleration of Finite Difference Schemes Used in Coupled Electromagnetic/Thermal Field Simulations," *IEEE Transactions on Magnetics*, Vol. 49, Nr. 5, S. 1649–1652, 2013.

[B5] ——, „GPU Acceleration of Algebraic Multigrid Preconditioners for Discrete Elliptic Field Problems," *IEEE Transactions on Magnetics*, Vol. 50, Nr. 2, S. 461–464, 2014.

[B6] ——, „GPU-Accelerated Mixed Precision Algebraic Multigrid Preconditioners for Discrete Elliptic Field Problems," in *9th IET International Conference on Computation in Electromagnetics (CEM 2014)*, Vol. 1, 2014, S. 33 – 34.

[B7] ——, „Multi-GPU Acceleration of Algebraic Multi-Grid Preconditioners for Elliptic Field Problems," *IEEE Transactions on Magnetics*, Vol. 51, Nr. 3, S. 1–4, 2015.

[B8] ——, „Multi-GPU Acceleration of Algebraic Multigrid Preconditioners," *Mathematics in Industry*, Vol. 23, S. 83 – 90, 2016.

[B9] ——, „Adaptive Multi-GPU Acceleration of Algebraic Multigrid Preconditioners for Elliptic Field Problems," in *16th International IGTE Symposium (IGTE 2014),Graz, Austria, 14.-17.09.2014. Book of Abstracts*, 2014, S. 23.

[B10] C. Richter, S. Schöps, J. Dutine, R. Schreiber, und M. Clemens, „Transient Simulation of Nonlinear Electro-Quasistatic Field Problems Accelerated by Multiple GPUs," *IEEE Transactions on Magnetics*, Vol. 52, Nr. 3, S. 1–4, 2016.

[B11] M. Clemens, C. Richter, D. Schmidtäusler, S. Schöps, und H. Ye, „Aktuelle Entwicklungen zu numerischen Simulationsverfahren in der elektrischen Energieübertragungstechnik," *6. RCC Fachtagung Werkstoffe- Forschung und Entwicklung neuer Technologien zur Anwendung in der elektrischen Energietechnik, Berlin, Germany, 20.-21.05.2015*, S. 307–312, 2015.

[B12] J. Dutiné, M. Clemens, C. Richter, S. Schöps, und G. Wimmer, „Explicit Time Integration of Eddy Current Problems with a Weakly Gauged Schur-Complement Vector Potential Formulation," in *URSI e.V. Kleinheubacher Tagung, Miltenberg, 28.-30.09.2015. Book of Abstracts*, 2015.

[B13] J. Dutiné, C. Richter, S. Schöps, und M. Clemens, „Explicit Time Integration Techniques for Electro- and Magneto-Quasistatic Field Simulations," *International Conference on Electromagnetics in Advanced Applications & IEEE-APS Topical Conference on Antennas and Propagation in Wireless Communications (ICEAA IEEE-APS 2017), Verona, Italy, 11.-15.09.2017*, 2017.