Institut für Parallele und Verteilte Systeme

Universität Stuttgart
Universitätsstraße 38
D–70569 Stuttgart

Bachelorarbeit

Mehrgitterverfahren in gemischter Genauigkeit auf GPUs

Daniel Fink

Studiengang:	Simulation Technology
Prüfer/in:	Prof. Dr. rer. nat. habil. Miriam Mehl Prof. Dr. rer. nat. Dirk Pflüger
Betreuer/in:	Malte Brunn, M.Sc.
Beginn am:	9. April 2019
Beendet am:	9. Oktober 2019

Kurzfassung

Mehrgitterverfahren kommen heutzutage in weiten Teilen des wissenschaftlichen Rechnens zur Anwendung. Durch die Verwendung von gemischten Genauigkeiten kann die Ausführungszeit für solche Verfahren reduziert werden. Im Zuge dieser Arbeit werden vier gemischt genaue Mehrgitterverfahren hergeleitet und die Fehlerentwicklung sowie die Ausführungszeit analysiert. Hierbei werden die drei Datentypen Double (FP64), Single (FP32) und Half (FP16) verwendet. Es stellt sich heraus, dass die Verwendung von unterschiedlichen Datentypen keinen Einfluss auf die Fehlerentwicklung hat. Der größte gemessene Speed-Up beträgt dabei 25 Prozent gegenüber einem herkömmlichen Mehrgitterverfahren in doppelter Genauigkeit.

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

Algorithmenverzeichnis

Teil I
Propädeutikum

1 Einleitung und Übersicht

Durch die rasante Entwicklung im Bereich des Maschinellen Lernens werden immer leistungsfähigere Computer benötigt, um die riesigen Datenmengen in einer angemessenen Zeit verarbeiten zu können. Hierbei stellte sich heraus, dass für das Lernen von großen neuronalen Netzen die Genauigkeit, in der eine Zahl berechnet wird, deutlich geringer sein kann, als dies für andere wissenschaftliche Bereiche der Fall ist. Dadurch begannen Hardwarehersteller auch Recheneinheiten für Datentypen mit geringeren Genauigkeiten nativ auf den Chips zu implementieren. Im Zuge dieser Arbeit werden Möglichkeiten aufgeführt, so dass auch in den Fällen, in denen eine hohe Genauigkeit erforderlich ist, aus der Verwendung solcher Datentypen profitiert werden kann. Hierfür wird exemplarisch das Mehrgitterverfahren betrachtet, welches zum Lösen von partiellen Differentialgleichungen verwendet werden kann.

Die vorliegende Arbeit besteht aus zwei großen Teilen: dem Propädeutikum und der Bachelorthesis. Ersteres umfasst die Kapitel 1 und 2 und beschäftigt sich vorwiegend mit den theoretischen Grundlagen, die für die spätere Analyse von gemischt genauen Mehrgitterverfahren notwendig sind. Die Bachelorthesis streckt sich über die Kapitel 3 bis 6 und behandelt neben der Herleitung der gemischt genauen Verfahren auch die Analyse der Implementierung und die Diskussion der Resultate.

In Kapitel 1 wird zunächst das Ausgangsproblem und die verwendete Diskretisierung erläutert. Neben einer kleinen Einführung in die Gleitkommaarithmetik wird ebenfalls auf spezielle Aspekte der Grafikkartenprogrammierung eingegangen. Abschließend werden die ersten Erfolge von gemischt genaue Verfahren aufgezeigt. In Kapitel 2 wird das herkömmliche Mehrgitterverfahren motiviert und hergeleitet. Wie sich unterschiedliche Datentypen in solchen Verfahren verwenden lassen, ist Gegenstand des 3. Kapitels. Hierbei wird neben der Herleitung der gemischt genauen Verfahren auch auf den zu erwartenden Speed-Up eingegangen. Dieser wird neben den Fehlerentwicklungen in Kapitel 4 anhand der Implementierung diskutiert. In Kapitel 5 werden die Ergebnisse der Untersuchung in Form eines Fazits zusammengefasst ehe in Kapitel 6 ein Ausblick über mögliche Erweiterungen und Verbesserungen gegeben wird.

1.1 Poisson-Problem

Um die gemischt genauen Verfahren praktisch untersuchen zu können, werden im Zuge dieser Arbeit Mehrgitterverfahren implementiert, um das Poisson-Problem in zwei Dimensionen zu lösen. Hierbei handelt es sich um eine elliptische partielle Differentialgleichung (PDE) zweiter Ordnung, welche vorwiegend in Teilen der Physik zur Anwendung kommt. Unter Anderem dient sie im Teilbereich der Elektrostatik der Bestimmung des elektrischen Potentials [1]:

Aufgrund der ruhenden Ladungen erhält man in der Elektrostatik ein konservatives elektrisches Feld $\mathbf{E}$, welches über den Gradienten $\nabla\Phi$ des elektrischen Potentials Φ ausgedrückt werden kann:

$$\mathbf{E}(\mathbf{r}) = -\nabla\Phi(\mathbf{r}).$$

Wendet man den Divergenz-Operator ∇ auf die Gleichung an, so erhält man

$$\nabla \cdot \mathbf{E}(\mathbf{r}) = -\Delta\Phi(\mathbf{r}),$$

wobei $\Delta = \nabla \cdot \nabla = \partial_x^2 + \partial_y^2$ den Laplace-Operator in zwei Raumdimensionen darstellt. Gemäß der ersten Maxwell-Gleichung gilt zusätzlich

$$\nabla \cdot \mathbf{E}(\mathbf{r}) = \frac{\rho(\mathbf{r})}{\varepsilon}$$

mit der Ladungsdichte ρ und der Permittivität ε. Damit folgt die Poisson-Gleichung in der Elektrostatik zu

$$-\Delta\Phi(\mathbf{r}) = \frac{\rho(\mathbf{r})}{\varepsilon}.$$

Im Allgemeinen wird beim Poisson-Problem nach einer unbekannten, zweimal stetig differenzierbaren Funktion u bei gegebener rechter Seite f gesucht. Für die Eindeutigkeit der Lösung werden zusätzliche Randbedingungen gefordert. Mit Dirichlet-Randbedingungen lautet das 2D-Poisson-Problem auf dem Einheitsquadrat $\Omega = (0,1) \times (0,1)$ somit

$$\begin{aligned} -\Delta u &= f \quad \text{in } \Omega \\ u &= g \quad \text{auf } \partial\Omega, \end{aligned}$$

wobei $g : \partial\Omega \to \mathbb{R}$ die Randwerte darstellen. Im Fall von $g \equiv 0$ spricht man von *homogenen* Dirichlet-Randbedingungen.

Um PDEs wie das Poisson-Problem zu lösen, wird in der Praxis häufig die schnelle Fourier-Transformation verwendet. Hierbei wird das Problem in den Fourier-Raum überführt, wobei aus den zweiten partiellen Ableitungen eine Multiplikation mit einem Polynom zweiten Grades wird, was das Problem deutlich vereinfacht. Im Zuge dieser Arbeit steht das Poisson-Problem exemplarisch für eine Reihe an partiellen Differentialgleichungen, welche nicht, oder nicht so einfach, durch die schnelle Fourier-Transformation gelöst werden können.

1.2 Diskretisierung

Für eine computergestützte Berechnung werden die kontinuierliche Gleichung und insbesondere der Differentialoperator Δ diskretisiert. Hierbei gibt es unterschiedliche Methoden und Verfahren wie z.B. die Finite-Elemente-Methode. Da der Fokus dieser Arbeit auf der Verwendung gemischter Genauigkeiten liegt, wird im Folgenden die einfachste Möglichkeit der Diskretisierung verwendet, die *Finite-Differenzen-Methode*. Hierbei wird das Einheitsquadrat durch ein reguläres Gitter angenähert. Vereinfachend kann die Kantenlänge in alle Dimensionsrichtungen äquidistant gewählt werden. Für n Punkte pro Raumdimension ergibt sich damit das Gitter

$$\Omega_h = \left\{(ih, jh) : i, j = 0, 1, 2, \ldots, n-1\right\},$$

wobei $h = 1/(n-1)$ die *Gitterweite* bzw. *Kantenlänge* beschreibt. Insgesamt besteht das Gitter aus n^2 Gitterpunkten. Bei Verwendung von Dirichlet-Randbedingungen sind die Werte an den Randpunkten keine Unbekannten. Daher reduziert sich das Problem in diesem Fall auf $(n-2)^2$ innere Gitterpunkte. Im Folgenden wird deshalb auch nur die Anzahl der inneren Gitterpunkte pro Raumdimension mit n bezeichnet.

Die im Poisson-Problem auftretenden Ableitungen werden mit zentralen Differenzenquotienten zweiter Ordnung approximiert. Allgemein gilt

$$\partial_x^2 u(x_i, y_j) = \frac{u_{i+1,j} - 2u_{i,j} + u_{i-1,j}}{h^2} + \mathcal{O}(h^2) \qquad \text{mit} \qquad i, j = 1, \ldots, n$$

für die Ableitung in x-Richtung und

$$\partial_y^2 u(x_i, y_j) = \frac{u_{i,j+1} - 2u_{i,j} + u_{i,j-1}}{h^2} + \mathcal{O}(h^2) \qquad \text{mit} \qquad i,j = 1,\ldots,n$$

für die Ableitung in y-Richtung. Die Variablen $u_{i,j} := u(x_i, y_j)$ beschreiben die Werte der Lösung an den Gitterpunkten (x_i, y_j). Durch Addition der beiden Gleichungen folgt das diskretisierte Poisson-Problem

$$-\Delta u(x_i, y_j) = \frac{-u_{i+1,j} - u_{i,j+1} + 4u_{i,j} - u_{i-1,j} - u_{i,j-1}}{h^2} + \mathcal{O}(h^2) = f(x_i, y_j) \qquad \text{mit} \quad i,j = 1,\ldots,n. \tag{1}$$

Wird der Term $\mathcal{O}(h^2)$ vernachlässigt, erhält man in Gleichung (1) punktweise Approximationen $v_{i,j}$ der Lösungen $u_{i,j}$.

Mit der Definition

$$\mathbf{v} := \left(v_{1,1},\ldots,v_{n,1},v_{1,2},\ldots,v_{n,2},\ldots,v_{n,n}\right)^T \in \mathbb{R}^{n^2} \quad \text{und}$$
$$\mathbf{f} := \left(f_{1,1},\ldots,f_{n,1},f_{1,2},\ldots,f_{n,2},\ldots,f_{n,n}\right)^T \in \mathbb{R}^{n^2},$$

d.h. einer *zeilenweisen* Nummerierung der Gitterpunkte, lässt sich das diskretisierte Poisson-Problem mit homogenen Dirichlet-Randbedingungen kompakt formulieren zu

$$\mathbf{A}\mathbf{v} = \mathbf{f}.$$

Die auftretende Matrix $\mathbf{A} \in \mathbb{R}^{n^2 \times n^2}$ ist eine Blockmatrix und setzt sich aus den Matrizen $\mathbf{I} \in \mathbb{R}^{n \times n}$ und $\mathbf{D} \in \mathbb{R}^{n \times n}$ zusammen über

$$\mathbf{A} = \frac{1}{h^2}\begin{bmatrix} \mathbf{D} & -\mathbf{I} & & \\ -\mathbf{I} & \ddots & \ddots & \\ & \ddots & \ddots & -\mathbf{I} \\ & & -\mathbf{I} & \mathbf{D} \end{bmatrix},$$

$$\text{wobei } \mathbf{I} = \begin{bmatrix} 1 & & \\ & \ddots & \\ & & 1 \end{bmatrix} \text{ und } \mathbf{D} = \begin{bmatrix} 4 & -1 & & \\ -1 & 4 & \ddots & \\ & \ddots & \ddots & -1 \\ & & -1 & 4 \end{bmatrix}.$$

Der Vektor $\mathbf{v}$ beinhaltet lediglich die unbekannten inneren Gitterpunkte. Für inhomogene Randbedingungen können die bekannten Werte an den Rändern in die rechte Seite $\mathbf{f}$ integriert werden.

Stencil-Notation

Diskrete Operatoren lassen sich häufig kompakt in Form eines „*Stencils*“ beschreiben. Bei dieser Notation wird nur der Anteil des Operators notiert, der für die Berechnung eines einzelnen Gitterpunkts erforderlich ist. Formal hat ein Stencil in zwei Dimensionen die Form

$$\begin{bmatrix} & \vdots & \vdots & \vdots & \\ \cdots & s_{-1,-1} & s_{0,-1} & s_{1,-1} & \cdots \\ \cdots & s_{-1,0} & s_{0,0} & s_{1,0} & \cdots \\ \cdots & s_{-1,1} & s_{0,1} & s_{1,1} & \cdots \\ & \vdots & \vdots & \vdots & \end{bmatrix}.$$

Mit der Anwendungsvorschrift

$$v_{i,j} \mapsto \sum_{k,l} s_{k,l} v_{i+k,j+l}$$

wird somit ein Operator auf dem Raum der Gitterfunktionen definiert. Für den diskretisierten Laplace-Operator Δ_h in zwei Dimensionen erhält man

$$\Delta_h = \frac{1}{h^2} \begin{bmatrix} & -1 & \\ -1 & 4 & -1 \\ & -1 & \end{bmatrix},$$

was auch als 5-Punkte-Stern bezeichnet wird. Abbildung 1 veranschaulicht den Nutzen der Stencil-Notation im Hinblick auf die Gewichtung der Knoten im Gitter.

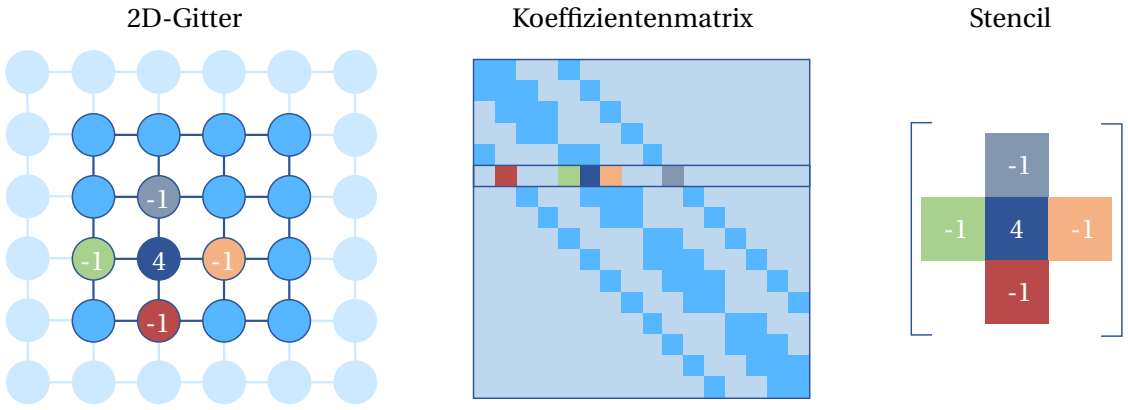


Abbildung 1: Links: Zweidimensionales Gitter mit n = 4. Der 5-Punkte-Stern mit zugehöriger Gewichtung ist farbig markiert. Mitte: Koeffizientenmatrix des Poisson-Problems. Eine Zeile ist exemplarisch für die mit Δ_h assoziierten Gitterpunkte markiert. Rechts: Stencil-Notation des diskretisierten Operators. Die Einträge des Stencils entsprechen einer Zeile der Koeffizientenmatrix.

1D-Poisson-Problem

Um Fehlerentwicklungen theoretisch zu untersuchen, bietet sich häufig das eindimensionale Poisson-Problem an, da die resultierenden Gleichungssysteme eine einfachere Form besitzen. Im Konkreten betrachtet man also

$$-\frac{d^2}{dx^2} u = f.$$

Wird der Differentialoperator wie im zweidimensionalen Fall mit zentralen Differenzenquotienten zweiter Ordnung approximiert und eine zeilenweise Nummerierung der Gitterpunkte verwendet, so erhält man als Gleichungssystem

$$\frac{1}{h^2} \begin{bmatrix} 2 & -1 & & & \\ -1 & 2 & -1 & & \\ & \ddots & \ddots & \ddots & \\ & & -1 & 2 & -1 \\ & & & -1 & 2 \end{bmatrix} \begin{bmatrix} v_1 \\ v_2 \\ \vdots \\ v_{n-1} \\ v_n \end{bmatrix} = \begin{bmatrix} f_1 \\ f_2 \\ \vdots \\ f_{n-1} \\ f_n \end{bmatrix}. \tag{2}$$

Der Differentialoperator im eindimensionalen Fall lässt sich analog zum zweidimensionalen Fall in Stencil-Notation darstellen und lautet

$$\Delta_h = \frac{1}{h^2} \begin{bmatrix} -1 & 2 & -1 \end{bmatrix}.$$

Red-Black-Nummerierung

Neben der zeilenweisen Nummerierung wird in der Anwendung auch häufig die sogenannte *„red-black"-Nummerierung* verwendet. Hierbei werden die Knoten nacheinander zuerst mit einer Farbe markiert, sodass zwei benachbarte Knoten immer eine unterschiedliche Farbe erhalten. Anschließend erfolgt die Nummerierung zeilenweise, wobei zuerst nur die eingefärbten und später dann die übrigen Knoten durchnummeriert werden. Der Vorteil dieser Nummerierung liegt in der höheren Datenunabhängigkeit. Dies gilt zumindest für Diskretisierungen mit dem 5-Punkte-Stern, da die Gleichungen der Unbekannten mit gleicher Farbe somit unabhängig voneinander sind und getrennt berechnet werden können. Eine solche Nummerierung kann je nach Verfahren zu einem höheren Parallelisierungsgrad führen. In Abbildung 2 ist die red-black-Nummerierung anhand eines Gitters mit n = 4 exemplarisch dargestellt.

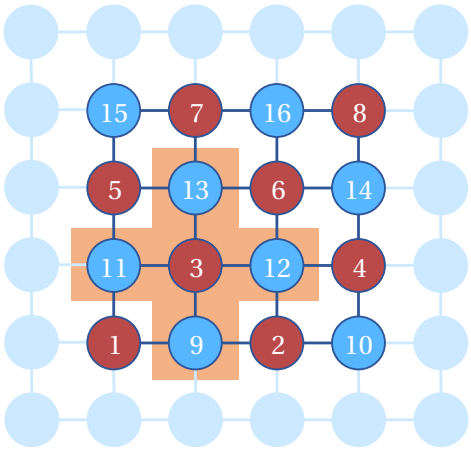


Abbildung 2: Zweidimensionales Gitter mit n = 4 und red-black-Nummerierung. Der 5-Punkte-Stern ist in Orange hervorgehoben. Für die Berechnung der Lösung an Gitterpunkt 3 werden nur die Gitterpunkte 9, 11, 12 und 13 benötigt.

1.3 Gleitkommaarithmetik

Nachdem das kontinuierliche Problem diskretisiert wurde, lässt sich damit auf einem Rechner arbeiten. Auf Grund des endlichen Speichers von Computern können jedoch nicht alle reellen Zahlen exakt dargestellt werden. Gleitkommazahlen stellen eine Approximation der reellen Zahlen dar und bilden mit den auf ihnen definierten Operationen eine endliche Arithmetik. Eine Gleitkommazahl besitzt die Form $M \cdot B^E$, wobei M als Mantisse, B als Basis und E als Exponent bezeichnet wird. In den meisten Rechnern wird als Basis B = 2 verwendet. Deshalb genügt es, wenn die Werte M und E im Speicher abgelegt werden, um eine Gleitkommazahl darzustellen. Je mehr Werte für M und E zulässig sind, desto genauer können die reellen Zahlen approximiert werden. Im IEEE-Standard 754-2008 [2] sind vier binäre Datentypen definiert: Half, Single, Double und Quadruple. Diese unterscheiden sich in der Anzahl an Bits, die für das Speichern der Mantisse und des Exponenten verwendet werden. Im Deutschen spricht man beim Datentyp Half auch von halber, bei Single von einfacher, bei Double von doppelter und bei Quadruple von vierfacher Genauigkeit. Um eine eindeutige Darstellung einer Gleitkommazahl zu erreichen, wird die Mantisse normiert, was bedeutet, dass als Vorkommazahl stets die Null gewählt wird. In Tabelle 1 sind die vier binären Datentypen mit den zugehörigen Speichergrößen für Mantisse und Exponent aufgelistet. Da es sich bei allen Gleitkommazahlen um Approximationen handelt, müssen die Ergebnisse von Rechnungen *gerundet* werden. Die gängigsten Rundungsverfahren sind

- Runden zum Nächsten,

Datentyp T	Gesamtlänge	Länge Mantisse M	Länge Exponent E	ε_T
Half	16 Bit	10 Bit	5 Bit	$2^{-11} \approx 4,8 \cdot 10^{-4}$
Single	32 Bit	23 Bit	8 Bit	$2^{-24} \approx 5,9 \cdot 10^{-8}$
Double	64 Bit	52 Bit	11 Bit	$2^{-53} \approx 1,1 \cdot 10^{-16}$
Quadruple	128 Bit	112 Bit	15 Bit	$2^{-113} \approx 9,6 \cdot 10^{-35}$

Tabelle 1: Speichergrößen und Maschinengenauigkeiten ε der binären Datentypen nach IEEE 754-2008. Neben der Mantisse und dem Exponenten ist in der Gesamtlänge noch ein Bit für das Vorzeichen enthalten.

- Runden in Richtung Null,
- Runden in Richtung $+\infty$ und
- Runden in Richtung $-\infty$.

Für viele Operationen wird das Runden zur nächsten Gleitkommazahl, häufig auch als korrektes Runden bezeichnet, standardmäßig verwendet. Aus Sicht des Datentyps Single stellt Half ebenfalls eine Approximation dar. Liegt eine Zahl im Datentyp Single vor, so kann diese mit den obigen Rundungsverfahren in das Half-Format gerundet werden. In diesem Fall spricht man von *Konvertierung*.

Der relative Fehler ε_{rd}, welcher beim Runden auftreten kann, entspricht dem relativen Abstand zweier aufeinanderfolgender Gleitkommazahlen und berechnet sich zu

$$\varepsilon_{rd} = \frac{(|M+1| \cdot B^E) - |M| \cdot B^E}{|M| \cdot B^E} = \frac{B^E}{|M| \cdot B^E} = \frac{1}{|M|}.$$

Da die Mantisse nach unten beschränkt ist ($M = 0$ wird als Spezialfall betrachtet), lässt sich der Rundungsfehler nach oben abschätzen:

$$\varepsilon_{rd} \leq \frac{1}{|M_{min}|} =: \rho.$$

ρ wird als *Auflösung* bezeichnet und hängt vom gewählten Datentyp ab. Da beim korrekten Runden immer zur nächstgelegenen Gleitkommazahl gerundet wird, beträgt hier der maximale Fehler nur die Hälfte. Dies führt zur Definition der *Maschinengenauigkeit* ε:

$$\varepsilon := \frac{\rho}{2}.$$

Da bereits die Auflösung vom verwendeten Datentyp abhängt, gilt dies auch für die Maschinengenauigkeit. In Tabelle 1 sind die Maschinengenauigkeiten der einzelnen Datentypen aufgelistet. In der Anwendung beschreibt ε unter Anderem die kleinstmögliche Abbruchschranke für iterative Lösungsverfahren, die noch sinnvoll eingesetzt werden kann. Wählt man kleinere Schranken, so könnten diese möglicherweise alleine aufgrund der Rundungen in den Rechnungen nicht erreicht werden.

1.4 Rechnen auf Grafikkarten

Das diskretisierte Problem kann nun mithilfe von Gleitkommazahlen auf einem Rechner gelöst werden. Klassischerweise wird hierfür die CPU eines Rechners verwendet. Anfang des 21. Jahrhunderts fing man jedoch an, das Potential der massiven Parallelität von Grafikkarten auch für andere Zwecke als Grafikanwendungen zu entdecken. Gerade im Bereich der numerischen Simulation zeigte sich, dass hochdimensionale Probleme auf Grafikkarten sehr effizient gelöst

werden können [3]. 2007 startete NVIDIA deshalb mit der Einführung von *CUDA*, einem Framework für Streamprozessoren auf NVIDIA-Grafikkarten [4]. 2008 wurde mit OpenCL schließlich der erste Standard einer Programmierschnittstelle für allgemeine Multi- und Manycore-Architekturen veröffentlicht [5]. Da für die Umsetzung der in dieser Arbeit behandelten Verfahren Grafikkarten des Herstellers NVIDIA zur Verfügung standen, fiel die Wahl auf CUDA.

CUDA

Die von NVIDIA entwickelte Programmier-Technik für paralleles Rechnen auf NVIDIA-Grafikkarten ist eine Erweiterung der Programmiersprache C++. Beim Programmieren unterscheidet man dabei zwischen Host- und Device-Code. Letzterer wird auf der Grafikkarte (Device) ausgeführt. Eine Funktion, die auf der Grafikkarte ausgeführt werden kann, wird Kernel genannt. Um Berechnungen durchzuführen, wird dabei ein Kernel mit mehreren Threads gestartet. Diese werden dann in einzelnen Blöcken von den jeweiligen Recheneinheiten bearbeitet. Dies erfolgt nach dem Prinzip von „*Single Instruction, Multiple Data (SIMD)*“, was bedeutet, dass pro Operation gleichzeitig mehrere Daten verarbeitet werden. Der Kernel beschreibt dabei den Code, der pro Thread ausgeführt wird. Um zu spezifizieren auf welchem Datensatz gearbeitet wird, besitzt jeder Thread einen eindeutigen Index.

Dass die SIMD-Eigenschaft der Grafikkarte ausgenutzt werden, sollte bei der Implementierung der Kernel darauf geachtet werden, dass möglichst alle Threads dieselben Operationen durchführen. Da der Zugriff auf den Grafikspeicher verhältnismäßig langsam ist, sollten dabei möglichst viele Threads ausgelastet werden, um Latenzzeiten, wie das Warten bei Speicheranforderungen, zu kaschieren. Aufgrund der parallelen Ausführung sollten auch Abhängigkeiten zwischen den Threads vermieden werden.

Neben den eben genannten weichen Kriterien für die Implementierung eines CUDA-Kernels gibt es eine ganze Reihe an Techniken und weiteren Hardwareaspekten. Auf einige davon wird im Folgenden näher eingegangen.

Vektorisierte Speicherzugriffe

Viele Anwendungen der numerischen Simulation sind speichergebunden, was bedeutet, dass der limitierende Faktor das Lesen und Schreiben von Daten ist. Bei einer Vektor-Vektor-Addition müssen pro Element lediglich eine Berechnung, aber mehrere Speicheranforderungen durchgeführt werden. Dies führt dazu, dass der Speichercontroller den größten Teil der Arbeit verrichtet und die Recheneinheiten nur einen Bruchteil ihres Potentials ausschöpfen. Eine Möglichkeit, wie man den Datendurchsatz erhöhen kann, liegt in der Verwendung von *vektorisierten Datentypen*. Hierbei werden zwei oder mehr skalare Daten zu einem Datenbündel zusammengefasst, auf denen dann die arithmetischen Operationen durchgeführt werden. Diese Art der Datenverarbeitung führt dazu, dass weniger Hardwareinstruktionen benötigt werden und der Speichercontroller damit auch weniger ausgelastet wird. Insgesamt kann somit ein höherer Datendurchsatz erzeugt werden.

CUDA-Cores und halbe Genauigkeit

Um Berechnungen durchzuführen, werden von den Prozessorherstellern einzelne Recheneinheiten in den Prozessoren verbaut. Diese sog. „Arithmetic Logic Units“, kurz ALU, sind abhängig von der Größe der Datentypen, mit denen die Berechnung durchgeführt wird. Da der Datentyp Single nur die Hälfte des Speicherplatzes von Double benötigt, sind die entsprechenden Single-ALU kleiner als die von Double. In vielen Grafikkarten sind deshalb mehr Single-ALU als Double-ALU verbaut, was zu einem höheren Rechendurchsatz bei Verwendung von Single führt [6]. Seit der Volta™-Architektur unterstützen NVIDIA-Grafikkarten den Half-Datentyp

nativ. Hierbei bearbeitet eine Single-ALU simultan zwei Half-Operationen, womit sich der Rechendurchsatz für Half verdoppelt, sofern der vektorisierte Datentyp *half2* verwendet wird. Bei Spezifikationen von NVIDIA-Grafikkarten wird meist nicht die Anzahl an ALU, sondern die der sog. „*CUDA Cores*“ angegeben. NVIDIA bezeichnet damit eine programmierbare Recheneinheit, welche die ALU der Gleit- und Festkommazahlen beinhaltet.

Geteilter Speicher

Auf NVIDIA-Grafikkarten stehen neben dem globalem Speicher noch weitere Speicherbereiche zur Verfügung. Ein Beispiel hierfür ist der *geteilte Speicher*. Dieser ist wie ein Zwischenspeicher aus einzelnen Speicherbänken aufgebaut und wird blockweise bereitgestellt, was bedeutet, dass alle Threads innerhalb eines Blocks auf denselben geteilten Speicher Zugriff haben. Hierzu können auch alle Threads eines Blocks synchronisiert werden. Im Zuge dieser Arbeit dient der geteilte Speicher vorwiegend als software-gesteuerter Zwischenspeicher, um bereits geladene Daten effizient wiederzuverwenden.

Textur-Zwischenspeicher

Der geteilte Speicher wird häufig in Kombination mit dem sog. *Textur-Zwischenspeicher* verwendet. Während aus ersterem sowohl gelesen als auch geschrieben werden kann, handelt es sich beim Textur-Zwischenspeicher um einen auf das Lesen beschränkten Zwischenspeicher. Im Gegensatz zum L1- und L2-Zwischenspeicher, welche softwareseitig nicht gesteuert werden können, kann beim Textur-Speicher spezifiziert werden, welche Zugriffe aus dem globalen Speicher darüber geroutet werden sollen. Der Textur-Speicher ist auf eine zweidimensionale Datenlokalität optimiert, was bedeutet, dass Daten von räumlich benachbarten Threads zwischengespeichert werden.

Fused Multiply-Add (FMA)

Neben den Grundrechenarten (Addition, Subtraktion, Multiplikation und Division) wird häufig auch eine Kombination aus Multiplikation und Addition direkt durchgeführt. Die sog. „*Fused Multiply-Add (FMA)*“-Operation berechnet das Produkt aus zwei Zahlen a und b und addiert das Ergebnis auf eine weitere Zahl c, d.h. $\mathrm{FMA}(a,b,c) = a \times b + c$. Die FMA-Operation ist in Abbildung 3 schematisch dargestellt. Der Vorteil dieser Operation ist dass lediglich das Endergebnis gerundet wird. Die Multiplikation selbst wird nicht gerundet und in diesem Sinne exakt durchgeführt, was zu weniger arithmetischen Fehlern führt. NVIDIA verwendet diese Operation seit der Fermi™-Architektur, welche 2010 veröffentlicht wurde [7]. Die FMA-Operation ist ebenfalls im IEEE-Standard 754-2008 [2] definiert.

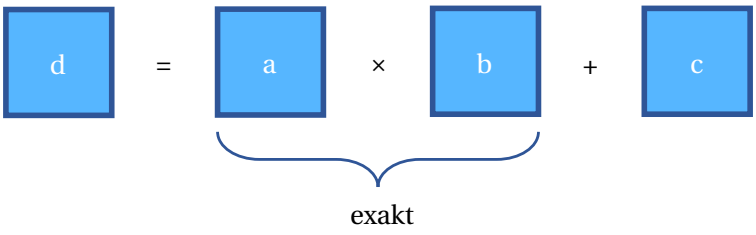


Abbildung 3: Schematische Darstellung einer Fused Multiply-Add-Operation. Das Produkt $a \times b$ wird exakt berechnet. Alle Werte liegen im selben Datentyp vor.

Tensor Core

Da in den letzten Jahren Grafikkarten vermehrt in Verbindung mit neuronalen Netzen verwendet werden, führte NVIDIA mit der Volta™-Architektur spezielle Recheneinheiten ein, um das Lernen solcher Netze zu beschleunigen [8]. Diese sog. „*Tensor Cores*“ erweitern die Fused Multiply-Add-Operationen auf Matrizen der Dimension 4×4. Ein Tensor Core kann damit eine Matrix-Matrix-Multiplikation mit anschließender Matrix-Addition durchführen, d.h. $TC(\mathbf{A},\mathbf{B},\mathbf{C}) = \mathbf{A} \times \mathbf{B} + \mathbf{C}$. In Abbildung 4 ist die Berechnung eines Tensor Cores schematisch dargestellt. Im Gegensatz zur FMA-Operation, bei welcher alle beteiligten Zahlen im selben Datentyp vorliegen, verwendet ein Tensor Core unterschiedliche Datentypen: Für die beiden Matrizen **A** und **B** wird das Half-Format verwendet, währenddessen für die Matrizen **C** und **D** Single oder Half verwendet werden kann. Damit führt ein Tensor Core insgesamt 64 gemischt genaue FMA-Operationen durch.

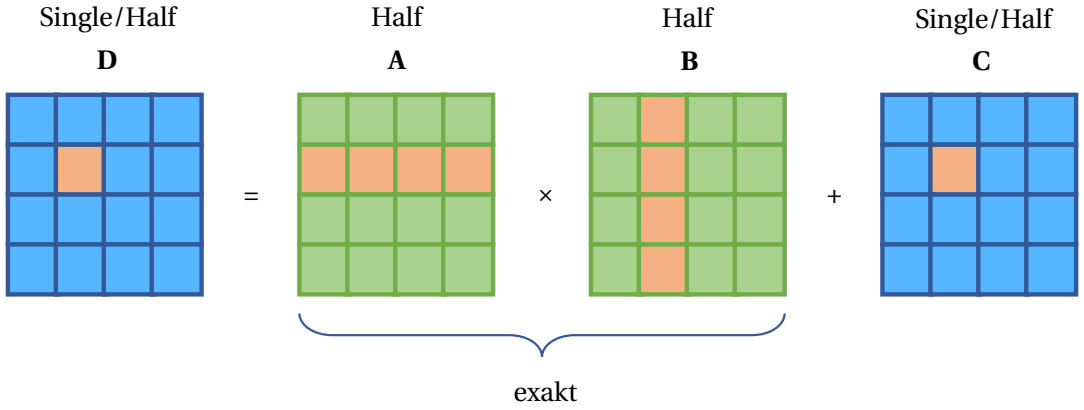


Abbildung 4: Schematische Darstellung der Berechnung eines Tensor Cores. Das Produkt $\mathbf{A} \times \mathbf{B}$ wird exakt berechnet. Die Matrizen **A** und **B** liegen im Half-Format vor, **C** und **D** in Single oder Half. Die Berechnung eines Eintrags der Matrix **D** ist exemplarisch eingefärbt.

1.5 Gemischte Genauigkeit

Hinter dem Begriff *gemischte Genauigkeit* steht die Idee, dass unterschiedliche Datentypen verwendet werden, um die Ausführungszeit, den Speicherbedarf oder den Kommunikationsaufwand von numerischen Verfahren zu verringern. Möchte man das Endresultat in doppelter Genauigkeit berechnen, so können bestimmte Teile des Verfahrens in einfacher oder gar in halber Genauigkeit gerechnet werden, ohne dass das Endresultat davon stark beeinflusst wird. Die entscheidende Frage ist jedoch, was „bestimmte Teile“ eines Verfahrens sind und ob ein solches Lösungsverfahren letztendlich auch gegen die exakte Lösung konvergiert. Ein mögliches Verfahren, anhand dessen diese Fragen geklärt werden können, ist die *iterative Verfeinerung*.

Iterative Verfeinerung

Ausgehend von einer Approximation $\mathbf{v}^{(k)}$ der Lösung $\mathbf{v}$ lässt sich der *Fehler* definieren als

$$\mathbf{e}^{(k)} := \mathbf{v} - \mathbf{v}^{(k)}.$$

Stellt man diese Gleichung nach $\mathbf{v}$ um, so folgt

$$\mathbf{v} = \mathbf{v}^{(k)} + \mathbf{e}^{(k)}.$$

Die Lösung $\mathbf{v}$ erhält man also, ausgehend von einer Approximation $\mathbf{v}^{(k)}$, durch Addition eines Korrekturterms, welcher dem Fehler $\mathbf{e}^{(k)}$ entspricht. Mit der Definition des *Residuums*

$$\mathbf{r}^{(k)} = \mathbf{b} - \mathbf{A}\mathbf{v}^{(k)}$$

erhält man somit

$$\mathbf{A}\mathbf{e}^{(k)} = \mathbf{A}(\mathbf{v} - \mathbf{v}^{(k)}) = \mathbf{A}\mathbf{v} - \mathbf{A}\mathbf{v}^{(k)} = \mathbf{b} - \mathbf{A}\mathbf{v}^{(k)} = \mathbf{r}^{(k)},$$

also das Gleichungssystem

$$\mathbf{A}\mathbf{e}^{(k)} = \mathbf{r}^{(k)}, \tag{3}$$

welches auch *Residuumsgleichung* genannt wird. Diese Erkenntnis macht sich die iterative Verfeinerung zu nutze. In Algorithmus 1 sind die drei Schritte des Verfahrens aufgeführt. Für den

Algorithmus 1 Iterative Verfeinerung

Für $k = 1, 2, \ldots$

1. Berechnung des Residuums und Konvergenzkontrolle
   $\mathbf{r}^{(k)} = \mathbf{b} - \mathbf{A}\mathbf{v}^{(k)}$
2. Lösen des Gleichungssystems
   $\mathbf{A}\mathbf{e}^{(k)} = \mathbf{r}^{(k)}$
3. Addition des Korrekturterms
   $\mathbf{v}^{(k+1)} = \mathbf{v}^{(k)} + \mathbf{e}^{(k)}$

zweiten Schritt kann prinzipiell jedes direkte Lösungsverfahren verwendet werden. Wilkinson [9] war der erste, der die iterative Verfeinerung im Hinblick auf gemischte Genauigkeiten untersuchte. Hierbei nahm er an, dass der erste und dritte Schritt exakt gerechnet werden und nur im zweiten Schritt arithmetische Fehler entstehen. Für das Lösen des Gleichungssystems verwendete er eine LR-Zerlegung. Mit diesen Annahmen konnte er die Konvergenz gegen die exakte Lösung beweisen.

Während Wilkinson für seinen Beweis im Jahre 1963 eine sog. „blockskalierende Arithmetik“ verwendete, erweiterte Moler [10] 1967 die Theorie auf Gleitkommaarithmetik, wodurch das Verfahren praxistauglich wurde. Erst in den letzten Jahren drang die Idee der gemischt genauen iterativen Verfeinerung wieder vermehrt in den Fokus. Ein paar der Ideen sollen im Folgenden kurz aufgeführt werden.

Iterative Verfeinerung mit direkten Lösern

Buttari et al. [11] führten die Punkte eins und drei der iterativen Verfeinerung in doppelter Genauigkeit und Punkt zwei in einfacher Genauigkeit durch. Hierbei verwendeten sie direkte Löser wie die LR- und Cholesky-Zerlegung. Die gemischt genaue iterative Verfeinerung erreichte in den Tests einen Speedup von $1,8$ für unsymmetrische und $1,5$ für symmetrische Probleme. Zum Vergleich diente die iterative Verfeinerung, welche komplett in doppelter Genauigkeit durchgeführt wurde. Nachdem der Speedup für direkte Löser aussagekräftig war, untersuchten sie auch die Verwendung von iterativen Lösungsverfahren. Hierbei kamen Krylov-Unterraum-Verfahren zum Einsatz.

Gemischt genaue Inner-Outer-Iterationen

Für die weitere Untersuchung verwendeten Buttari et al. [12] das CG-Verfahren für symmetrisch positiv definite Probleme und das GMRES-Verfahren für unsymmetrische Probleme. Im

Gegensatz zur iterativen Verfeinerung, bei der im dritten Schritt lediglich eine einfache Korrektur durchgeführt wird, verwendeten sie ein vorkonditioniertes CG- bzw. FGMRES-Verfahren. Solche Verfahren, die aus zwei jeweils iterativen Verfahren bestehen, werden auch „Inner-Outer"-Iterationen genannt. Im Fall von symmetrisch positiv definiten Problemen führt dies zu CG-PCG-Iterationen, bei denen die äußere Iteration symmetrisch vorkonditioniert wird. Beim GMRES-Verfahren erhält man analog GMRES-FGMRES-Iterationen. Die inneren Iterationen wurden von Buttari et al. in einfacher und die äußeren in doppelter Genauigkeit gerechnet. Bei Verwendung der gemischt genauen CG-PCG-Iterationen konnte ein Speedup von bis zu 1,6 gemessen werden. Zum Vergleich diente das CG-Verfahren in doppelter Genauigkeit. Beim GMRES-FGMRES-Verfahren lag der Speedup zum Teil bei über 6,0.

Iterative Verfeinerung mit Mehrgitterverfahren

Göddeke et al. [13] untersuchten die Verwendung von Mehrgitterverfahren, um das Gleichungssystem im zweiten Schritt der iterativen Verfeinerung zu lösen. Zur damaligen Zeit wurde lediglich der Datentyp Single auf den Grafikkarten nativ unterstützt, weshalb Göddeke et al. die Grafikkarte als Koprozessor verwendeten. Hierbei wurden die Schritte eins und drei der iterativen Verfeinerung auf der CPU durchgeführt, auf welcher die doppelte Genauigkeit nativ unterstützt wird. In ihren Untersuchungen erreichte die gemischt genaue iterative Verfeinerung einen Speedup von mehr als 4.0 gegenüber einem CPU-basierten Mehrgitterverfahren in doppelter Genauigkeit.

Gemischt genaue Mehrgitterverfahren

Die Untersuchungen, die Gegenstand dieser Arbeit sind, gehen orthogonal zu den bisherigen Überlegungen vor. Bei der Verwendung von unterschiedlichen Datentypen stellt sich immer die Frage, welche Teile eines Verfahrens in welcher Genauigkeit durchgeführt werden sollen. Mehrgitterverfahren bieten hier eine natürliche Herangehensweise, da sie wie die Datentypen selbst hierarchisch aufgebaut sind. Im Gegensatz zur iterativen Verfeinerung mit Mehrgitterverfahren sollen also nicht gesamte Mehrgitterverfahren in einer Genauigkeit durchgeführt und wenige Update-Terme in einer anderen Genauigkeit aufsummiert, sondern das Verfahren selbst in eine gemischt genaue Form gebracht werden. Um diese Idee weiter auszuführen, ist es notwendig, die grundlegenden Eigenschaften und den Aufbau von Mehrgitterverfahren näher zu betrachten.

2 Mehrgitterverfahren

Im Folgenden sollen die Mehrgitteridee dargestellt und entsprechende Lösungsverfahren hergeleitet werden. Hierbei erfolgt zunächst ein grober Überblick über den Aufbau und die Struktur solcher Verfahren. Anschließend werden die Vor- und Nachteile gegenübergestellt. Es folgt ein Überblick über die restlichen Unterkapitel, deren Fokus auf den einzelnen Komponenten von Mehrgitterverfahren liegt. Die Beschreibung dieses Kapitels orientiert sich inhaltlich am Vorlesungsskript „Wissenschaftliches Rechnen“ von Dominik Göddeke [14].

Aufbau und Struktur

Eine der wesentlichen Eigenschaften von Mehrgitterverfahren ist, wie der Name bereits vermuten lässt, die Verwendung von mehreren Gittern. Liegt das Ausgangsproblem $\mathbf{Av} = \mathbf{f}$ auf einem Gitter der Weite h vor, so werden in Mehrgitterverfahren auch gröbere Gitter, d.h. Gitter mit einer Weite von $H > h$, verwendet. Diese immer gröber werdenden Gitter werden rekursiv durchlaufen, wobei mit dem feinsten Gitter begonnen wird. In jedem Schritt wird als Erstes ein einfaches iteratives Lösungsverfahren verwendet, um „bestimmte Anteile“ des Fehlers zu entfernen. Je nach Gitter werden dabei unterschiedliche Anteile entfernt. Von den iterativen Lösungsverfahren werden jedoch nur wenige Schritte durchgeführt, weshalb sie im Kontext von Mehrgitterverfahren auch *Relaxationsverfahren* genannt werden. Danach wird das Problem auf das nächstgröbere Gitter transferiert und erneut ein einfaches iteratives Lösungsverfahren durchlaufen.

Da für ein Grobgitter $H > h$ gefordert wird, führt dies zur schrittweisen Reduzierung der Problemgröße. Sobald die Dimension klein genug ist, kann auf dem gröbsten Gitter das Problem direkt gelöst werden. Diese Lösung dient dann als Approximation für das nächstfeinere Gitter. Durch den Transfer vom groben auf das feine Gitter entstehen jedoch Fehlerterme, welche durch erneutes Anwenden eines Relaxationsverfahrens eliminiert werden. Insgesamt stellt sich ein sog. Mehrgitter-Zyklus ein.

Vorteile

Im Gegensatz zu anderen iterativen Lösungsverfahren sind Mehrgitterverfahren asymptotisch optimal. Das bedeutet, dass die Konvergenzgeschwindigkeit *nicht* von der ursprünglichen Gitterweite h abhängt. Damit ist pro Unbekannte lediglich eine konstante Anzahl an arithmetischen Operationen zur Berechnung einer Lösung notwendig, unabhängig davon, wie fein das Ausgangsproblem diskretisiert wird. Bei den ebenfalls häufig zum Einsatz kommenden vorkonditionierten Krylov-Unterraum-Verfahren hängt die Konvergenzgeschwindigkeit hingegen von der Konditionszahl der Koeffizientenmatrix und dadurch von der Gitterweite ab. Je feiner das kontinuierliche Problem diskretisiert wird, desto mehr Iterationen (und dadurch arithmetische Operationen) werden zur Konvergenz benötigt. Beim CG-Verfahren liegt die Konvergenzrate in $\mathcal{O}(1-h)$ und es werden $\mathcal{O}(n^{3/2}\log\varepsilon)$ Operationen benötigt, um eine Genauigkeit von ε zu erreichen. Bei Mehrgitterverfahren ist die Konvergenzrate hingegen konstant ($c < 1$) und unabhängig von h. Die Anzahl an Operationen beläuft sich dabei auf $\mathcal{O}(n\log\varepsilon)$. Beim sog. „Full Multigrid (FMG)“-Verfahren sind lediglich $\mathcal{O}(n)$ Operationen nötig, um eine beliebige Genauigkeit ε zu erreichen. Auf den Beweis der asymptotischen Optimalität wird in dieser Arbeit nicht eingegangen. Er lässt sich jedoch z.B. in [15] nachlesen.

Nachteile

Den Vorteilen stehen allerdings auch Nachteile gegenüber. Im Fall von Mehrgitterverfahren bedeutet dies, dass es nicht *das* Mehrgitterverfahren gibt, sondern dass die einzelnen Komponenten problemspezifisch gewählt werden müssen. Dies beinhaltet insbesondere die Wahl

der Relaxationsverfahren, die Anzahl an Relaxationsschritten, die Vergröberungsstrategie, die Gittertransferstrategie sowie die Wahl des Lösungsverfahrens für das Problem auf dem gröbsten Gitter. Um herauszufinden, welche Komponenten für ein konkretes Problem geeignet sind, werden meist numerische Experimente durchgeführt.

Übersicht

Die Idee von Mehrgitterverfahren beruht auf zwei zentralen Grundkonzepten, der *Fehlerdämpfung* und der *Grobgitterkorrektur*. Diese beiden Konzepte sind dafür verantwortlich, dass ein rekursives Verfahren, wie zu Beginn dieses Kapitels beschrieben, funktioniert. Ebenfalls gewinnen Mehrgitterverfahren ihre Stärken aus diesen beiden Grundkonzepten. Im Folgenden wird deshalb die Idee der Fehlerdämpfung exemplarisch anhand eines Relaxationsverfahrens aufgezeigt. Im Anschluss wird die Idee der Grobgitterkorrektur motiviert und mögliche Transferstrategien zwischen den Gittern aufgezeigt. Die Kombination aus Fehlerdämpfung und Grobgitterkorrektur führt dann zum Zweigitterverfahren, welches im Anschluss zu allgemeinen Mehrgitterverfahren verallgemeinert wird. Zum Abschluss des Kapitels wird das Mehrgitterverfahren durch geschickte Wahl von Startlösungen in das sog. „Full Multigrid (FMG)“-Verfahren überführt.

2.1 Fehlerdämpfung

Das Konzept der Fehlerdämpfung lässt sich durch eine Analyse der Fehlerentwicklung bei einfachen iterativen Lösungsverfahren herleiten. Dies erfolgt exemplarisch für das Jacobi-Verfahren und lässt sich auf viele weitere Verfahren ausweiten.

Jacobi-Verfahren

Beim Jacobiverfahren wird die Matrix $\mathbf{A}$ des zu lösenden Gleichungssystems $\mathbf{Av} = \mathbf{f}$ in einen Diagonalteil sowie die obere und untere Rest-Dreiecksmatrix aufgeteilt:

$$\mathbf{A} = \mathbf{D} - \mathbf{L} - \mathbf{U}.$$

Hierbei beschreibt $\mathbf{D}$ die Diagonaleinträge, $\mathbf{L}$ die untere und $\mathbf{U}$ die obere Dreiecksmatrix. Setzt man diese Aufteilung in das Gleichungssystem ein und bringt den Nicht-Diagonalteil auf die rechte Seite, so erhält man

$$\begin{aligned}(\mathbf{D} - \mathbf{L} - \mathbf{U})\mathbf{v} &= \mathbf{f} \\ \mathbf{Dv} &= (\mathbf{L} + \mathbf{U})\mathbf{v} + \mathbf{f}.\end{aligned}$$

Da sich die Diagonalmatrix $\mathbf{D}$ einfach invertieren lässt, erhält man als Iterationsvorschrift

$$\mathbf{v}^{(m+1)} = \mathbf{P}\mathbf{v}^{(m)} + \mathbf{g}, \tag{4}$$

wobei $\mathbf{P} = \mathbf{D}^{-1}(\mathbf{L} + \mathbf{U})$ die Iterationsmatrix und $\mathbf{g} = \mathbf{D}^{-1}\mathbf{f}$ ein konstanter Vektor ist.

Das Jacobi-Verfahren kann als „Eingitterverfahren“ verwendet werden, um lineare Gleichungssysteme zu lösen. Im Sinne der Mehrgitteridee wird es jedoch als Relaxationsverfahren verwendet und bei dessen Anwendung nicht vollständig durchgeführt. Üblicherweise werden pro Gitter zwei bis vier Iterationen durchgeführt. Wie zu Beginn des zweiten Kapitels beschrieben soll die Anwendung eines Relaxationsverfahrens dazu führen, dass „bestimmte“ Anteile aus dem Fehler entfernt werden. Hierfür betrachten wir zunächst, wie sich die Anwendung von Jacobi-Schritten auf den Fehler auswirkt.

Da es sich bei der exakten Lösung $\mathbf{v}$ um einen Fixpunkt der Gleichung (4) handelt, gilt

$$\mathbf{v} = \mathbf{P}\mathbf{v} + \mathbf{g}. \tag{5}$$

Die Subtraktion der Gleichungen (4) von (5) liefert

$$\mathbf{e}^{(m+1)} = \mathbf{P}\mathbf{e}^{(m)}, \tag{6}$$

bzw. nach rekursiver Anwendung

$$\mathbf{e}^{(m+1)} = \mathbf{P}^{m+1}\mathbf{e}^{(0)}.$$

Geht man zu einer induzierten, verträglichen Matrixnorm $||\cdot||$ über, so erhält man

$$||\mathbf{e}^{(m+1)}|| \leq ||\mathbf{P}||^{m+1}||\mathbf{e}^{(0)}||$$

und $||\mathbf{P}|| < 1$ als Konvergenzbedingung. Je kleiner die Norm der Iterationsmatrix ist, desto schneller konvergiert das Verfahren. Die Konvergenzgeschwindigkeit ist gleichbedeutend mit der Geschwindigkeit, mit der der Fehler reduziert wird.

Gedämpftes Jacobi-Verfahren

In Mehrgitterverfahren wird das Jacobi-Verfahren meist in seiner gedämpfter Form verwendet. Hierzu wird ein Dämpfungsparameter ω eingeführt, welcher zu einer gewichteten Summe in der Iterationsvorschrift führt:

$$\mathbf{v}^{(m+1)} = ((1-\omega)\mathbf{I} + \omega\mathbf{P})\mathbf{v}^{(m)} + \omega\mathbf{g}.$$

Als Iterationsmatrix für das gedämpfte Verfahren erhält man somit $\mathbf{P}_\omega = (1-\omega)\mathbf{I} + \omega\mathbf{P}$. Für $\omega = 1$ geht das gedämpfte in das ungedämpfte Jacobi-Verfahren über. Der Parameter ω kann zunächst frei gewählt werden.

Um das Fehlerverhalten genauer zu untersuchen, können die Eigenwerte der Iterationsmatrix $\mathbf{P}_\omega$ betrachtet werden. Im Folgenden wird hierfür das Poisson-Problem in einer Dimension aus Kapitel 1.2 herangezogen. In zwei Dimensionen lässt sich die Analyse analog durchführen, die zusätzliche Komplexität durch Doppelindizes ist dabei jedoch hinderlich.

Es lässt sich zeigen, dass

$$\lambda_i = 1 - 2\omega\sin^2\left(\frac{1}{2}ih\pi\right) \qquad \text{mit} \quad i = 1\ldots n$$

die Eigenwerte und

$$(\mathbf{w}_i)_j = \sin\left(ijh\pi\right) \qquad \text{mit} \quad i = 1\ldots n \quad \text{und} \quad j = 1\ldots n$$

die zugehörigen Eigenvektoren der Iterationsmatrix $\mathbf{P}_\omega$ sind. Weiter zeigt sich, dass die Eigenvektoren von $\mathbf{A}$ mit denen von $\mathbf{P}_\omega$ übereinstimmen. Aufgrund der Form der Eigenvektoren werden diese auch häufig als Fehlerfrequenzen bezeichnet. Da $\mathbf{P}_\omega$ symmetrisch ist gilt

$$||\mathbf{P}||_2 = \lambda_{max} = 1 - 2\omega\sin^2\left(\frac{1}{2}h\pi\right).$$

In erster Näherung gilt $\sin(x) \approx x$ und damit

$$||\mathbf{P}||_2 \approx 1 - \frac{1}{2}\omega h^2\pi^2 = \mathcal{O}(1-\omega h^2). \tag{7}$$

Dies ist genau die h^2-abhängige Konvergenz des Jacobi-Verfahrens. Im Konkreten bedeutet das, dass für eine konstante Fehlerreduktion auf einem Gitter der Weite $h/2$ vier mal so viele Iterationen benötigt werden wie auf einem Gitter der Weite h. Um den Diskretisierungsfehler zu verringern, muss h möglichst klein gewählt werden. Dies führt jedoch zu einem Anstieg der Iterationen und ebenfalls zur Vergrößerung der Koeffizientenmatrix und des Lösungsvektors.

Eigenvektor-Expansion des Fehlers

Entwickelt man den Fehler $\mathbf{e}^{(0)}$ nach den Eigenvektoren von $\mathbf{A}$, so erhält man

$$\mathbf{e}^{(0)} = \sum_{i=1}^{n} c_i \mathbf{w}_i.$$

Mit Gleichung (6) folgt damit

$$\mathbf{e}^{(m+1)} = \sum_{i=1}^{n} c_i \mathbf{P}_\omega^{m+1} \mathbf{w}_i = \sum_{i=1}^{n} c_i \lambda_i^{m+1} \mathbf{w}_i.$$

Der Fehler lässt sich damit immer als eine Überlagerung unterschiedlicher Sinusfunktionen (Fehlerfrequenzen) schreiben. Weiter zeigt sich in dieser Darstellung, dass Fehlerfrequenzen, die zu betragsmäßig großen Eigenwerten gehören, nur sehr langsam reduziert werden.

Im Folgenden werden die Fehlerfrequenzen in die Kategorien

- *niederfrequent* oder *glatt* ($0 \leq i \leq n/2$) und
- *hochfrequent, oszillierend* oder *rau* ($n/2 \leq i \leq n$)

aufgeteilt. In Abbildung 5 sind die Eigenwerte der Iterationsmatrix $\mathbf{P}_\omega$ für unterschiedliche Werte von ω dargestellt. Beim ungedämpften Jacobi-Verfahren ($\omega = 1$) werden die Fehlerfrequenzen im mittleren Bereich ($i \approx 1/2\ n$) schnell reduziert, während die sehr hoch- und sehr niederfrequenten Anteile (kleine i und große i) kaum reduziert werden. Für eine Wahl von $\omega = 2/3$ werden alle Anteile im hochfrequenten Bereich optimal relaxiert. Weiter lässt sich zeigen, dass in diesem Fall $|\lambda_i| \leq 1/3$ gilt. Die hochfrequenten Fehleranteile werden in jedem Schritt somit mindestens um den Faktor $1/3$ gedämpft. Bemerkenswert ist, dass dieser Faktor unabhängig von h ist. Mehrgitterverfahren erhalten einen großen Teil ihrer Stärke aus dieser h-unabhängigen Dämpfungseigenschaft. Niederfrequente Fehleranteile können jedoch für keine Wahl von ω hinreichend schnell gedämpft werden.

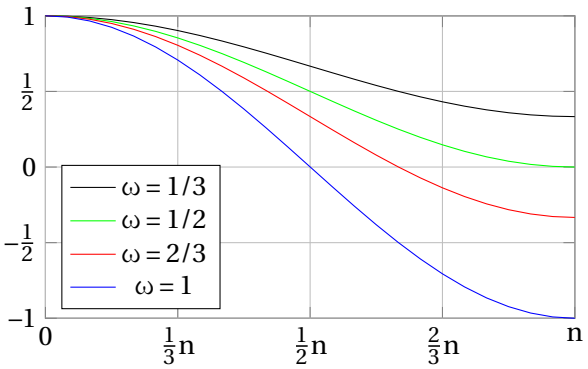


Abbildung 5: Eigenwerte der Iterationsmatrix $\mathbf{P}_\omega$ für unterschiedliche Werte des Relaxationsparameter ω. Die diskreten Eigenwerte ($1 \leq i \leq n$) sind aufgrund der besseren Visualisierung kontinuierlich gezeichnet.

Andere Relaxationsverfahren

Neben dem (gedämpften) Jacobi-Verfahren lassen sich auch andere iterative Lösungsverfahren als Relaxationsverfahren einsetzen. Die Fehleranalyse ist dabei meist technisch aufwendiger; so fallen beispielsweise beim Gaus-Seidel-Verfahren die Eigenvektoren der Iterations- und Koeffizientenmatrix nicht zusammen. In der Praxis wird häufig eine Gauß-Seidel-Relaxation

durchgeführt, da formal gezeigt werden kann, dass das Gauß-Seidel-Verfahren zu einer schnelleren Dämpfung des Fehlers führt. Bei zeilenweiser Nummerierung ist das Verfahren jedoch nur schlecht parallelisierbar, weshalb beim Gauß-Seidel-Verfahren meist die red-black-Nummerierung verwendet wird.

Zusammenfassung

Die wichtigste Erkenntnis dieses Unterkapitels soll noch einmal kurz zusammengefasst werden: Mit Relaxationsverfahren wie dem Jacobi- oder Gauß-Seidel-Verfahren können hochfrequente Fehleranteile schnell reduziert werden. Niederfrequente Anteile lassen sich dadurch nicht eliminieren. Wie mit diesen Fehleranteilen umgegangen werden kann, beschreibt das zweite Grundprinzip der Mehrgitteridee.

2.2 Grobgitterkorrektur

Die Analyse des (gedämpften) Jacobi-Verfahrens hat gezeigt, dass sich niederfrequente Fehleranteile prinzipiell nicht schnell dämpfen lassen. Gleichung (7) zeigt sogar, dass diese Fehleranteile umso langsamer gedämpft werden, je feiner das zugrundeliegende Gitter gewählt wird. Dies legt allerdings auch die folgende Vermutung nahe: Hätte man das Gitter gröber gewählt, so würden die niederfrequenten Fehleranteile womöglich schneller gedämpft. Genau diese Idee steckt hinter der Grobgitterkorrektur.

Im Folgenden wird erneut das eindimensionale Poisson-Problem (2) betrachtet. Wählt man als rechte Seite $\mathbf{f} = \mathbf{0}$, so ist die exakte Lösung des Gleichungssystems trivialerweise $\mathbf{v} = \mathbf{0}$. Eine Startlösung $\mathbf{v}^{(0)}$ entspricht somit dem initialen Fehler:

$$\mathbf{e}^{(0)} = \mathbf{v} - \mathbf{v}^{(0)} = -\mathbf{v}^{(0)}.$$

Um unterschiedliche Frequenzen besser unterscheiden zu können, wählen wir als Startlösung eine reine Sinusschwingung, d.h.

$$\mathbf{v}^{(0)} = \begin{bmatrix} \sin(1 \cdot 4\pi h) \\ \sin(2 \cdot 4\pi h) \\ \vdots \\ \sin((n-1) \cdot 4\pi h) \\ \sin(n \cdot 4\pi h) \end{bmatrix}.$$

Abbildung 6 zeigt die Startlösung (also den initialen Fehler) auf einem Gitter der Weite $h = 1/12$ und $h = 1/6$. Betrachtet man die Schwingung auf dem gröberen Gitter, so erscheint sie oszillierender in Bezug auf h. Hier kann ein Relaxationsverfahren den ursprünglich niederfrequenten Fehler dämpfen. Da niederfrequente Fehler auf einem gröberen Gitter oszillierender erscheinen, lässt sich die Frage stellen, wie sich hochfrequente Fehler auf einem gröberen Gitter verhalten. Es lässt sich zeigen, dass diese wieder niederfrequent werden. Da bei Mehrgitterverfahren jedoch vom feinsten Gitter ausgegangen und in jedem Schritt eine Fehlerdämpfung durchgeführt wird, sind die ursprünglich hochfrequenten Fehleranteile bereits hinreichend gedämpft.

Um die Idee der Grobgitterkorrektur für ein Lösungsverfahren zu verwenden, kann die Residuumsgleichung

$$\mathbf{Ae} = \mathbf{r}$$

betrachtet werden. Es lässt sich zeigen, dass die Relaxation auf dem Ausgangsproblem $\mathbf{Av} = \mathbf{f}$ für beliebige Startlösungen $\mathbf{v}^{(0)}$ äquivalent zur Relaxation der Residuumsgleichung mit Startlösung $\mathbf{e}^{(0)} = \mathbf{0}$ ist. Damit wird es möglich, direkt auf dem Fehler zu relaxieren, indem nicht

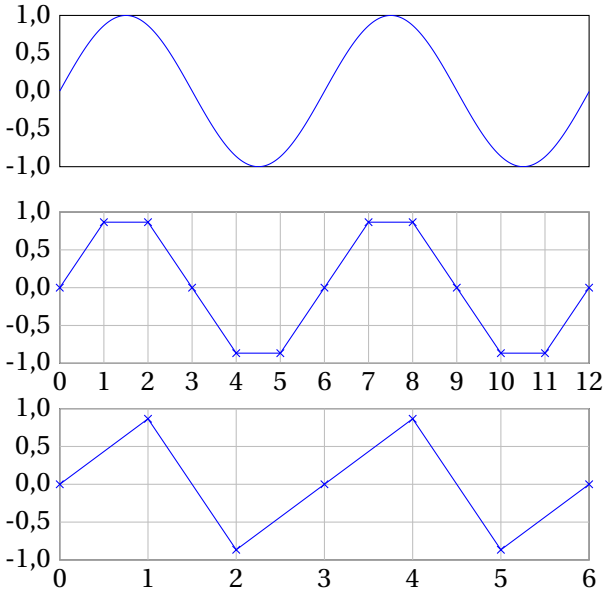


Abbildung 6: Repräsentation einer kontinuierlichen Sinusschwingung (oben), auf einem Gitter der Weite $h = 1/12$ (mitte) und $h = 1/6$ (unten).

das Ausgangsproblem, sondern die Residuumsgleichung auf einem gröberen Gitter betrachtet wird. Um ein Verfahren zu erhalten, kann analog zur iterativen Verfeinerung aus Kapitel 1.5 vorgegangen werden. Der einzige Unterschied besteht darin, dass das Hilfsproblem auf einem gröberen Gitter relaxiert wird. Verwendet man als gröberes Gitter Ω_{2h}, d.h. ein Gitter mit der doppelten Gitterweite, so erhält man für einen Schritt der Grobgitterkorrektur Algorithmus 2.

Algorithmus 2 Grobgitterkorrektur

Relaxiere $\mathbf{A}\mathbf{v} = \mathbf{f}$ auf Ω_h mit beliebigem $\mathbf{v}^{(0)} \longrightarrow$ Approximation $\mathbf{v}_h$
Berechne $\mathbf{r} = \mathbf{f} - \mathbf{A}\mathbf{v}_h$
Relaxiere $\mathbf{A}\mathbf{e} = \mathbf{r}$ auf Ω_{2h} mit $\mathbf{e}^{(0)} = \mathbf{0} \longrightarrow$ Approximation $\mathbf{e}_{2h}$
Korrigiere $\mathbf{v}_h = \mathbf{v}_h + \mathbf{e}_{2h}$

Neben der Verdopplung der Gitterweite existieren auch andere Vergröberungsstrategien. Im Zuge dieser Arbeit wird jedoch ausschließlich die Verdopplung verwendet, welche auch als *Standard-Vergröberung* bezeichnet wird.

Wesentliche Fragen bleiben in Algorithmus 2 jedoch noch offen: Wie sieht der Operator $\mathbf{A}$ auf dem Grobgitter aus und wie werden das Residuum und der Fehler zwischen den Gittern transferiert? Zur besseren Übersicht wird das Subskript nun auch für den Operator und das Residuum eingeführt.

Der Operator $\mathbf{A}_{2h}$ auf dem Grobgitter kann auf die gleiche Weise konstruiert werden wie auf dem Feingitter. Der einzige Unterschied ist die Gitterweite. In Stencil-Notation bedeutet dies

$$\mathbf{A}_{2h} = \frac{1}{(2h)^2} \begin{bmatrix} & -1 & \\ -1 & 4 & -1 \\ & -1 & \end{bmatrix}.$$

Das Residuum auf dem Grobgitter erhält man durch *Restriktion* des Residuums auf dem Feingitter. Formal kann ein Restriktionsoperator $\mathbf{I}_h^{2h}$ eingeführt werden, der das Residuum vom fei-

nen auf das grobe Gitter transferiert:

$$\hat{\mathbf{r}}_{2h} := \mathbf{I}_h^{2h}\mathbf{r}_h.$$

An dieser Stelle sei darauf hingewiesen, dass $\hat{\mathbf{r}}_{2h}$ nicht das Residuum des Gleichungssystems auf dem Grobgitter darstellt, sondern eine Approximation des Residuums $\mathbf{r}_h$ auf dem Grobgitter Ω_{2h}. Zur Verdeutlichung dient die Notation $\hat{\cdot}$, um zwischen Residuum auf dem Grobgitter und Approximation zu unterscheiden.

Für den Fehler kann der umgekehrte Weg gegangen werden, da dieser vom groben auf das feine Gitter transferiert wird. Dies geschieht mittels *Prolongation*, d.h. einer Interpolation zwischen den bekannten Grobgitterpunkten. Der Prolongationsoperator $\mathbf{I}_{2h}^h$ genügt damit

$$\hat{\mathbf{e}}_h := \mathbf{I}_{2h}^h\mathbf{e}_{2h}.$$

Für die konkrete Wahl der Gittertransfer-Operatoren gibt es mehrere Möglichkeiten, welche je nach Anwendungsfall besser oder schlechter geeignet sind. Im Folgenden werden die gängigsten Operatoren dargestellt, welche für elliptische Probleme geeignet sind.

Restriktion

Die einfachste Möglichkeit, das Residuum auf ein gröberes Gitter zu restringieren, ist die Injektion. Hierbei werden die bekannten Gitterpunkte einfach übernommen, sodass

$$\hat{\mathbf{r}}_{2h}(x,y) = (\mathbf{I}_h^{2h}\mathbf{r}_h)(x,y) = \mathbf{r}_h(x,y) \qquad \text{für} \qquad (x,y) \in \Omega_{2h} \subset \Omega_h$$

gilt. Für Diskretisierungen mit finiten Differenzen können Restriktionsoperatoren definiert werden, welche die umliegenden Gitterpunkte miteinbeziehen. Beispiele hierfür sind der volle bzw. halbe Gewichtungsoperator,

$$\mathbf{I}_{h,voll}^{2h} = \frac{1}{16}\begin{bmatrix}1 & 2 & 1\\ 2 & 4 & 2\\ 1 & 2 & 1\end{bmatrix} \qquad \text{bzw.} \qquad \mathbf{I}_{h,halb}^{2h} = \frac{1}{8}\begin{bmatrix} & 1 & \\ 1 & 4 & 1\\ & 1 & \end{bmatrix}.$$

Prolongation

Für die Prolongation können typische Interpolationsverfahren verwendet werden, wie die bilineare Interpolation in zwei Dimensionen. Für Feingitterpunkte werden die Werte der Grobgitterpunkte übernommen und bei Grobgitterpunkten wird bilinear interpoliert. Die Anwendungsvorschrift des Prolongationsoperators lautet somit:

$$\mathbf{I}_{2h}^h\hat{\mathbf{e}}_{2h}(x,y) = \begin{cases} \hat{\mathbf{e}}_{2h}(x,y) & \text{für } \bullet \\ \frac{1}{2}(\hat{\mathbf{e}}_{2h}(x,y+h) + \hat{\mathbf{e}}_{2h}(x,y-h)) & \text{für } \square \\ \frac{1}{2}(\hat{\mathbf{e}}_{2h}(x+h,y) + \hat{\mathbf{e}}_{2h}(x-h,y)) & \text{für } \diamond \\ \frac{1}{4}(\hat{\mathbf{e}}_{2h}(x+h,y+h) + \hat{\mathbf{e}}_{2h}(x+h,y-h) & \text{für } \circ \\ \quad +\hat{\mathbf{e}}_{2h}(x-h,y+h) + \hat{\mathbf{e}}_{2h}(x-h,y-h)) & \end{cases}$$

Die vier Symbole kennzeichnen die unterschiedlichen Feingitterpunkte, welche in Abbildung 7 mit den entsprechenden Gewichtungen dargestellt sind.

Zusammenfassung

Die Betrachtung des Fehlers auf unterschiedlichen Gittern zeigt, dass niederfrequente Anteile auf gröberen Gittern deutlich oszillierender erscheinen. Hier kann ein Relaxationsverfahren die Fehlerterme dämpfen. Residuen werden von feinen auf gröbere Gitter restringiert und Fehler von gröberen auf feine Gitter interpoliert.

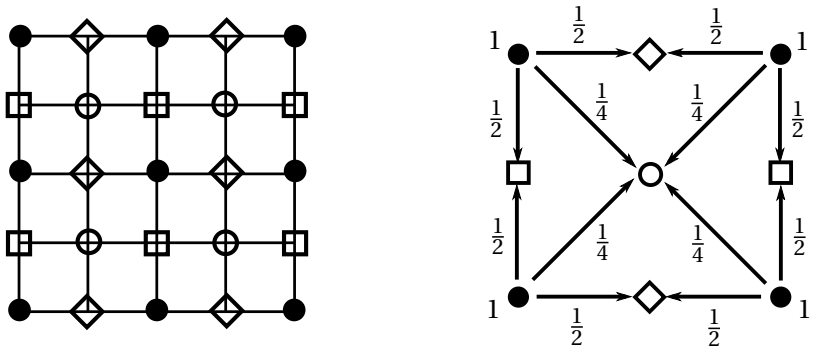


Abbildung 7: Links: Klassifizierung unterschiedlicher Feingitterpunkte aus der Sicht von Grobgitterpunkten •. Rechts: Gewichtungen für die Berechnung der unterschiedlichen Feingitterpunkte[1].

2.3 Mehrgitterverfahren

Wird die Grobgitterkorrektur mit der Fehlerdämpfung verbunden, so erhält man Mehrgitterverfahren. Im Folgenden beschränken wir uns zunächst auf den Fall zweier Gitter – ein feines Gitter Ω_h und ein grobes Gitter Ω_{2h}. Man erhält ein sog. *Zweigitterverfahren.*

Zweigitterverfahren

Zu Beginn des Verfahrens wird eine feste Anzahl an Relaxationsschritten durchgeführt, um die oszillierenden Fehleranteile zu glätten. Mit dem Parameter k_{vor} wird die Anzahl an Vorglättungsschritten bezeichnet. Im Anschluss wird das Residuum berechnet und mittels Restriktion auf das grobe Gitter transferiert. Auf Ω_{2h} wird dann die Residuumsgleichung gelöst. Die berechnete Lösung wird nun mittels Interpolation auf das feine Gitter Ω_h transferiert. Nach k_{nach} Relaxationsschritten, um die durch Interpolation hinzugekommenen Fehlerterme zu glätten, wird die Approximation auf dem Feingitter Ω_h korrigiert. In Algorithmus 3 ist ein Schritt des Zweigitterverfahrens aufgeführt.

Prüft man nach jeder Vorglättung die Konvergenz, beispielsweise indem die Norm des Residuums berechnet wird, so erhält man ein iteratives Lösungsverfahren. In Abbildung 8 auf der linken Seite ist der entstehende Zweigitterzyklus schematisch dargestellt.

Mehrgitterzyklen

Ausgehend von Zweigitterverfahren lassen sich Mehrgitterverfahren durch Rekursion aufbauen. Konkret bedeutet dies, dass in Algorithmus 3 innerhalb der Grobgitterkorrektur nicht die Residuumsgleichung direkt gelöst wird, sondern ein weiterer Zweigitterzyklus aufgerufen wird. Diese Rekursion kann nun so lange fortgeführt werden, bis das Problem eine Größe erreicht hat, die sich einfach lösen lässt. Ausgehend von dem gröbsten Gitter Ω_{min} werden die Korrekturen dann wieder auf die feineren Gitter transferiert. Dadurch entsteht eine Gitterhierarchie

$$\Omega_h \to \Omega_{2h} \to \Omega_{4h} \to \cdots \to \Omega_{min} \to \cdots \to \Omega_{4h} \to \Omega_{2h} \to \Omega_h,$$

die rekursiv durchlaufen wird. Der entstehende Zyklus wird *V-Zyklus* genannt und ist in Abbildung 8 rechts oben schematisch dargestellt.

Es ist klar, dass neben dem V-Zyklus auch weitere Zyklen möglich sind, wenn mehr als zwei Gitter verwendet werden. Mit drei Gittern lässt sich ein *W-Zyklus* kreieren, indem auf dem mittleren Gitter nicht ein, sondern zwei Zweigitterzyklen durchgeführt werden. Mit vier Gittern gibt

[1] Die Abbildung stammt aus [14] und wurde leicht abgeändert

Algorithmus 3 Zweigitterverfahren

Berechnung von $\mathbf{v}_h^{(m+1)}$ aus $\mathbf{v}_h^{(m)}$:

Vorglättung:

Berechne $\mathbf{v}_h^{(m+\frac{1}{3})}$ durch k_{vor}-malige Anwendung eines Relaxationsverfahrens auf $\mathbf{A}_h\mathbf{v}_h^{(m)} = \mathbf{f}_h$

Grobgitterkorrektur:

Berechne das Residuum $\mathbf{r}_h = \mathbf{f}_h - \mathbf{A}_h\mathbf{v}_h^{(m+\frac{1}{3})}$

Restringiere das Residuum auf das Grobgitter Ω_{2h} : $\hat{\mathbf{r}}_{2h} = \mathbf{I}_h^{2h}\mathbf{r}_h$

Löse $\mathbf{A}_{2h}\hat{\mathbf{e}}_{2h} = \hat{\mathbf{r}}_{2h}$ mit $\hat{\mathbf{e}}_{2h}^{(0)} = \mathbf{0}$

Interpoliere die Korrektur zurück auf Ω_h : $\hat{\mathbf{e}}_h = \mathbf{I}_{2h}^h\hat{\mathbf{e}}_{2h}$

Korrigiere die Approximation: $\mathbf{v}_h^{(m+\frac{2}{3})} = \mathbf{v}_h^{(m+\frac{1}{3})} + \hat{\mathbf{e}}_h$

Nachglättung:

Berechne $\mathbf{v}_h^{(km+1)}$ durch k_{nach}-malige Anwendung eines Relaxationsverfahrens auf $\mathbf{A}_h\mathbf{v}_h^{(m+\frac{2}{3})} = \mathbf{f}_h$

es noch weitere Zyklen, von denen in dieser Arbeit lediglich der *F-Zyklus* beschrieben wird. Für praxisrelevante Probleme ist der W-Zyklus meist sehr aufwändig, weshalb der F-Zyklus als eine Mischung aus V- und W-Zyklus in der Praxis gerne verwendet wird. Für die Hinrichtung, d.h. die Rekursion in die Tiefe, wird dabei ein V-Zyklus verwendet und für den Rückweg ein W-Zyklus. In Abbildung 8 rechts unten ist der F-Zyklus für ein Viergitterverfahren dargestellt.

Mehrgitterparameter

Nachdem die unterschiedlichen Mehrgitterzyklen betrachtet wurden, sollen an dieser Stelle die Parameter von Mehrgitterverfahren noch einmal aufgeführt werden. Wie bereits zu Beginn dieses Kapitels beschrieben, hängt der Erfolg von Mehrgitterverfahren maßgeblich von der problemgerechten Wahl der Mehrgitterkomponenten ab. Zieht man die Verwendung von Mehrgitterverfahren für die Lösung eines Problems in Betracht, so müssen die folgenden Punkte berücksichtigt werden:

- Die Anzahl an Vor- und Nachglättungsschritten, k_{vor} bzw. k_{nach}: Diese müssen nicht für jedes Gitter gleich gewählt werden. Vielmehr können die beiden Parameter auch von der aktuellen Tiefe abhängen. Wie in Kapitel 2.1 beschrieben ist es allerdings nicht sinnvoll, sehr viele Glättungsschritte pro Gitter durchzuführen, da sich niederfrequente Fehleranteile prinzipiell nicht schnell dämpfen lassen.
- Die Wahl der Relaxationsverfahren: In der Praxis wird für die Vor- und Nachglättung häufig dasselbe Relaxationsverfahren verwendet. Unterschiedliche Verfahren sind jedoch explizit nicht ausgeschlossen.
- Das Verfahren zum Lösen der Residuumsgleichung auf dem gröbsten Gitter.
- Die Wahl der Vergröberungsstrategie: Diese stellt in vielerlei Hinsicht einen Ansatzpunkt für weitere Optimierungen dar. Je nach Problemfall sind in den Fehlertermen manche Frequenzen gar nicht enthalten, sodass eine Relaxation auf dem zugehörigen Gitter kaum Auswirkungen hat. Im Gegensatz dazu kann die Standard-Vergröberung für manche Probleme bereits zu grob sein, sodass nicht alle Fehlerspektren optimal gedämpft werden.
- Der Restriktionsoperator: Beispiele hierfür sind der volle und halbe Gewichtungsoperator.
- Der Prolongationsoperator: Gängige Interpolationsverfahren wie z.B. die bilineare Interpolation in zwei Dimensionen.
- Die Art der Mehrgitterzyklen und die Tiefe einer Iteration.

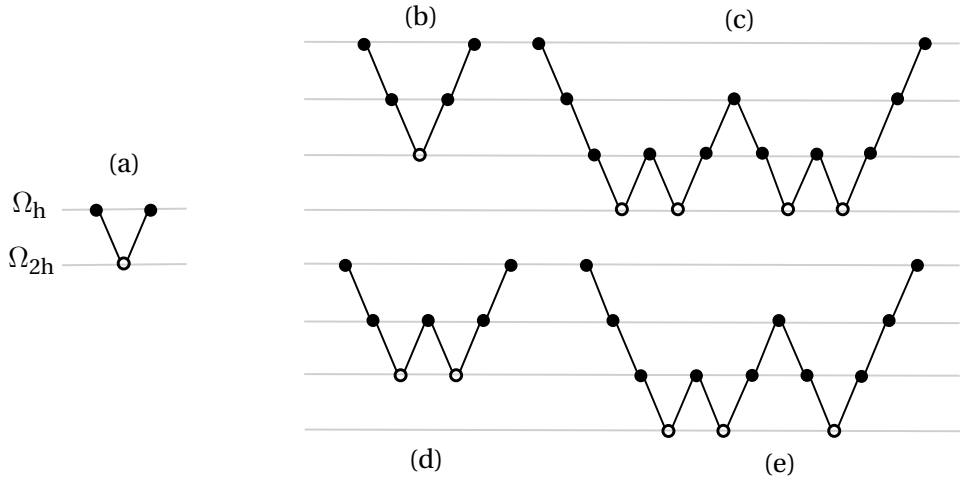


Abbildung 8: (a) Zweigitterzyklus, (b) V-Zyklus bei einem Dreigitterverfahren, (c) W-Zyklus bei einem Viergitterverfahren, (d) W-Zyklus bei einem Dreigitterverfahren, (e) F-Zyklus bei einem Viergitterverfahren. Die ausgefüllten Punkte • stellen Glättungsschritte dar, ∘ kennzeichnet das Lösen des Problems auf dem gröbsten Gitter[2].

- Die Wahl der Startlösung für das Mehrgitterverfahren.

Der letzte Punkt kann für Mehrgitterverfahren sehr leistungsfähig sein, da mit der Grobgitterkorrektur ein natürlicher Zugang für die Erzeugung „guter“ Startlösungen existiert. Diese Herangehensweise führt zum sog. *„Full Multigrid (FMG)“-Verfahren*, welches im Folgenden beschrieben wird.

2.4 FMG-Verfahren

In Kapitel 2.2 wurde die Grobgitterkorrektur nach dem „Top-Down“-Ansatz eingeführt: Beginnend vom feinsten Gitter wird der Fehler schrittweise bis nach unten auf das gröbste Gitter transferiert. Die Idee der Fehlerrepräsentation auf unterschiedlichen Gittern lässt sich allerdings auch mit einem „Bottom-Up“-Ansatz verwenden. Hierzu relaxiert man zunächst das Problem $\mathbf{Av} = \mathbf{f}$ auf dem gröbsten Gitter und verwendet das Ergebnis als Startlösung für das nächstfeinere Gitter. Rekursiv lässt sich somit eine Startlösung für das Problem auf dem feinsten Gitter generieren. Diese Strategie wird *geschachtelte Iteration* genannt.

Führt man die geschachtelte Iteration vom gröbsten bis zum feinsten Gitter durch und verwendet die erhaltene Approximation als Startlösung für ein Mehrgitterverfahren, so enthält die Approximation allerdings viele hoch- und niederfrequente Fehleranteile aufgrund der Interpolation. Wie bei der Grobgitterkorrektur müsste deshalb auf jedem Gitter eine Nachglättung stattfinden. Im einfachsten Fall führt dies allerdings lediglich auf die „rechte Seite“ eines V-Zyklus. Beim FMG-Verfahren wird deshalb iterativ vorgegangen, sodass in jeder Iteration die Tiefe des Mehrgitterverfahrens, d.h. die Anzahl an Gittern, vergrößert wird. Die Fehler, die durch Interpolationen entstehen, werden dadurch genau an der Stelle gedämpft, an der sie entstehen. In Abbildung 9 ist ein FMG-Verfahren mit V-Zyklen schematisch dargestellt.

Im Gegensatz zu Mehrgitterverfahren, bei denen die Korrekturterme $\hat{\mathbf{e}}$ von groben auf feine Gitter transferiert werden, werden bei der geschachtelten Iteration Lösungsapproximationen $\hat{\mathbf{v}}$ und rechte Seiten $\hat{\mathbf{f}}$ transferiert. Erstere können wieder mittels Interpolation transferiert werden. Der Prolongationsoperator muss dabei jedoch eine höhere Ordnung besitzen, um weiter

[2] Die Abbildung stammt aus [14] und wurde leicht abgeändert.

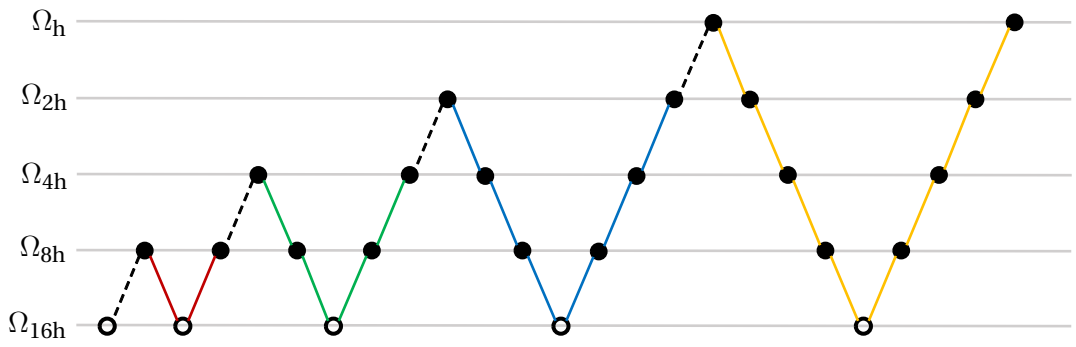


Abbildung 9: Schematische Darstellung eines FMG-Verfahrens mit V-Zyklen. Die gestrichelten Linien stehen für die Prolongationen der geschachtelten Iteration. Die Farben kennzeichnen die V-Zyklen der Mehrgitterverfahren[3].

die h-unabhängige Konvergenz einzuhalten [14]. Formal wird der Operator für die geschachtelte Iteration deshalb mit Π_h^{2h} bezeichnet. Für die rechte Seite kann prinzipiell derselbe Restriktionsoperator (z.B. voller oder halber Gewichtungsoperator) verwendet werden. Häufig ist die Injektion allerdings ausreichend.

In Algorithmus 4 ist das FMG-Verfahren mit einer Maximaltiefe k_{max} aufgeführt. Der Befehl $\mathbf{MG}(k, \mathbf{v}_k^{(0)}, \mathbf{f}_k)$ steht dabei für die Anwendung eines Mehrgitterverfahrens mit Tiefe k, Startlösung $\mathbf{v}_k^{(0)}$ und rechter Seite $\mathbf{f}_k$. Die Iterationsvariable k beschreibt hierbei auch die Gitterhierarchie, sodass mit $k = 0$ das gröbste Gitter Ω_0 und mit $k = k_{max}$ das feinste Gitter $\Omega_{k_{max}}$ bezeichnet wird.

Algorithmus 4 FMG-Verfahren

Löse $\mathbf{A}_0\mathbf{v}_0 = \mathbf{f}_0$ mit beliebigem $\mathbf{v}_0^{(0)} \longrightarrow$ Approximation einer Startlösung $\mathbf{v}_0^{FMG}$
Für $k = 1,2,\ldots,k_{max}$:
  Interpoliere die Startlösung auf das nächstfeinere Gitter Ω_k: $\mathbf{v}_k^{(0)} = \Pi_{k-1}^{k}\mathbf{v}_{k-1}^{FMG}$
  Führe $\mathbf{MG}(k, \mathbf{v}_k^{(0)}, \mathbf{f}_k)$ durch $\longrightarrow$ Startlösung für nächste Iteration $\mathbf{v}_k^{FMG}$

Sobald die maximale Tiefe k_{max} erreicht ist, beinhaltet $\mathbf{v}_{k_{max}}^{FMG}$ eine „optimale“ Startlösung. Im Anschluss können dann noch Mehrgitterverfahren angewendet werden, bis zu einem definiertem Abbruchkriterium. Auf Grund der Qualität der Startlösung werden allerdings nicht mehr viele Iterationen benötigt. Das FMG-Verfahren ist im Unterschied zu Mehrgitterverfahren asymptotisch optimal, d.h. der Aufwand wächst lediglich linear mit der Problemgröße, unabhängig von der gewünschten Genauigkeit.

Sowohl herkömmliche Mehrgitter- als auch FMG-Verfahren bieten auf Grund ihrer hierarchischen Struktur eine natürliche Herangehensweise für die Verwendung unterschiedlicher Datentypen. Nachdem in diesem Kapitel die zugrundeliegende Idee dieser Verfahren dargestellt wurde, widmet sich das nächste Kapitel im Speziellen der Verwendung der gemischter Datentypen.

[3] Die Abbildung stammt aus [14] und wurde leicht abgeändert.

Teil II
Bachelorthesis

3 Gemischt genaue Mehrgitterverfahren

Nachdem im ersten Teil dieser Arbeit die Theorie für Mehrgitterverfahren aufgezeigt wurde geht es im Folgenden um die Implementierung und die Verwendung von gemischt genauen Verfahren. Prinzipiell lassen sich gemischte Genauigkeiten sowohl für herkömmliche Mehrgitter- als auch für FMG-Verfahren verwenden. Im Folgenden werden allerdings ausschließlich gemischt genauen Mehrgitterverfahren betrachtet. Wie sich die Verwendung von mehreren Datentypen auch auf FMG-Verfahren anpassen lässt, wird in Kapitel 6 skizziert. Es folgt zunächst die Zielsetzung und ein Überblick über den zweiten Teil dieser Arbeit.

Zielsetzung Im Zuge dieser Arbeit wird das übergeordnete Ziel verfolgt, mittels der Verwendung von gemischten Genauigkeiten die Ausführungszeit eines Mehrgitterverfahrens zu beschleunigen. Hierbei wird zunächst eine theoretische Schranke des Speed-Ups hergeleitet und im Anschluss die Fehlerentwicklungen und Ausführungszeiten der implementierten Verfahren diskutiert. Zum Vergleich dient jeweils ein herkömmliches Mehrgitterverfahren in doppelter Genauigkeit. Insgesamt müssen die verglichenen Verfahren dieselbe Genauigkeit erreichen. Das Ausgangsproblem wird jeweils in doppelter Genauigkeit gestellt und die Lösung muss nach der Ausführungszeit ebenfalls in doppelter Genauigkeit vorliegen. Konkret bedeutet das, dass für beide Verfahrensklassen dasselbe Abbruchkriterium verwendet wird, welches eine Lösung mit einem Fehler in der Größenordnung von ε_d vermuten lässt.

Überblick Im Folgenden wird zunächst die Testkonfiguration und der Rahmen dieser Arbeit erläutert. Danach erfolgt die theoretische Aufwandsanalyse und die Implementierung der einzelnen Kernel. Aufbauend darauf werden die gemischt genauen Verfahren hergeleitet und der theoretische Speed-Up untersucht. In Kapitel 4 erfolgt dann die Untersuchung und Diskussion der numerischen Experimente. Hierbei wird als Erstes das herkömmliche Mehrgitterverfahren mit unterschiedlichen Datentypen betrachtet. Dies ermöglicht im Anschluss darauf die Untersuchung der gemischt genauen Verfahren. Die diskutierten Ergebnisse werden in Kapitel 5 in Form eines Fazit noch ein mal zusammengefasst. Abschließend wird in Kapitel 6 ausgehend von den Ergebnissen ein Ausblick auf mögliche Erweiterungen und Verbesserungen gegeben.

3.1 Testkonfiguration und Rahmen

Wie bereits in Kapitel 2.3 erwähnt, handelt es sich beim Mehrgitterverfahren nicht um ein einzelnes Verfahren, sondern um eine ganze Klasse an Verfahren. Aus diesem Grund werden zunächst die Mehrgitterparameter festgelegt, die im Zuge dieser Arbeit verwendet werden:

- Als Relaxationsverfahren wird das gedämpfte Jacobi-Verfahren verwendet.
- Pro Level werden drei Vor- und Nachglättungsschritte durchgeführt ($k_{vor} = k_{nach} = 3$).
- Als Vergröberungsstrategie kommt die Standardvergröberung zum Einsatz (d.h. die Gitterweite wird bei Vergröberung halbiert).
- Als Restriktion wird der halbe Gewichtungsoperator verwendet.
- Die Prolongation wird mittels bilinearer Interpolation durchgeführt.
- Sofern nicht näher spezifiziert, werden für die Verfahren V-Zyklen verwendet.
- Als Startlösung dient der Nullvektor (d.h. $\mathbf{v}^{(0)} = \mathbf{0}$).
- Als Löser auf dem gröbsten Gitter kommt das (ungedämpfte) Jacobi-Verfahren zum Einsatz.

Häufig werden bei Mehrgitterverfahren direkte Löser für das Lösen auf dem gröbsten Gitter verwendet. Für die Parallelisierung auf Grafikkarten ergibt sich jedoch auf Grund der Blockgröße eine untere Schranke für die Anwendung eines Kernels, die noch sinnvoll ausgeführt werden kann. Deshalb wird im Zuge dieser Arbeit nicht bis ganz in die Tiefe vergröbert, sondern bereits ab einer moderaten Problemgröße das ungedämpfte Jacobi-Verfahren angewendet. Hierbei werden $k_{\text{lösen}} = 151$ Iterationen durchgeführt. Die Anzahl an Iterationen und die Tiefe von Mehrgitterverfahren beeinflussen die Fehlerentwicklung und die Ausführungszeit ungemein. Die oben genannte Zahl hat sich in den numerischen Experimenten als ein guter Kompromiss zwischen Mehrgitter-Zyklen und Anzahl an Jacobi-Schritten herausgestellt. Eine kurze Untersuchung hierzu befindet sich in Anhang A. Da der Fokus dieser Arbeit auf der Verwendung von gemischten Genauigkeiten liegt, sind die obigen Parameter fest gewählt und werden sowohl für die experimentellen gemischt genauen Verfahren, als auch für die zur Referenz implementierten Mehrgitterverfahren verwendet. An dieser Stelle sei jedoch ausdrücklich darauf hingewiesen, dass auch die obigen Parameter in Verbindung mit gemischten Genauigkeiten näher untersucht werden können, um so die Ausführungszeiten der Verfahren zu verbessern.

Testumgebung

Alle in dieser Arbeit aufgeführten Verfahren wurden auf einer NVIDIA Tesla V100 Beschleunigerkarte durchgeführt. Die auf der Volta-Architektur basierende Karte besitzt einen 32 GByte großen HBM2 Speicher, welcher mit einem 4096-Bit Bus zu einer theoretischen Speicherbandbreite von 900 GByte/s führt [16]. Die Taktfrequenz der GPU ist dynamisch gesteuert und beträgt laut AnandTech knapp 1,46 GHz im Boost-Betrieb [17]. Bei Verwendung von einfacher Genauigkeit erreicht die V100 einen Rechendurchsatz von 15 TFLOP/s. Wird in doppelter Genauigkeit gerechnet, halbiert sich der Rechendurchsatz entsprechend. Bei Verwendung des Datentyps Half erhält man den doppelten Rechendurchsatz, jedoch nur, wenn die Operationen mit dem vektorisierten Datentyp *half2* durchgeführt werden. Mit 80 Streamprozessoren und jeweils 64 CUDA-Kernen stellt die Tesla V100 insgesamt 5.120 CUDA-Kerne bereit. Neben den traditionellen Recheneinheiten sind pro Prozessor noch zusätzliche 8 Tensor Cores verbaut. Bei der reinen Verwendung dieser Einheiten lässt sich laut AnandTech ein theoretischer Rechendurchsatz von 120 TFLOP/s erzielen. Dies entspricht dem Vierfachen des Rechendurchsatzes bei der Verwendung von *half2* und den herkömmlichen CUDA-Kernen. Neben den Recheneinheiten besitzt die Beschleunigerkarte ebenfalls einen 6 MByte großen L2-Zwischenspeicher. Jeder Streamprozessor verfügt zusätzlich über einen 128 KByte großen geteilten Speicher, welcher sich jedoch auch als L1- und Textur-Zwischenspeicher konfigurieren lässt [18].

Als Host wurde ein System basierend auf IBMs Power9 Architektur verwendet. Der Server besitzt zwei Sockets mit je 20 Kernen, wobei ein Kern parallel vier Threads bearbeiten kann. Dies macht in der Summe 160 Threads aus, welche parallel verarbeitet werden können. Die Taktfrequenz reicht von 2,3 bis 3,8 GHz und die beiden NUMA-Knoten sind an insgesamt 384 GByte DDR4 Arbeitsspeicher angebunden.

Die im Zuge dieser Arbeit implementierten Verfahren verwenden keine vektorisierten Datentypen. Das bedeutet, dass selbst bei der Verwendung von halber Genauigkeit lediglich derselbe Rechendurchsatz erzielt werden kann, wie bei der entsprechende Single-Implementierung. Der gemessene Speed-Up durch die Verwendung von halber Genauigkeit resultiert deshalb alleine auf Grund des geringeren Speicherbedarfs. Wie sich in Kapitel 4.2 zeigt, liegt der Rechendurchsatz der implementierten Kernel deutlich unterhalb dem theoretischen Maximum. Dadurch wird die Ausführungszeit maßgeblich durch den Speicherzugriff dominiert. Es ist deshalb anzunehmen, dass der Verzicht auf vektorisierte Datentypen keine allzu großen Einschränkungen mit sich bringt. Im Gegensatz zu den Ausführungszeiten sind die Fehlerentwicklungen per se unabhängig davon, ob vektorisierte Datentypen verwendet werden oder nicht.

Messtechnik Die Messungen von Ausführungszeiten wurden mit einem CUDA-basierten Timer, dessen Implementierung aus dem Doktorandenkurs „GPU Programming with CUDA“ [19] stammt, durchgeführt. Hierbei wird beim Start der Messung ein Ereignis im Bearbeitungsstrom der Grafikkarte registriert. Nachdem die zu messenden Berechnungen asynchron in den Bearbeitungsstrom gegeben werden, wird ein weiteres Ereignis registriert. Nach Beendigung der Berechnungen kann die zeitliche Differenz der beiden Ereignisse im Mikrosekunden-Bereich ermittelt werden.

Um Einflüsse der Zwischenspeicher auszuschließen, wurde vor den Messungen der Kernel die Initialisierung eines 2^{21} großen Arrays in doppelter Genauigkeit durchgeführt. Die anfallenden $2^{21} \cdot 8$ Byte $\approx 16,78$ MByte reichen aus, um den 6 MByte großen L2-Zwischenspeicher, sowie die einzelnen L1-/Textur-Zwischenspeicher der Beschleunigerkarte zu bereinigen.

Problemgrößen Da die Kondition des Problems stark von der Problemgröße abhängt und Auswirkung auf die Fehlerentwicklung hat, werden im Zuge dieser Arbeit unterschiedliche Problemgrößen miteinander verglichen. Für eine Gitterweite von $h = 1/(n+1)$ ergibt sich eine Gesamtgröße von $(n+2)^2$, wobei $n \in \mathbb{N}$ für die Anzahl an inneren Gitterpunkten steht. Auf Grund der Standardvergröberung kann $h = 2^{-k}$ für ein $k \in \mathbb{N}$ angenommen werden. Im Folgenden wird das $k \in \mathbb{N}$, welches zum feinsten Gitter gehört, als Level bezeichnet. Eine Problemgröße kann damit durch das zugehörige Level charakterisiert werden. Für Level 12 erhält man somit eine Gitterweite von $h = 2^{-12}$ und eine Gesamtgröße von $(2^{12}+2)^2 = 16.793.604$ Gitterknoten. In Tabelle 2 sind die betrachteten Level mit zugehöriger Gitterweite und Problemgröße aufgelistet. Im Zuge dieser Arbeit wird bei jedem Verfahren stets bis zum 5. Level vergröbert.

Level	Gitterweite h	Problemgröße $(n+2)^2$
5	$2^{-5} = 1/32$	1.156
6	$2^{-6} = 1/64$	4.356
7	$2^{-7} = 1/128$	16.900
8	$2^{-8} = 1/256$	66.564
9	$2^{-9} = 1/512$	264.196
10	$2^{-10} = 1/1024$	1.052.676
11	$2^{-11} = 1/2048$	4.202.500
12	$2^{-12} = 1/4096$	16.793.604

Tabelle 2: Auflistung der Gitterweite und Problemgröße in Abhängigkeit des Levels.

3.2 Theoretische Aufwandsanalyse und Implementierung der Kernel

Die Parallelisierung beginnt mit der Wahl und Konzeption der Kernel. In Algorithmus 5 ist hierzu die Durchführung eines Schritts des Mehrgitterverfahrens aufgeführt. Hierbei wird ein Jacobi-Verfahren (JACOBI), eine Residuumsberechnung (DEFECT), eine Restriktion (RESTRICT), eine Interpolation (PROLONGATE) und eine Vektoraddition (UPDATE) durchgeführt. Um Abbruchkriterien zu verwenden, muss ebenfalls die Berechnung einer Norm (NORM) durchgeführt werden. Die rein skalaren Berechnungen werden bei der Parallelisierung vernachlässigt, da der Mehraufwand durch Kopiervorgänge und Start eines Kernels bereits aufwändiger sind, als die Berechnung selbst, siehe hierzu auch Kapitel 5.8.2 der Dissertation von Dominik Göddeke [6]. Auf Grund der Verwendung unterschiedlicher Datentypen ist ebenfalls eine Konvertierung notwendig. Diese kann in den einzelnen Kerneln direkt erfolgen. Die beschriebene Verwendung des Textur-und geteilten Speichers orientiert sich an Kapitel 5.2.4 aus [6].

Algorithmus 5 Mehrgitterverfahren

Berechnung von $\mathbf{v}_k^{(m+1)} = \mathbf{MG}(k, \mathbf{v}_k^{(m)}, \mathbf{f}_k)$:
- Vorglättung:
  - Berechne $\mathbf{v}_k^{(m+\frac{1}{3})}$ durch k_{vor}-malige Anwendung des gedämpften Jacobi-Verfahrens: $\mathbf{v}_k^{(m+\frac{1}{3})} = \mathbf{JACOBI}^{k_{vor}}(\mathbf{v}_k^{(m)}, \mathbf{f}_k)$
- Grobgitterkorrektur:
  - Berechne das Residuum: $\mathbf{r}_k = \mathbf{DEFECT}(\mathbf{v}_k^{(m+\frac{1}{3})}, \mathbf{f}_k)$
  - Falls $k = k_{max}$:
    - Berechne die Norm des Residuums: $||\mathbf{r}_k|| = \mathbf{NORM}(\mathbf{r}_k)$
    - Führe eine Konvergenzkontrolle durch
  - Restringiere das Residuum: $\hat{\mathbf{r}}_{k-1} = \mathbf{RESTRICT}(\mathbf{r}_k)$
  - Berechne eine Approximation $\hat{\mathbf{e}}_{k-1}$ der Residuumsgleichung durch:
    - Falls $k = k_{min}$:
      - Approximiere mit dem ungedämpften Jacobi-Verfahren: $\hat{\mathbf{e}}_{k-1} = \mathbf{JACOBI}^{k_{lösen}}(\mathbf{0}, \hat{\mathbf{r}}_{k-1})$
    - Falls $k > k_{min}$:
      - Approximiere durch die Anwendung eines Mehrgitterverfahrens: $\hat{\mathbf{e}}_{k-1} = \mathbf{MG}(k-1, \mathbf{0}, \hat{\mathbf{r}}_{k-1})$
  - Prolongiere die Korrektur: $\hat{\mathbf{e}}_k = \mathbf{PROLONGATE}(\hat{\mathbf{e}}_{k-1})$
  - Korrigiere die Approximation: $\mathbf{v}_k^{(m+\frac{2}{3})} = \mathbf{UPDATE}(\mathbf{v}_k^{(m+\frac{1}{3})}, \hat{\mathbf{e}}_k)$
- Nachglättung:
  - Berechne $\mathbf{v}_k^{(m+1)}$ durch k_{nach}-malige Anwendung des gedämpften Jacobi-Verfahrens: $\mathbf{v}_k^{(m+1)} = \mathbf{JACOBI}^{k_{nach}}(\mathbf{v}_k^{(m+\frac{2}{3})}, \mathbf{f}_k)$

Roofline-Modell

Um Aussagen über den theoretischen Speed-Up der gemischt genauen Verfahren treffen zu können, wird im Folgenden auch auf den Rechenaufwand und den Speicherzugriff der einzelnen Kernel eingegangen. Für eine solche Analyse wird häufig das sog. *Roofline-Modell* herangezogen [20]. Hierbei wird zunächst die Rechenintensität I eines Verfahrens bestimmt. Sie berechnet sich als Quotient zwischen der Anzahl an Gleitkommaoperationen und dem damit verbundenen Speicherzugriff und besitzt die Einheit FLOP pro Byte. Zusammen mit dem theoretischen Rechendurchsatz P_{max} und der theoretischen Speicherbandbreite b_s lässt sich die Leistung P eines Verfahrens dadurch wie folgt berechnen:

$$P = \min(I \cdot b_s, P_{max}).$$

Eine wesentliche Annahme des Roofline-Modells ist, dass die einzelnen Recheneinheiten optimal verwendet und die gesamte Speicherbandbreite ausgenützt wird. Das bedeutet zum einen, dass der theoretische Rechendurchsatz erreicht wird, sofern die Daten schnell genug geladen werden können, und zum andern, dass sämtliche Daten stets mit der vollen Speicherbandbreite geladen werden. Möchte man das Roofline-Modell auf gemischt genaue Verfahren anwenden, so zeigt sich eine Reihe von Problemen: Zum einen kann kein eindeutiger theoretischer Rechendurchsatz angegeben werden, da für die Hardware lediglich der theoretische Rechendurchsatz pro Datentyp spezifiziert ist, nicht aber, wie sich der Rechendurchsatz unter der Verwendung von gemischten Genauigkeiten verhält. Zum anderen stellt sich auch die Bestimmung der Rechenintensität eines Verfahrens mit unterschiedlichen Datentypen als schwierig heraus, da zwischen Double- Single- und Half-Operationen unterschieden werden muss und auch Konvertierungen miteinbezogen werden sollten. Aus diesem Grund wird im Folgenden

nicht mit dem Roofline-Modell, sondern mit einem vereinfachten Leistungsmodell gearbeitet, welches eine Aussage über den theoretischen Speed-Up erlaubt.

Vereinfachtes Leistungsmodell

Der Speed-Up berechnet sich allgemein als Quotient zwischen der Ausführungszeit des herkömmlichen Mehrgitterverfahrens und den gemischt genauen Verfahren. Wird mit T_{MG} die Ausführungszeit des Mehrgitterverfahrens und mit T_{GGMG} die Ausführungszeit eines gemischt genauen Verfahrens bezeichnet, dann gilt

$$\mathrm{Speed-Up} = \frac{\mathrm{T_{MG}}}{\mathrm{T_{GGMG}}}.$$

Neben den Annahmen des Roofline-Modells wird beim vereinfachten Leistungsmodell zusätzlich angenommen, dass die Berechnung mit unterschiedlichen Datentypen seriell stattfindet. Ebenfalls werden die Konvertierungen seriell zu den Rechenoperationen durchgeführt. Der Speicherzugriff erfolgt hingegen parallel zu den Gleitkommaoperationen und „überlappt vollständig“. Das bedeutet, dass das Laden und Schreiben der Daten zeitgleich zu den Rechenoperationen stattfindet. Die Annahmen des vereinfachten Leistungsmodells sind in Abbildung 10 schematisch dargestellt.

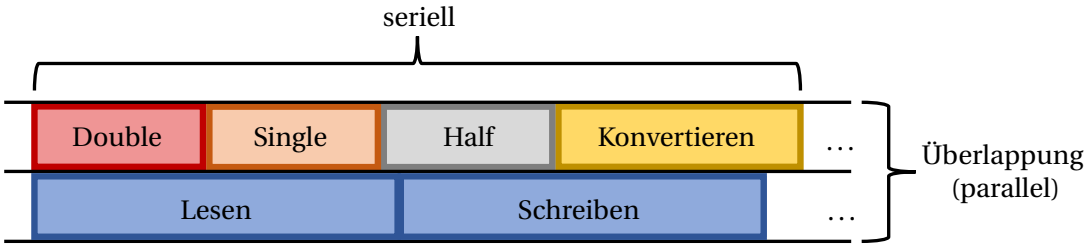


Abbildung 10: Visualisierung der Annahmen des gemischt genauen Roofline-Modells. Die Rechen- und Konvertierungsoperationen werden seriell durchgeführt. Lese- und Schreibvorgänge überlappen vollständig mit den Gleitkommaoperationen.

Die Ausführungszeit eines Verfahrens lässt sich aus den einzelnen Ausführungszeiten der Kernel zusammensetzen, d.h.

$$\mathrm{T} = \sum_{\mathrm{kernel}} \mathrm{t_{kernel}},$$

wobei t_{kernel} für die Ausführungszeit eines Kernels steht und über alle ausgeführten Kernel summiert wird. Beim vereinfachten Leistungsmodell berechnet sich die theoretische Ausführungszeit eines Kernels als das Maximum zwischen der Speicher- und Rechenzeit. Steht t_{mem} für die Speicher- und t_{calc} für die Rechenzeit eines Kernels, so gilt

$$\mathrm{t_{kernel}} = \max\left(\mathrm{t_{mem}}, \mathrm{t_{calc}}\right). \tag{8}$$

Die jeweiligen Zeiten lassen sich mittels der Hardwarespezifikation sowie des Rechenbedarfs bzw. des Speicherzugriffs berechnen. Im Folgenden steht ${}^{dt}P_{max}$ für den theoretischen Rechendurchsatz, ${}^{dt}CALC_{ker}$ für den Rechenbedarf und ${}^{dt}MEM_{ker}$ für den Speicherzugriff des Kernels *ker* mit Datentyp *dt*. Die zugehörige Einheit beim Rechenbedarf wird mit ${}^{dt}FLOP$ bezeichnet, sodass zwischen den einzelnen Datentypen unterschieden werden kann. Im Folgenden wird mit der Abkürzung *d* für Double, *s* für Single und *h* für Half gearbeitet. Da bei Konvertierungen zwei Datentypen verwendet werden, setzt sich *dt* hierbei aus den beiden beteiligten Datentypen zusammen. Somit steht *ds* für Double-Single und *sh* für Single-Half. Die Zeit für

das Laden und Speichern der Daten für einen Kernel *ker* mit Datentyp *dt* berechnet sich damit zu

$$^{dt}t_{mem,ker} = \frac{^{dt}MEM_{ker}}{b_s}.$$

Für die Rechenzeit gilt analog

$$^{dt}t_{calc,ker} = \frac{^{dt}CALC_{ker}}{^{dt}P_{max}}.$$

Hierbei ist anzumerken, dass der theoretische Speed-Up auch in diesem Modell von der verwendeten Hardware abhängt. In Tabelle 3 ist der Rechendurchsatz für ausgewählte Rechenoperationen pro GPU-Zyklus und Streamprozessor aufgelistet. Die Zahlen stammen aus [21] und beziehen sich auf die Compute Capability 7.0, welche bei der Tesla V100 zum Einsatz kommt. Da der Aufwand für eine Konvertierung unabhängig von der Richtung ist, wird sowohl die Konvertierung von Double nach Single, als auch die von Single nach Double mit ^{ds}CONV bezeichnet. Das theoretische Maximum für die Konvertierung wird von NVIDIA nicht näher

Datentyp dt	Rechenoperation	Durchsatz	Theo. Max. $^{dt}P_{max}$
Half	^{h}ADD, ^{h}MUL, ^{h}FMA	128	30 TFLOP/s = 1/4 $^{d}P_{max}$
Single	^{s}ADD, ^{s}MUL, ^{s}FMA	64	15 TFLOP/s = 1/2 $^{d}P_{max}$
Double	^{d}ADD, ^{d}MUL, ^{d}FMA	32	7,5 TFLOP/s = 1 $^{d}P_{max}$
Single-Half	^{sh}CONV	16	3,75 TFLOP/s = 2 $^{d}P_{max}$
Double-Single	^{ds}CONV	16	3,75 TFLOP/s = 2 $^{d}P_{max}$

Tabelle 3: Auflistung des Rechendurchsatzes für ausgewählte Rechenoperationen pro Zyklus und Zeit. Der Durchsatz beschreibt die Anzahl an Ergebnissen pro GPU-Zyklus und Streamprozessor.

spezifiziert. Da jede Konvertierung nur die Hälfte des Durchsatzes einer Gleitkommaoperation in doppelter Genauigkeit besitzt, wird im Folgenden auch die Hälfte des theoretischen Maximums dieser Operationen verwendet. An dieser Stelle sei ebenfalls darauf hingewiesen, dass in dem hier verwendeten Modell sämtliche Effekte durch die Zwischenspeicher den Textur- und geteilten Speicher vernachlässigt werden. Außerdem wird für die Konvertierung angenommen, dass keine zusätzlichen Speicherzugriffe durchgeführt werden müssen. Für die Berechnung der Ausführungszeit eines Verfahrens müssen die Effekte wie das Starten eines Kernels oder das Synchronisieren mit der CPU berücksichtigt werden. Da diese Effekte jedoch sowohl beim herkömmlichen Mehrgitterverfahren wie auch bei den gemischt genauen Verfahren in etwa der selben Größenordnung liegen, werden sie bei der Berechnung der Ausführungszeit vernachlässigt.

Nachdem das zugrundeliegende Leistungsmodell erläutert wurde, wird im Folgenden auf die Implementierung und Parallelisierung eingegangen. Dabei wird jeder Kernel separat betrachtet und der zugehörige Rechenbedarf und Speicherzugriff hergeleitet.

Parallelisierung

Für alle implementierten Kernel werden eindimensionale CUDA-Blöcke verwendet, was die Verwendung des geteilten Speichers vereinfacht. Das CUDA-Gitter wird zweidimensional gewählt, sodass eine einfache Indexarithmetik für das zugrundeliegende 2D-Gitter verwendet werden kann. Die Zusammensetzung aus Blöcken und dem Gitter ist in Abbildung 11 schematisch dargestellt. Da auf jedem Level je vier Vektoren ($\mathbf{v}, \mathbf{f}, \mathbf{r}, \hat{\mathbf{e}}$) verwendet werden, besitzen

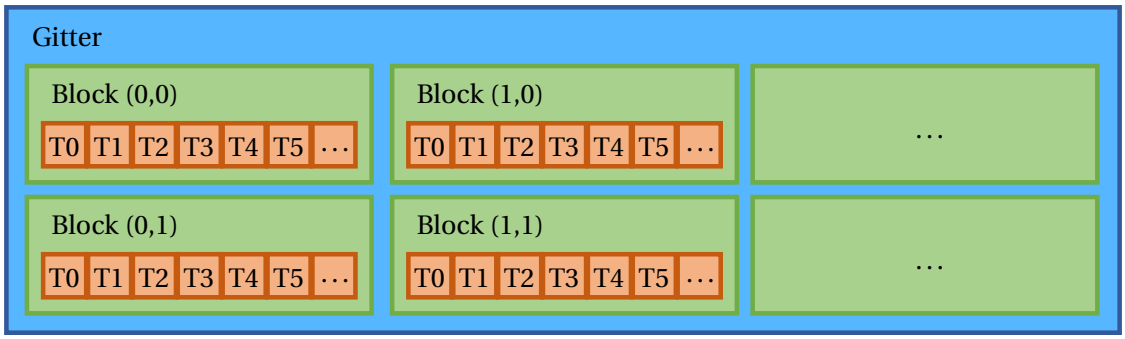


Abbildung 11: Schematische Darstellung der CUDA-Gitter und Block-Aufteilung. Es werden zweidimensionale Gitter mit eindimensionalen Blöcken verwendet.

Mehrgitterverfahren einen verhältnismäßig hohen Speicherbedarf. Der Speicher, der pro Level benötigt wird, wird bereits im Vorfeld innerhalb einer Initialisierungsphase allokiert. Damit besitzt das Verfahren zwar über die gesamte Laufzeit hinweg die maximale Speicherauslastung, Zeitmessungen beinhalten somit allerdings lediglich die für das Verfahren notwendigen Berechnungen.

Vereinfachung bei Dirichlet-Randbedingungen

Im Gegensatz zur formalen Beschreibung des Problems aus Kapitel 1.2 beinhaltet die Lösungsapproximation in der Implementierung auch die Randwerte. Dies hat den Vorteil, dass die Anwendung eines Stencils einfach implementiert werden kann und für die Residuumsberechnung und Interpolation eine Vereinfachung getroffen werden kann. Der Nachteil ist jedoch, dass mehr Speicherplatz benötigt wird. Da die Werte an den Rändern a priori bekannt sind, ist es möglich, diese direkt für die Startlösung $\mathbf{v}^{(0)}$ zu setzen. Die Systemmatrix vergrößert sich dadurch und erhält zusätzliche Einträge für die Ränder. Auf der Hauptdiagonalen werden Einser hinzugefügt, sodass die Anwendung eines Stencils die bekannten Werte an den Rändern nicht verändert. In Abbildung 12 ist die erweiterte Systemmatrix schematisch dargestellt. Die Wer-

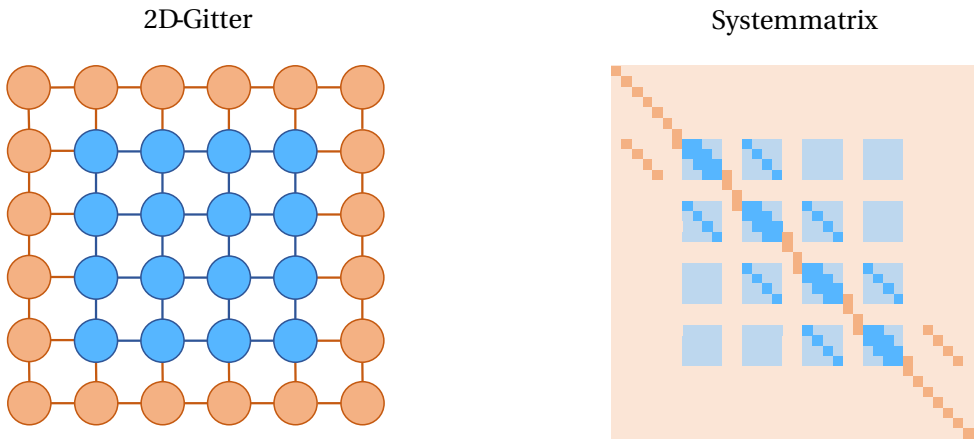


Abbildung 12: Links: 2D-Gitter mit orange eingefärbten Randpunkten. Rechts: Erweiterung der Systemmatrix auf Randpunkte. Die Erweiterung ist in Dunkelorange farblich hervorgehoben.

te auf den Rändern von $\mathbf{v}$ und damit von $\mathbf{Av}$ stimmen dadurch mit denen der rechten Seite $\mathbf{f}$ überein, d.h.

$$\mathbf{v}_{i,j} = (\mathbf{Av})_{i,j} = \mathbf{f}_{i,j} \qquad \text{für} \quad i \in \{0, n+1\} \quad \text{oder} \quad j \in \{0, n+1\}.$$

Damit berechnen sich die Werte des Residuums auf dem Rand zu

$$\mathbf{r}_{i,j} = \mathbf{f}_{i,j} - (\mathbf{A}\mathbf{v})_{i,j} = 0 \qquad \text{für} \quad i \in \{0, n+1\} \quad \text{oder} \quad j \in \{0, n+1\}.$$

Die Erweiterung hat ebenfalls Auswirkungen auf den Fehler **e**, welcher mittels der Residuumsgleichung $\mathbf{A}\mathbf{e} = \mathbf{r}$ bestimmt wird. Da die Systemmatrix die Werte an den Rändern unverändert lässt und das Residuum auf dem Rand verschwindet, gilt

$$(\mathbf{e})_{i,j} = (\mathbf{A}\mathbf{e})_{i,j} = (\mathbf{r})_{i,j} = 0 \qquad \text{für} \quad i \in \{0, n+1\} \quad \text{oder} \quad j \in \{0, n+1\}.$$

D.h. sowohl das Residuum **r** als auch der Fehler **e** verschwinden auf dem Rand. Diese Erkenntnis lässt sich für die Implementierung der Residuumsberechnung und der Interpolation ausnutzen. Als Nächstes wird deshalb die Implementierung der einzelnen Kernel näher betrachtet.

JACOBI

Unter Verwendung der obigen Systemmatrix lautet die zweidimensionale, elementweise Berechnungsvorschrift des gedämpften Jacobiverfahrens

$$\mathbf{v}_{i,j}^{(m+\frac{1}{2})} = \frac{1}{4}\left(h^2 \cdot \mathbf{f}_{i,j} + \mathbf{v}_{i-1,j}^{(m)} + \mathbf{v}_{i,j-1}^{(m)} + \mathbf{v}_{i+1,j}^{(m)} + \mathbf{v}_{i,j+1}^{(m)}\right)$$
$$\mathbf{v}_{i,j}^{(m+1)} = (1-\omega)\,\mathbf{v}_{i,j}^{(m)} + \omega\,\mathbf{v}_{i,j}^{(m+\frac{1}{2})},$$

für $i, j = 1, 2, \ldots, n$. Die Werte an den Rändern können einfach aus der Startlösung übernommen werden, d.h.

$$\mathbf{v}_{i,j}^{(m+1)} = \mathbf{v}_{i,j}^{(m)} \qquad \text{für} \quad i \in \{0, n+1\} \quad \text{oder} \quad j \in \{0, n+1\}.$$

Implementierung Da benachbarte Elemente von $\mathbf{v}^{(m)}$ nicht linear im Speicher abliegen, werden diese zunächst in den geteilten Speicher geladen. Hierbei wird der Datenstrom durch den Textur-Zwischenspeicher geroutet, um die zweidimensionale räumliche Struktur auszunutzen. Im Gegensatz zum Textur-Zwischenspeicher fungiert der geteilte Speicher in dieser Implementierung als softwaregesteuerter Zwischenspeicher. Der Verlauf der Datenströme ist in Abbildung 13 schematisch dargestellt. Die Werte der rechten Seite **f** werden explizit nicht in den geteilten Speicher oder durch den Textur-Zwischenspeicher geladen, da pro Thread lediglich ein Wert ausgelesen wird. Im zweiten Schritt erfolgt dann die Berechnung der neuen Lösung $\mathbf{v}^{(m+1)}$, wobei die Werte der Startlösung $\mathbf{v}^{(m)}$ aus dem geteilten Speicher geladen werden.

An dieser Stelle ist anzumerken, dass in CUDA derzeit keine Texturen in doppelter oder halber Genauigkeit erstellt werden können. Ganzzahlige Datentypen werden allerdings unterstützt. Durch die Verwendung von *__hiloint2double()* und *__ushort_as_half()* können somit $2 \cdot 4$ Byte große Ganzzahlen (*uint2*) als Double und $1 \cdot 2$ Byte große Ganzzahlen (*ushort*) als Half interpretiert werden.

Aufwandsanalyse Pro Lösungswert werden zehn elementare Rechenoperationen durchgeführt. Diese lassen sich in zwei Multiplikationen, vier Additionen und zwei FMA-Operationen unterteilen. Der Faktor h^2 wird dabei als Parameter dem Kernel übergeben. Der Aufwand des JACOBI-Kernels beläuft sich damit auf insgesamt

$$\begin{aligned} {}^{d}\mathrm{CALC}_{jac}(n) &= (2 \cdot {}^{d}\mathrm{MUL} + 4 \cdot {}^{d}\mathrm{ADD} + 2 \cdot {}^{d}\mathrm{FMA}) \cdot n^2 \\ &= 8 \cdot n^2\ {}^{d}\mathrm{FLOP}. \end{aligned}$$

Für die Berechnung werden alle Elemente der Startlösung, aber nur die inneren Punkte der rechten Seite benötigt. Zusätzlich werden alle Gitterpunkte der neuen Lösung zurück in den

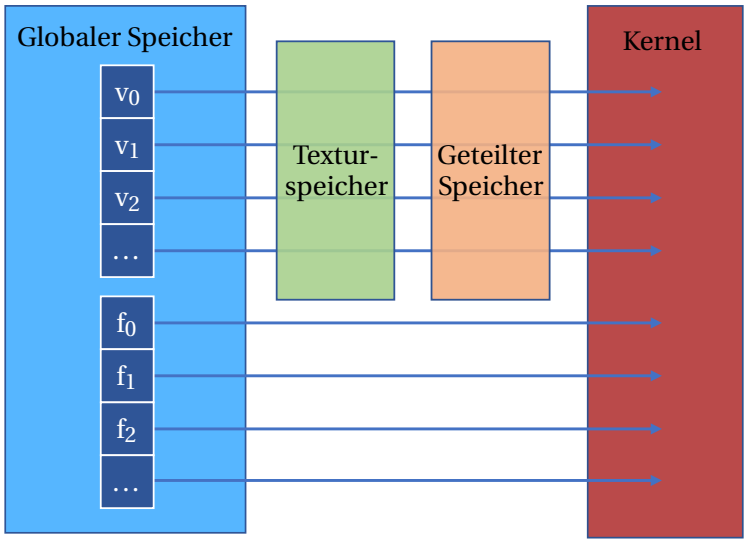


Abbildung 13: Datenstrom beim JACOBI-Kernel. Der Lösungsvektor **v** wird aus dem globalen Speicher über den Textur-Speicher in den geteilten Speicher geladen. Die rechte Seite **f** wird direkt aus dem globalen Speicher geladen.

Speicher geschrieben. Der JACOBI-Kernel in doppelter Genauigkeit besitzt somit einen Speicherzugriff von

$$\begin{aligned} {}^{\mathrm{d}}\mathrm{MEM}_{\mathrm{jac}}(n) &= \left((n+2)^2 + n^2 + (n+2)^2\right) \cdot 8\ \mathrm{Byte} \\ &= 24 \cdot n^2 + 64 \cdot n + 64\ \mathrm{Byte} \end{aligned}$$

Für die Verwendung von einfacher oder halber Genauigkeit wird entsprechend mit vier bzw. zwei Byte multipliziert.

DEFECT

Die Berechnungsvorschrift für das Residuum lautet

$$\mathbf{r}_{i,j}^{(m)} = \mathbf{f}_{i,j} + \frac{1}{h^2}\left(\mathbf{v}_{i,j-1}^{(m)} + \mathbf{v}_{i-1,j}^{(m)} - 4 \cdot \mathbf{v}_{i,j}^{(m)} + \mathbf{v}_{i+1,j}^{(m)} + \mathbf{v}_{i,j+1}^{(m)}\right),$$

für $i,j = 1,2,\ldots,n$. Die Werte an den Rändern müssen auf Grund der Vereinfachung nicht berücksichtigt werden, sofern die Vektoren zuvor mit Null initialisiert werden.

Implementierung Die Implementierung erfolgt analog zum JACOBI-Kernel: Auch hier wird der Lösungsvektor über den Textur-Zwischenspeicher in den geteilten Speicher geladen. Im Gegensatz zum JACOBI-Kernel sollte der DEFECT-Kernel allerdings mit zwei unterschiedlichen Datentypen umgehen können. Konkret bedeutet das, dass ausgehend von einer Approximation in Single das zugehörige Residuum in doppelter Genauigkeit berechnet werden soll. Dies ist notwendig, wenn das Residuum als Abbruchkriterium verwendet wird. Dieser Sachverhalt ist ein Resultat der späteren Untersuchung von gemischt genauen Mehrgitterverfahren, auf welchen in Kapitel 4.4 näher eingegangen wird. In diesem Falle werden die einzelnen Komponenten der Approximation $\mathbf{v}^{(m)}$ zunächst in die höhere Genauigkeit konvertiert und anschließend die Berechnung durchgeführt.

Aufwandsanalyse Durch die Anwendung eines DEFECT-Kernels werden je sieben elementare Rechenoperationen pro zu berechnendem Wert durchgeführt. Diese lassen sich unter Hin-

zunahme zweier FMA-Operationen auf fünf reduzieren, weshalb der DEFECT-Kernel einen Rechenbedarf von

$$\begin{aligned} {}^{\mathrm{d}}\mathrm{CALC}_{\mathrm{def}}(\mathrm{n}) &= (3 \cdot {}^{\mathrm{d}}\mathrm{ADD} + 2 \cdot {}^{\mathrm{d}}\mathrm{FMA}) \cdot \mathrm{n}^2 \\ &= 5 \cdot \mathrm{n}^2 \, {}^{\mathrm{d}}\mathrm{FLOP} \end{aligned}$$

für die Verwendung von doppelter Genauigkeit besitzt. Von der Approximation $\mathbf{v}^{(m)}$ müssen alle Werte bis auf die vier Ecken geladen werden, weshalb der DEFECT-Kernel insgesamt

$$\begin{aligned} {}^{\mathrm{d}}\mathrm{MEM}_{\mathrm{def}}(\mathrm{n}) &= \left((\mathrm{n}+2)^2 - 4 + 2 \cdot \mathrm{n}^2\right) \cdot 8 \text{ Byte} \\ &= 24 \cdot \mathrm{n}^2 + 32 \cdot \mathrm{n} \text{ Byte} \end{aligned}$$

an Daten verarbeitet. Der Summand $2 \cdot n^2$ steht für das Laden der rechten Seite und das Schreiben der berechneten Lösung. Wird das Residuum ausgehend von einfacher Genauigkeit berechnet, so beläuft sich der Rechenbedarf auf

$$\begin{aligned} {}^{\mathrm{ds}}\mathrm{CALC}_{\mathrm{def}}(\mathrm{n}) &= (3 \cdot {}^{\mathrm{d}}\mathrm{ADD} + 2 \cdot {}^{\mathrm{d}}\mathrm{FMA} + 5 \cdot {}^{\mathrm{ds}}\mathrm{CONV}) \cdot \mathrm{n}^2 \\ &= 5 \cdot \mathrm{n}^2 \, {}^{\mathrm{d}}\mathrm{FLOP} + 5 \cdot \mathrm{n}^2 \, {}^{\mathrm{ds}}\mathrm{FLOP}. \end{aligned}$$

Die Einheit ${}^{ds}FLOP$ kennzeichnet dabei den Aufwand einer Konvertierung von einfacher in doppelte Genauigkeit. Der Speicherzugriff ändert sich dadurch ebenfalls, da sich die Speichermenge der Approximation $\mathbf{v}^{(m)}$ halbiert hat. Damit verarbeitet der gemischt genaue DEFECT-Kernel insgesamt

$$\begin{aligned} {}^{\mathrm{ds}}\mathrm{MEM}_{\mathrm{def}}(\mathrm{n}) &= \left((\mathrm{n}+2)^2 - 4\right) \cdot 4 \text{ Byte} + \left(2 \cdot \mathrm{n}^2\right) \cdot 8 \text{ Byte} \\ &= 20 \cdot \mathrm{n}^2 + 16 \cdot \mathrm{n} \text{ Byte} \end{aligned}$$

an Daten. Analog kann die obige Rechnung für die Datentypen Single und Half durchgeführt werden.

RESTRICT

Wie bei der Residuumsberechnung und dem Jacobiverfahren handelt es sich bei der Restriktion um die Anwendung eines Stencils. Unter Verwendung des halben Gewichtungsoperators aus Kapitel 2.2 und der Standardvergröberung erhält man somit als elementweise Berechnungsvorschrift

$$\hat{\mathbf{r}}_{\mathrm{i,j}}^{(\mathrm{m})} = (\mathbf{I}_{\mathrm{h}}^{2\mathrm{h}} \mathbf{r}^{(\mathrm{m})})_{\mathrm{i,j}} = \frac{1}{8}\left(\mathbf{r}_{2\mathrm{i},2\mathrm{j}-1}^{(\mathrm{m})} + \mathbf{r}_{2\mathrm{i}-1,2\mathrm{j}}^{(\mathrm{m})} + 4 \cdot \mathbf{r}_{2\mathrm{i},2\mathrm{j}}^{(\mathrm{m})} + \mathbf{r}_{2\mathrm{i}+1,2\mathrm{j}}^{(\mathrm{m})} + \mathbf{r}_{2\mathrm{i},2\mathrm{j}+1}^{(\mathrm{m})}\right),$$

für $\mathrm{i,j} = 1, 2, \ldots, \mathrm{n}_{\mathrm{grob}}$. Hierbei steht n_{grob} für die Anzahl an Grobgitterpunkten pro Dimension.

Implementierung Die Implementierung verläuft analog zum JACOBI- und DEFECT-Kernel. Neben dem DEFECT-Kernel muss auch die Restriktion mit mehreren Datentypen umgehen können. Ein Residuum in doppelter Genauigkeit kann somit bei der Restriktion auf das gröbere Gitter in einfache Genauigkeit konvertiert werden. Dasselbe gilt für die Verwendung von Single und Half. Im Gegensatz zum DEFECT-Kernel wird bei der Restriktion allerdings zunächst die Berechnung durchgeführt und anschließend konvertiert, was zu weniger Rundungsfehlern führt.

Aufwandsanalyse Die Durchführung einer Restriktion in doppelter Genauigkeit benötigt insgesamt

$$\begin{aligned} {}^{\mathrm{d}}\mathrm{CALC}_{\mathrm{res}}(\mathrm{n}_{\mathrm{grob}}) &= (1 \cdot {}^{\mathrm{d}}\mathrm{MUL} + 3 \cdot {}^{\mathrm{d}}\mathrm{ADD} + 1 \cdot {}^{\mathrm{d}}\mathrm{FMA}) \cdot \mathrm{n}_{\mathrm{grob}}^2 \\ &= 5 \cdot \mathrm{n}_{\mathrm{grob}}^2 \ {}^{\mathrm{d}}\mathrm{FLOP}. \end{aligned}$$

Für die spätere Analyse ist es hilfreich, den Aufwand in der Größenordnung des Levels anzugeben, auf welchem der Kernel ausgeführt wird. Im Falle der Restriktion bedeutet dies, den Aufwand in Abhängigkeit von n_{fein} anzugeben. Auf Grund der Standardvergröberung gilt

$$\mathrm{h} = \frac{1}{\mathrm{n}_{\mathrm{fein}} + 1} \qquad \text{und} \qquad 2 \cdot \mathrm{h} = \frac{1}{\mathrm{n}_{\mathrm{grob}} + 1}.$$

Setzt man die erste Gleichung in die Zweite ein und formt nach n_{grob} um, so folgt

$$\mathrm{n}_{\mathrm{grob}} = \frac{\mathrm{n}_{\mathrm{frein}} - 1}{2}.$$

Damit führt die Restriktion insgesamt

$${}^{\mathrm{d}}\mathrm{CALC}_{\mathrm{res}}(\mathrm{n}_{\mathrm{fein}}) = \frac{5}{4} \cdot (\mathrm{n}_{\mathrm{fein}} - 1)^2 \ {}^{\mathrm{d}}\mathrm{FLOP}$$

durch. Vom Residuum auf dem Feingitter werden nicht alle Werte benötigt. Abbildung 14 illustriert die Anwendung der Restriktion von einem Feingitter der Größe $\mathrm{n}_{\mathrm{fein}} = 7$. Hierbei sieht man, dass von den n_{fein}^2 inneren Gitterpunkten gerade $(n_{grob}+1)^2$ nicht betrachtet werden. Neben dem Lesen müssen noch zusätzlich n_{grob}^2 Gitterpunkte geschrieben werden. Damit verarbeitet der Double-RESTRICT-Kernel insgesamt

$$\begin{aligned} {}^{\mathrm{d}}\mathrm{MEM}_{\mathrm{res}}(\mathrm{n}_{\mathrm{fein}}) &= \left(\mathrm{n}_{\mathrm{fein}}^2 - (\mathrm{n}_{\mathrm{grob}} + 1)^2 + \mathrm{n}_{\mathrm{grob}}^2\right) \cdot 8 \ \mathrm{Byte} \\ &= 8 \cdot \mathrm{n}_{\mathrm{fein}}^2 - 8 \cdot \mathrm{n}_{\mathrm{fein}} \ \mathrm{Byte} \end{aligned}$$

an Daten.

Bei der gemischt genauen Restriktion mit doppelter und einfacher Genauigkeit muss zusätzlich noch eine Konvertierung durchgeführt werden. Damit berechnet sich der Rechenaufwand zu

$${}^{\mathrm{ds}}\mathrm{CALC}_{\mathrm{res}}(\mathrm{n}_{\mathrm{fein}}) = \frac{5}{4} \cdot (\mathrm{n}_{\mathrm{fein}} - 1)^2 \ {}^{\mathrm{d}}\mathrm{FLOP} + \frac{1}{4} \cdot (\mathrm{n}_{\mathrm{fein}} - 1)^2 \ {}^{\mathrm{ds}}\mathrm{FLOP}.$$

Der Speicherzugriff verringert sich dabei auf

$$\begin{aligned} {}^{\mathrm{ds}}\mathrm{MEM}_{\mathrm{res}}(\mathrm{n}_{\mathrm{fein}}) &= \left(\mathrm{n}_{\mathrm{fein}}^2 - (\mathrm{n}_{\mathrm{grob}} + 1)^2\right) \cdot 8 \ \mathrm{Byte} + \mathrm{n}_{\mathrm{grob}}^2 \cdot 4 \ \mathrm{Byte} \\ &= 7 \cdot \mathrm{n}_{\mathrm{fein}}^2 - 6 \cdot \mathrm{n}_{\mathrm{fein}} - 1 \ \mathrm{Byte}. \end{aligned}$$

Die Berechnung für die Datentypen Single und Half verlaufen analog.

PROLONGATE

Mit dem Textur-Speicher stellt NVIDIA eine hardware-unterstützte Möglichkeit zur Interpolation bereit. Diese ist jedoch auf einfache Genauigkeit beschränkt, weshalb im Zuge dieser Arbeit ein eigener Interpolations-Kernel geschrieben wurde, welcher für alle betrachteten Datentypen verwendet werden kann.

Implementierung Wie schon bei den obigen Kerneln kann der Textur-Zwischenspeicher auch bei der Prolongation in Kombination mit dem geteilten Speicher verwendet werden, sodass die zweidimensionale räumliche Struktur ausgenutzt werden kann. Analog zum RESTRICT-Kernel muss auch die Prolongation mit zwei Datentypen umgehen können. Hierbei werden die Werte des Vektors auf dem Grobgitter zunächst konvertiert und im Anschluss die Berechnung durchgeführt.

Aufwandsanalyse Bei der bilinearen Interpolation in zwei Dimensionen gibt es pro zu berechnendem Wert vier Möglichkeiten: horizontale Mittlung, vertikale Mittlung, zentrierte Mittlung und direkte Übernahme des groben Werts, siehe hierzu auch Abbildung 7 aus Kapitel 2.2. Um bei der Implementierung auf Fallunterscheidungen verzichten zu können, kann wie folgt vorgegangen werden: Pro Thread wird je eine horizontale, eine vertikale und eine zentrierte Mittlung durchgeführt. Die Interpolation arbeitet sich damit in Form von Vierecken durch das gesamte Gebiet hindurch. Diese Sichtweise ist schematisch in Abbildung 14 dargestellt. Auf

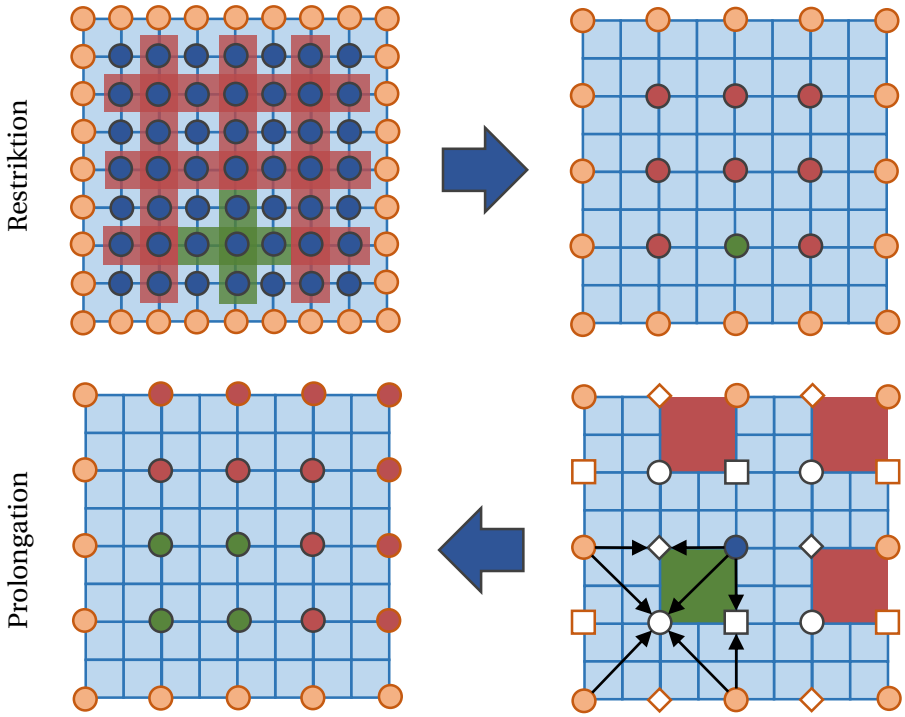


Abbildung 14: Oben: Illustration der Implementierung der Restriktion. Exemplarisch ist die Anwendung eines halb-gewichteten Stencils in Grün markiert. Unten: Schematische Darstellung der Implementierung der bilinearen Interpolation. In Grün ist exemplarisch ein Viereck markiert, welches die Berechnungen eines Threads zusammenfasst.

Grund der Vereinfachung durch Dirichlet-Randbedingungen sind die Werte an den Rändern Null. Dadurch können sie bei der Berechnung der Interpolation ignoriert werden, sofern die Vektoren zuvor mit Null initialisiert wurden. Pro Viereck werden vier Elemente des Grobgitters benötigt, welche durch den Textur-Zwischenspeicher in den geteilten Speicher geladen werden. Insgesamt werden für die Berechnung alle $(n_{grob}+2)^2$ Grobgitterpunkte verwendet, weshalb die Interpolation

$$\begin{aligned} {}^{\mathrm{d}}\mathrm{MEM}_{\mathrm{pro}}(n_{\mathrm{fein}}) &= \left((n_{\mathrm{grob}}+2)^2+(n_{\mathrm{fein}}+1)^2\right)\cdot 8\,\mathrm{Byte} \\ &= 10\cdot n_{\mathrm{fein}}^2+28\cdot n_{\mathrm{fein}}+26\,\mathrm{Byte} \end{aligned}$$

an Daten verarbeitet. Der Summand $(n_{fein}+1)^2$ stellt dabei den zu schreibenden Anteil dar. Das Feingitter lässt sich in $(n_{grob}-1)^2$ Vierecke unterteilen, wobei pro Viereck acht elementare Rechenoperationen durchgeführt werden (zwei für die horizontale, zwei für die vertikale und vier für die zentrale Mittlung). Durch die Anwendung einer Prolongation werden damit

$$\begin{aligned} {}^{\mathrm{d}}\mathrm{CALC}_{\mathrm{pro}}(n_{\mathrm{fein}}) &= (3 \cdot {}^{\mathrm{d}}\mathrm{MUL} + 5 \cdot {}^{\mathrm{d}}\mathrm{ADD}) \cdot (n_{\mathrm{grob}}-1)^2 \\ &= 2 \cdot (n_{\mathrm{fein}}-3)^2 \, {}^{\mathrm{d}}\mathrm{FLOP} \end{aligned}$$

durchgeführt. Für die Verwendung von gemischten Genauigkeiten ergibt sich ein Speicherbedarf von

$$\begin{aligned} {}^{\mathrm{ds}}\mathrm{MEM}_{\mathrm{pro}}(n_{\mathrm{fein}}) &= (n_{\mathrm{grob}}+2)^2 \cdot 4\ \mathrm{Byte} + (n_{\mathrm{fein}}+1)^2 \cdot 8\ \mathrm{Byte} \\ &= 9 \cdot n_{\mathrm{fein}}^2 + 22 \cdot n_{\mathrm{fein}} + 17\ \mathrm{Byte} \end{aligned}$$

für Double und Single. Pro Thread müssen die vier Grobgitterpunkte konvertiert werden. Damit besitzt die gemischt genaue Prolongation einen Rechenbedarf von

$$\begin{aligned} {}^{\mathrm{ds}}\mathrm{CALC}_{\mathrm{pro}}(n_{\mathrm{fein}}) &= (3 \cdot {}^{\mathrm{d}}\mathrm{MUL} + 5 \cdot {}^{\mathrm{d}}\mathrm{ADD} + 4 \cdot {}^{\mathrm{ds}}\mathrm{CONV}) \cdot (n_{\mathrm{grob}}-1)^2 \\ &= 2 \cdot (n_{\mathrm{fein}}-3)^2 \, {}^{\mathrm{d}}\mathrm{FLOP} + (n_{\mathrm{fein}}-3)^2 \, {}^{\mathrm{ds}}\mathrm{FLOP}. \end{aligned}$$

Der Speicherzugriff und Rechenbedarf für die anderen Genauigkeiten kann analog berechnet werden.

UPDATE

Beim UPDATE-Kernel werden die Vektoren elementweise addiert, was zu einer Berechnungsvorschrift von

$$a_{i,j} = b_{i,j} + c_{i,j},$$

für $i, j = 1, 2, \ldots, n$ führt.

Implementierung Im Gegensatz zu den stencil-basierten Kernel wird hierbei weder der geteilte Speicher, noch der Textur-Zwischenspeicher verwendet, da jedes Element lediglich einmal geladen werden muss.

Aufwandsanalyse Bei der Berechnung werden insgesamt

$${}^{\mathrm{d}}\mathrm{CALC}_{\mathrm{upd}}(n) = (1 \cdot {}^{\mathrm{d}}\mathrm{ADD}) \cdot n^2 = n^2 \, {}^{\mathrm{d}}\mathrm{FLOP}$$

durchgeführt und

$${}^{\mathrm{d}}\mathrm{MEM}_{\mathrm{upd}}(n) = 3 \cdot n^2 \cdot 8\ \mathrm{Byte} = 24 \cdot n^2\ \mathrm{Byte}$$

an Daten verarbeitet.

NORM

Die Berechnung der Norm eines Vektors kann in zwei Schritten erfolgen. Zunächst werden die einzelnen Vektorkomponenten quadriert und in den geteilten Speicher geschrieben. Als Nächstes werden dann die Threads innerhalb eines Blocks synchronisiert und pro Thread zwei Einträge der zuvor berechneten Produkte aufaddiert. Dieser Prozess wird so lange wiederholt, bis pro Block lediglich ein einzelner Wert übrig bleibt. Durch diese Herangehensweise entsteht eine Baumstruktur der Berechnung, welche in Abbildung 15 graphisch dargestellt ist. Die ver-

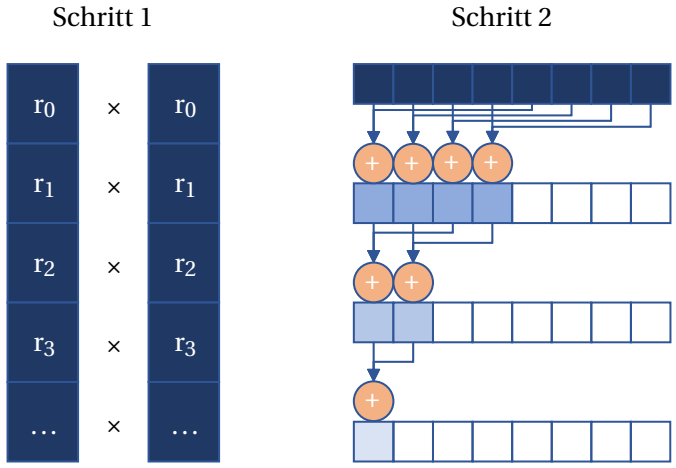


Abbildung 15: Links: Im ersten Schritt werden die Einträge des Vektors zunächst quadriert. Rechts: Im zweiten Schritt werden dann die einzelnen Einträge sukzessive aufaddiert, bis pro Block lediglich ein Wert übrig bleibt.

bleibenden Einträge werden zum Schluss zur CPU transferiert und sequentiell aufaddiert. Der letzte Schritt kann damit begründet werden, dass eine globale Kommunikation zwischen den Blöcken notwendig ist, um die letzten Einträge aufzusummieren. Dies hätte den Start eines weiteren Kernels zur Folge, was im Vergleich zur verhältnismäßig kleinen Zahl an Rechenoperationen, die noch notwendig sind, sehr aufwändig ist.

Da NVIDIA mit CUBLAS bereits eine optimierte Bibliothek für Aufgaben der Linearen Algebra mittels CUDA bereitstellt, wurde im Folgenden nicht die eigene, sondern die Implementierung aus CUBLAS verwendet. Hierbei ist anzumerken, dass CUBLAS nicht vollständig den Datentyp Half unterstützt. Für den NORM-Kernel kann zwar Half als Ein- und Ausgabeformat festgelegt werden, die Berechnung selbst findet jedoch in einfacher Genauigkeit statt [22]. Da im Zuge dieser Arbeit die Norm lediglich für das Abbruchkriterium verwendet wird und alle anderen, für die Verfahren notwendigen Rechenoperationen in halber Genauigkeit durchgeführt werden können, ist diese Einschränkung akzeptabel. Zusätzlich wird bei der Zeitmessung der Verfahren keine Normberechnung durchgeführt, weshalb im Folgenden auf die nähere Analyse des Rechenbedarfs und des Speicherzugriffs verzichtet wird.

Zusammenfassung Zum Ende dieses Unterkapitels werden der Rechenbedarf und der Speicherzugriff der einzelnen Kernel noch einmal zusammengefasst. Da für die Berechnung des Speed-Ups nicht die absoluten Werte, sondern das Verhältnis der Ausführungszeiten relevant ist, können die Gleitkommaoperationen der einzelnen Datentypen vereinheitlicht werden. Im Folgenden werden alle Gleitkommaoperationen in das Äquivalent einer Double-Gleitkommaoperation umgerechnet. Der entsprechende Faktor kann aus Tabelle 3 entnommen werden. So entspricht beispielsweise die Ausführung von vier Gleitkommaoperationen in halber Genauigkeit der Ausführung von einer Gleitkommaoperation in doppelter Genauigkeit, d.h.

$$4\,^{\mathrm{h}}\mathrm{FLOP} = 1\,^{\mathrm{d}}\mathrm{FLOP} = 1\,\mathrm{FLOP}.$$

Zur Vereinfachung wird im Folgenden bei der Einheit auch nicht mehr zwischen den Datentypen unterschieden. An dieser Stelle sei darauf hingewiesen, dass diese Vereinfachung lediglich für das vereinfachte Leistungsmodell verwendet werden kann. Für komplizierte Modelle, z.B. wenn die Berechnungen und Konvertierungen nicht vollständig seriell stattfinden, ist die Unterscheidung bei den einzelnen Gleitkommaoperationen notwendig. In Tabelle 4 ist der vereinheitlichte Rechenbedarf und Speicherzugriff für die jeweiligen Kernel zusammengefasst. Im

Kernel ker	Datentyp dt	${}^{dt}CALC_{ker}(n)$	${}^{dt}MEM_{ker}(n)$
JACOBI	Half	$2 \cdot n^2$ FLOP	$6 \cdot n^2 + 16 \cdot n + 16$ Byte
	Single	$4 \cdot n^2$ FLOP	$12 \cdot n^2 + 32 \cdot n + 32$ Byte
	Double	$8 \cdot n^2$ FLOP	$24 \cdot n^2 + 64 \cdot n + 64$ Byte
DEFECT	Half	$5/4 \cdot n^2$ FLOP	$6 \cdot n^2 + 8 \cdot n$ Byte
	Single-Half	$25/2 \cdot n^2$ FLOP	$10 \cdot n^2 + 8 \cdot n$ Byte
	Single	$5/2 \cdot n^2$ FLOP	$12 \cdot n^2 + 16 \cdot n$ Byte
	Double-Half	$15 \cdot n^2$ FLOP	$18 \cdot n^2 + 8 \cdot n$ Byte
	Double-Single	$15 \cdot n^2$ FLOP	$20 \cdot n^2 + 16 \cdot n$ Byte
	Double	$5 \cdot n^2$ FLOP	$24 \cdot n^2 + 32 \cdot n$ Byte
RESTRICT	Half	$5/16 \cdot (n-1)^2$ FLOP	$2 \cdot n^2 - 2 \cdot n$ Byte
	Single-Half	$9/8 \cdot (n-1)^2$ FLOP	$7/2 \cdot n^2 - 3 \cdot n - 1/2$ Byte
	Single	$5/8 \cdot (n-1)^2$ FLOP	$4 \cdot n^2 - 4 \cdot n$ Byte
	Double-Half	$7/4 \cdot (n-1)^2$ FLOP	$13/2 \cdot n^2 - 5 \cdot n - 3/2$ Byte
	Double-Single	$7/4 \cdot (n-1)^2$ FLOP	$7 \cdot n^2 - 6 \cdot n - 1$ Byte
	Double	$5/4 \cdot (n-1)^2$ FLOP	$8 \cdot n^2 - 8 \cdot n$ Byte
PROLONGATE	Half	$1/2 \cdot (n-3)^2$ FLOP	$5/2 \cdot n^2 + 7 \cdot n + 13/2$ Byte
	Single-Half	$3 \cdot (n-3)^2$ FLOP	$9/2 \cdot n^2 + 11 \cdot n + 17/2$ Byte
	Single	$1 \cdot (n-3)^2$ FLOP	$5 \cdot n^2 + 14 \cdot n + 13$ Byte
	Double-Single	$4 \cdot (n-3)^2$ FLOP	$9 \cdot n^2 + 22 \cdot n + 7$ Byte
	Double	$2 \cdot (n-3)^2$ FLOP	$10 \cdot n^2 + 28 \cdot n + 26$ Byte
UPDATE	Half	$1/4 \cdot n^2$ FLOP	$6 \cdot n^2$ Byte
	Single	$1/2 \cdot n^2$ FLOP	$12 \cdot n^2$ Byte
	Double	$1 \cdot n^2$ FLOP	$24 \cdot n^2$ Byte

Tabelle 4: Auflistung des Rechenbedarfs und Speicherzugriffs der einzelnen Kernel, abhängig von der Problemgröße.

Fall von Restriktion und Prolongation steht n für die Anzahl innerer Gitterpunkte auf dem Feingitter.

Beim vereinfachten Leistungsmodell berechnet sich die Ausführungszeit pro Kernel durch das Maximum zwischen Speicher- und Rechenzeit (vgl. Gleichung (8)). Um die Zeiten zu erhalten, wird beim Rechenbedarf durch das theoretische Maximum ${}^{d}P_{max} = 7.5$ TFLOP/s und beim Speicherzugriff durch die theoretische Speicherbandbreite $b_s = 900$ GByte/s geteilt. Dies führt dazu, dass die Speicherzeiten für alle Kernel und für alle Datentypen deutlich länger sind, als die Rechenzeit. Das Maximum der Ausführungszeiten wird dadurch in jedem Fall durch den Speicherzugriff bestimmt. Die Ausführungszeit ${}^{dt}t_{ker}$ für einen Kernel *ker* mit Datentyp *dt* berechnet sich dadurch zu

$$ {}^{dt}t_{ker} = \frac{{}^{dt}MEM_{ker}}{b_s}. $$

Nachdem die einzelnen Kernel und dessen Implementierung näher erläutert wurden, sollen als Nächstes die gemischt genauen Verfahren motiviert und hergeleitet werden. Zusätzlich kann ausgehend von der oben durchgeführten Analyse des Rechen- und Speicheraufwands für die jeweiligen Verfahren eine obere Schranke für den theoretischen Speed-Up hergeleitet werden.

3.3 Gemischt genaue Horizontal-Mehrgitterverfahren

Eine Möglichkeit, um unterschiedliche Datentypen bei einem Mehrgitterverfahren zu verwenden, besteht in einer horizontalen Denkweise. Betrachtet man ein Mehrgitterverfahren, so werden iterativ einzelne Zyklen nacheinander ausgeführt. Der horizontale Ansatz besteht nun darin, die Iterationen mit einem weniger genauen Datentyp zu beginnen und ab einer definierten Grenze den Datentyp zu wechseln und damit die Genauigkeit zu erhöhen. Abbildung 16 verdeutlicht diese Denkweise schematisch mit V-Zyklen. Da horizontal gesehen der Wechsel

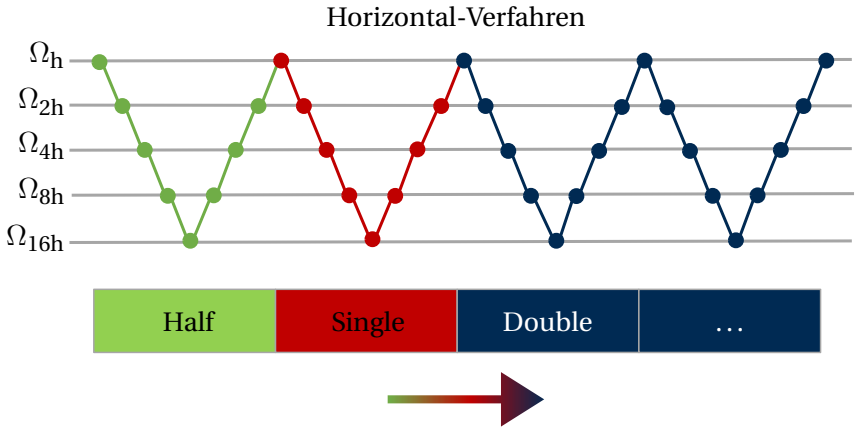


Abbildung 16: Visualisierung eines Horizontal-Verfahrens mit V-Zyklen. Der Wechsel der Datentypen findet horizontal gesehen von links nach rechts statt.

der Datentypen stattfindet, werden diese Ansätze im Zuge dieser Arbeit als *gemischt genaue Horizontal-Verfahren* bezeichnet.

Für die hier betrachteten Datentypen Double, Single und Half entstehen durch diese Herangehensweise zwei Verfahren, die im Folgenden eingeführt und theoretisch analysiert werden. Die beiden Verfahren unterscheiden sich lediglich in der Anzahl an Datentypen, die verwendet werden. Das erste hier betrachtete Verfahren verwendet lediglich einfache und doppelte Genauigkeit und wird deshalb als *Horizontales-Zweischicht-Mehrgitterverfahren* (H2SMG-Verfahren) bezeichnet.

Horizontales-Zweischicht-Mehrgitterverfahren (H2SMG-Verfahren)

Das H2SMG-Verfahren ist wie ein herkömmliches Mehrgitterverfahren iterativ aufgebaut und beginnt mit der Durchführung mehrerer Mehrgitterzyklen in einfacher Genauigkeit. Nach einer gewissen Anzahl an Durchläufen wird die Lösung der letzten Single-Iteration in doppelte Genauigkeit konvertiert und dient somit als Startlösung für die anschließenden Double-Iterationen. Das H2SMG-Verfahren ist in Algorithmus 6 aufgeführt. Der Befehl ${}^{ds}\mathbf{GGKH2SMG}\left(k, {}^{s}\mathbf{v}_k^{(m+1/3)}, {}^{s}\mathbf{f}_k\right)$ steht dabei für die Durchführung der Grobgitterkorrektur des H2SMG-Vefahrens mit Tiefe k, Approximation ${}^{s}\mathbf{v}_k^{(m+1/3)}$ und rechter Seite ${}^{s}\mathbf{f}_k$. Analog steht der Befehl ${}^{d}\mathbf{GGKMG}\left(k, {}^{d}\mathbf{v}_k^{(m+1/3)}, {}^{d}\mathbf{f}_k\right)$ für die Durchführung der herkömmlichen Grobgitterkorrektur in doppelter Genauigkeit (vgl. Algorithmus 5). Die Tiefe des Verfahrens ergibt sich aus der feinsten Diskretisierung h und der minimalen Tiefe k_{min}, die auf der verwendeten Hardware noch sinnvoll durchgeführt werden kann.

Da das Problem in doppelter Genauigkeit gestellt wird, muss sowohl die rechte Seite ${}^{d}\mathbf{f}_h$ als auch die Startlösung ${}^{d}\mathbf{v}_h^{(0)}$ zunächst nach Single konvertiert werden. Dieser Schritt kann jedoch im Rahmen der Vorarbeit durchgeführt werden. Zwischen den beiden Schichten muss

Algorithmus 6 H2SMG-Verfahren

Eingabe: Rechte Seite ${}^{d}\mathbf{f}_k$, Startlösung ${}^{d}\mathbf{v}_k^{(0)}$
Ausgabe: Approximation ${}^{d}\mathbf{v}_k$
Vorarbeit: Konvertiere ${}^{d}\mathbf{f}_k$ und ${}^{d}\mathbf{v}_k^{(0)}$ nach Single $\longrightarrow {}^{s}\mathbf{f}_k, {}^{s}\mathbf{v}_k^{(0)}$

Für m = 0,1,2,...:
  Vorglättung:
    Berechne ${}^{s}\mathbf{v}_k^{(m+\frac{1}{3})}$ durch k_{vor}-malige Anwendung des gedämpften
      Jacobi-Verfahrens in Single: ${}^{s}\mathbf{v}_k^{(m+\frac{1}{3})} = {}^{s}\mathbf{JACOBI}^{k_{vor}}({}^{s}\mathbf{v}_k^{(m)}, {}^{s}\mathbf{f}_k)$
  Grobgitterkorrektur:
    Berechne ${}^{s}\mathbf{v}_k^{(m+\frac{2}{3})}$ durch Anwendung der Grobgitterkorrektur
      des H2SMG-Verfahrens: ${}^{s}\mathbf{v}_k^{(m+\frac{2}{3})} = {}^{ds}\mathbf{GGKH2SMG}(k, {}^{s}\mathbf{v}_k^{(m+\frac{1}{3})}, {}^{s}\mathbf{f}_k)$
  Nachglättung:
    Berechne ${}^{s}\mathbf{v}_k^{(m+1)}$ durch k_{nach}-malige Anwendung des gedämpften
      Jacobi-Verfahrens in Single: ${}^{s}\mathbf{v}_k^{(m+1)} = {}^{s}\mathbf{JACOBI}^{k_{nach}}({}^{s}\mathbf{v}_k^{(m+\frac{2}{3})}, {}^{s}\mathbf{f}_k)$

Konvertiere Approximation ${}^{s}\mathbf{v}_k$ nach Double $\longrightarrow {}^{d}\mathbf{v}_k^{(0)}$

Für m = 0,1,2,...:
  Vorglättung:
    Berechne ${}^{d}\mathbf{v}_k^{(m+\frac{1}{3})}$ durch k_{vor}-malige Anwendung des gedämpften
      Jacobi-Verfahrens in Double: ${}^{d}\mathbf{v}_k^{(m+\frac{1}{3})} = {}^{d}\mathbf{JACOBI}^{k_{vor}}({}^{d}\mathbf{v}_k^{(m)}, {}^{d}\mathbf{f}_k)$
  Grobgitterkorrektur:
    Berechne ${}^{d}\mathbf{v}_k^{(m+\frac{2}{3})}$ durch Anwendung der Grobgitterkorrektur
      des MG-Verfahrens: ${}^{d}\mathbf{v}_k^{(m+\frac{2}{3})} = {}^{d}\mathbf{GGKMG}(k, {}^{d}\mathbf{v}_k^{(m+\frac{1}{3})}, {}^{d}\mathbf{f}_k)$
  Nachglättung:
    Berechne ${}^{d}\mathbf{v}_k^{(m+1)}$ durch k_{nach}-malige Anwendung des gedämpften
      Jacobi-Verfahrens in Double: ${}^{d}\mathbf{v}_k^{(m+1)} = {}^{d}\mathbf{JACOBI}^{k_{nach}}({}^{d}\mathbf{v}_k^{(m+\frac{2}{3})}, {}^{d}\mathbf{f}_k)$

die Single-Approximation nach Double konvertiert werden, weshalb dieses Verfahren einen gewissen Mehraufwand besitzt. Wie bereits angesprochen ist es beim H2SMG-Verfahren wesentlich, dass das Residuum auf dem feinsten Gitter in doppelter Genauigkeit berechnet wird. Dies gilt zumindest dann, wenn das Residuum als Abbruchkriterium verwendet werden soll. Hierzu werden der gemischt genaue DEFECT- und RESTRICT-Kernel verwendet. In Algorithmus 7 ist die Grobgitterkorrektur einer Single-Iteration beim H2SMG-Verfahren aufgeführt. Nach der Berechnung des Residuums in doppelter Genauigkeit kann durch Anwendung des NORM-Kernels eine Konvergenzkontrolle durchgeführt werden. Sofern noch nicht das kleinste Level k_{min} erreicht ist, wird in der Grobgitterkorrektur des H2SMG-Verfahrens ein Mehrgitterverfahren in einfacher Genauigkeit zum Lösen der Residuumsgleichung verwendet.

Theoretischer Speed-Up

Für die Berechnung des theoretischen Speed-Ups werden im Folgenden lediglich Verfahren mit V-Zyklen betrachtet. Bei Verfahren basierend auf anderen Zyklen lässt sich eine obere Schranke an den theoretischen Speed-Up analog herleiten. Es wird zunächst der Rechenaufwand einer

Algorithmus 7 Grobgitterkorrektur der Single-Iteration beim H2SMG-Verfahrens

Berechnung von ${}^{s}\mathbf{v}_k^{(m+\frac{2}{3})} = {}^{ds}\mathbf{GGKH2SMG}(k, {}^{s}\mathbf{v}_k^{(m+\frac{1}{3})}, {}^{s}\mathbf{f}_k)$:
- Falls $k = k_{max}$:
  - Berechne das Residuum in Double: ${}^{d}\mathbf{r}_k = {}^{ds}\mathbf{DEFECT}({}^{s}\mathbf{v}_k, {}^{d}\mathbf{f}_k)$
  - Berechne die Norm des Residuums in Double: $||{}^{d}\mathbf{r}_k|| = {}^{d}\mathbf{NORM}({}^{d}\mathbf{r}_k)$
  - Führe eine Konvergenzkontrolle durch
  - Restringiere das Residuum nach Single: ${}^{s}\hat{\mathbf{r}}_{k-1} = {}^{ds}\mathbf{RESTRICT}({}^{d}\mathbf{r}_k)$
- Falls $k < k_{max}$:
  - Berechne das Residuum in Single: ${}^{s}\mathbf{r}_k = {}^{s}\mathbf{DEFECT}({}^{s}\mathbf{v}_k, {}^{s}\mathbf{f}_k)$
  - Restringiere das Residuum in Single: ${}^{s}\hat{\mathbf{r}}_{k-1} = {}^{s}\mathbf{RESTRICT}({}^{s}\mathbf{r}_k)$
- Berechne eine Approximation ${}^{s}\hat{\mathbf{e}}_{k-1}$ der Residuumsgleichung durch:
  - Falls $k = k_{min}$:
    - Approximiere mit dem ungedämpften Jacobi-Verfahren in Single: ${}^{s}\hat{\mathbf{e}}_{k-1} = {}^{s}\mathbf{JACOBI}^{k_{lösen}}({}^{s}\mathbf{0}_{k-1}, {}^{s}\hat{\mathbf{r}}_{k-1})$
  - Falls $k > k_{min}$:
    - Approximiere durch die Anwendung des Mehrgitterverfahrens in Single: ${}^{s}\hat{\mathbf{e}}_{k-1} = {}^{s}\mathbf{MG}(k-1, {}^{s}\mathbf{0}_{k-1}, {}^{s}\hat{\mathbf{r}}_{k-1})$
- Prolongiere die Korrektur in Single: ${}^{s}\hat{\mathbf{e}}_k = {}^{s}\mathbf{PROLONGATE}({}^{s}\hat{\mathbf{e}}_{k-1})$
- Korrigiere die Approximation in Single: ${}^{s}\mathbf{v}_k = {}^{s}\mathbf{UPDATE}({}^{s}\mathbf{v}_k, {}^{s}\hat{\mathbf{e}}_k)$

Iteration des V-Zyklus in doppelter Genauigkeit betrachtet. Hierbei wird der Mehraufwand für die Konvertierung der Startlösung und der rechten Seite vernachlässigt. Pro Level werden bei einem V-Zyklus die folgenden Operationen durchgeführt:

- 3x Vorglättungsschritte
- 1x Residuumsberechnung
- 1x Restriktion
- 1x Prolongation
- 1x Vektoraddition
- 3x Nachglättungsschritte

Die Berechnung des Abbruchkriteriums (z.B. durch Anwendung des NORM-Kernels) wird dabei vernachlässigt. Damit ergibt sich eine theoretische Ausführungszeit von

$$\begin{aligned} {}^{d}T_{vzl}(n) &= \frac{1}{b_s}\left(6 \cdot {}^{d}MEM_{jac}(n) + {}^{d}MEM_{def}(n) + {}^{d}MEM_{res}(n) + {}^{d}MEM_{pro}(n) + {}^{d}MEM_{upd}(n)\right) \\ &= \frac{1}{900 \cdot 1{,}00E+09}\left(210 \cdot n^2 + 436 \cdot n + 410\right) \text{ Sekunden} \end{aligned}$$

pro Level. Das Subskript *vzl* steht dabei für V-Zyklus pro Level. Da die Ausführungszeit eines jeden Kernels durch den Speicherzugriff dominiert ist, beinhaltet der jeweilige Term stets den Faktor $1/b_s$. Da sich die Ausführungszeit als Summe der einzelnen Kernelzeiten zusammensetzt, ist es möglich, den Faktor aus der Summe auszuklammern:

$$T = \sum_{kernel} t_{kernel} = \frac{1}{b_s} \cdot \sum_{kernel} MEM_{kernel}.$$

Bei der Berechnung des Speed-Ups kann er somit gekürzt werden:

$$Speed-Up = \frac{T}{T'} = \frac{1/b_s \cdot \sum_{kernel} MEM_{kernel}}{1/b_s \cdot \sum_{kernel'} MEM_{kernel'}} = \frac{\sum_{kernel} MEM_{kernel}}{\sum_{kernel'} MEM_{kernel'}}.$$

Aus diesem Grund wird im Folgenden der Faktor $1/b_s$ bei der Berechnung der theoretischen Ausführungszeit weggelassen. Da sich auch die Einheiten bei der Quotientenbildung kürzen,

wird im Folgenden auch auf die Angabe der Einheit verzichtet. Die Begriffe *Aufwand* und *Ausführugnszeit* werden im Folgenden deshalb synonym verwendet. Auf Grund der obigen Vereinfachungen berechnet sich die theoretische Ausführungszeit eines Levels zu

$$^{d}T_{vzl}(n) = 210 \cdot n^2 + 436 \cdot n + 410.$$

Hierbei ist zu beachten, dass der Aufwand abhängig von der Anzahl an inneren Gitterpunkten pro Dimension angegeben ist. Auf Grund der Standardvergröberung gilt

$$h = \frac{1}{n+1} = 2^{-k}$$

mit dem Level *k*. Stellt man die letzte Gleichung nach *n* um, so folgt

$$n = 2^k - 1. \tag{9}$$

Somit lässt sich der Aufwand auch abhängig vom jeweiligen Level beschreiben. Auf dem gröbsten Level werden 151 Iterationen des (ungedämpften) Jacobi-Verfahrens durchgeführt. Analog zu Kapitel 2.4 wird im Folgenden mit k_{min} das gröbste und mit k_{max} das feinste Gitter bezeichnet. Die Ausführungszeit für einen V-Zyklus in doppelter Genauigkeit berechnet sich damit zu

$$^{d}T_{vz}(k_{max}, k_{min}) = \sum_{k=k_{min}+1}^{k_{max}} \left(^{d}T_{vzl}(2^k - 1) \right) + 151 \cdot {}^{d}MEM_{jac}(2^{k_{min}} - 1). \tag{10}$$

Der erste Term in (10) ist eine endliche Summe aus Polynomen vom Grad zwei bzgl. 2^k. Unter Verwendung der Gaußschen Summenformel

$$\sum_{k=1}^{n} k = \frac{n \cdot (n+1)}{2}$$

und der quadratischen Variante

$$\sum_{k=1}^{n} k^2 = \frac{n \cdot (n+1) \cdot (2n+1)}{6}$$

lässt sich der Ausdruck vereinfachen zu

$$^{d}T_{vz}(n_{max}, n_{min}) = 280 \cdot n_{max}^2 + 592 \cdot n_{max} + 3344 \cdot n_{min}^2 + 9072 \cdot n_{min} + 184 \cdot \log_2 \frac{n_{max}+1}{n_{min}+1} + 9664.$$

Hierbei wurde auf Grund der einfacheren Terme der Aufwand in Abhängigkeit von n_{max} und n_{min} angegeben, welche sich gemäß Gleichung (9) aus k_{max} und k_{min} bestimmen lassen.

Für die Single-Iterationen kann die Ausführungszeit analog hergeleitet werden, wobei darauf geachtet werden muss, dass auf dem obersten Level der gemischt genaue DEFECT- und RESTRICT-Kernel verwendet wird, d.h.

$$\begin{aligned} ^{s}T_{vzl}(n_{max}) = 6 \cdot {}^{s}MEM_{jac}(n_{max}) + {}^{ds}MEM_{def}(n_{max}) + {}^{ds}MEM_{res}(n_{max}) \\ + {}^{s}MEM_{pro}(n_{max}) + {}^{s}MEM_{upd}(n_{max}). \end{aligned}$$

Die Ausführungszeit insgesamt berechnet sich damit zu

$$\begin{aligned} ^{s}T_{vz}(n_{max}, n_{min}) &= {}^{s}T_{vzl}(n_{max}) + \sum_{k=k_{min}+1}^{k_{max}-1} \left(^{s}T_{vzl}(2^k - 1) \right) + 151 \cdot {}^{s}MEM_{jac}(n_{min}) \\ &= 151 \cdot n_{max}^2 + 306 \cdot n_{max} + 1672 \cdot n_{min}^2 + +4536 \cdot n_{min} + 92 \cdot \log_2 \frac{n_{max}+1}{n_{min}+1} + 4831 \end{aligned}$$

Der theoretischen Speed-Up vom H2SMG-Verfahren hängt maßgeblich von der Anzahl an durchgeführten Single-Iterationen ab. Im Folgenden bezeichnet α das Verhältnis zwischen Single-Iterationen und der Gesamtanzahl an durchgeführten Iterationen, d.h.

$$\alpha = \frac{\text{Anzahl Single-Iterationen}}{\text{Anzahl gesamter Iterationen}}.$$

Die Wahl von α ist a priori schwer festzulegen und abhängig von der Problemgröße n_{max}. In Kapitel 4.4 wird auf die optimale Wahl des Verhältnisses näher eingegangen. Der Speed-Up für das H2SMG-Verfahren berechnet sich unter Hinzunahme des Parameters α zu

$$\text{Speed}-\text{Up} = \frac{{}^{d}T_{mg}}{{}^{ds}T_{h2smg}} = \frac{{}^{d}T_{vz}}{\alpha \cdot {}^{s}T_{vz} + (1-\alpha) \cdot {}^{d}T_{vz}}.$$

Dies gilt zumindest dann, wenn für beide Verfahren insgesamt gleich viele Iterationen durchgeführt werden. In Tabelle 5 ist der theoretische Speed-Up für unterschiedliche Möglichkeiten von α für die Level 8 und 12 aufgelistet. Da bei der Implementierung nicht bis zum kleinsten Level vergröbert wird, wird im Folgenden $k_{min} = 5$ angenommen (vgl. Kapitel 3.1).

α	k_{max}	Theo. Speed-Up
1.0	8	1,88
	12	1,85
0.75	8	1,54
	12	1,53
0.5	8	1,30
	12	1,30
0.25	8	1,13
	12	1,13

Tabelle 5: Theoretischer Speed-Up des H2SMG-Verfahrens in Abhängigkeit des Parameters α und der Problemgröße k_{max}.

Es zeigt sich, dass der Speed-Up nicht größer als 2,0 ist. Dies ist anschaulich klar, da für die Ausführungszeit lediglich der Speicherzugriff relevant ist und bei Verwendung von einfacher Genauigkeit gerade die Hälfte des Speicherplatzes benötigt wird. Im Optimalfall werden alle Iterationen des H2SMG-Verfahrens in einfacher Genauigkeit durchgeführt ($\alpha = 1,0$). Dass der Speed-Up für die betrachteten Problemgrößen kleiner als 2,0 ist, liegt an dem gemischt genauen DEFECT- und RESTRICT-Kernel. Diese laden bzw. schreiben selbst bei den Single-Iterationen Vektoren in doppelter Genauigkeit. Bemerkenswert ist ebenfalls, dass der Speed-Up mit steigender Problemgröße sinkt. Dies lässt sich wie folgt erklären: Für kleine Problemgrößen ist der zusätzliche Aufwand der gemischt genauen DEFECT- und RESTRICT- Kernel im Gegensatz zu den restlichen Operationen vernachlässigbar. Mit steigender Problemgröße fängt das größte Level an, den Aufwand und damit den Speed-Up zu dominieren. Da selbst bei den Single-Iterationen ein Teil des größten Levels in doppelter Genauigkeit gerechnet wird, fällt der Zusatzaufwand der gemischt genauen Kernel für große Probleme stark ins Gewicht. Für die betrachteten Fälle befindet sich der theoretische Speed-Up zwischen 10 und 90 Prozent. Für kleinere Werte von α fällt er deutlich ab. Der Unterschied auf Grund der Problemgröße verschwindet dabei ebenfalls, da insgesamt weniger Single-Iterationen durchgeführt werden.

Nachdem das zweischichtige Horizontal-Verfahren hergeleitet und der theoretische Speed-Up diskutiert wurde, erfolgt als Nächstes die Herleitung des H3SMG-Verfahrens unter Hinzunahme des Datentyp Half.

Horizontales-Dreischicht-Mehrgitterverfahren (H3SMG-Verfahren)

Das *horizontale Dreischicht-Mehrgitterverfahren* geht analog zum H2SMG-Verfahren vor. Hierbei werden vor den Single-Iterationen zunächst einige Half-Iterationen durchgeführt. Um das Residuum als Abbruchkriterium verwenden zu können, muss die Berechnung in den Half-Iterationen wie schon beim H2SMG-Verfahren in doppelter Genauigkeit erfolgen. Hierfür wird der gemischt genaue DEFECT- und RESTRICT-Kernel mit den Datentypen Double und Half verwendet. In Algorithmus 8 ist die Grobgitterkorrektur einer Half-Iteration beim H3SMG-Verfahren aufgeführt. Die Konvergenzkontrolle kann analog zum H2SMG-Verfahren direkt nach

Algorithmus 8 Grobgitterkorrektur der Half-Iteration beim H3SMG-Verfahrens

Berechnung von ${}^{h}\mathbf{v}_k^{(m+\frac{2}{3})} = {}^{dh}\mathbf{GGKH3SMG}(k, {}^{h}\mathbf{v}_k^{(m+\frac{1}{3})}, {}^{h}\mathbf{f}_k)$:
- Falls $k = k_{max}$:
  - Berechne das Residuum in Double: ${}^{d}\mathbf{r}_k = {}^{dh}\mathbf{DEFECT}({}^{h}\mathbf{v}_k, {}^{d}\mathbf{f}_k)$
  - Berechne die Norm des Residuums in Double: $||{}^{d}\mathbf{r}_k|| = {}^{d}\mathbf{NORM}({}^{d}\mathbf{r}_k)$
  - Führe eine Konvergenzkontrolle durch
  - Restringiere das Residuum nach Half: ${}^{h}\hat{\mathbf{r}}_{k-1} = {}^{dh}\mathbf{RESTRICT}({}^{d}\mathbf{r}_k)$
- Falls $k < k_{max}$:
  - Berechne das Residuum in Single: ${}^{h}\mathbf{r}_k = {}^{h}\mathbf{DEFECT}({}^{h}\mathbf{v}_k, {}^{h}\mathbf{f}_k)$
  - Restringiere das Residuum in Single: ${}^{h}\hat{\mathbf{r}}_{k-1} = {}^{h}\mathbf{RESTRICT}({}^{h}\mathbf{r}_k)$
- Berechne eine Approximation ${}^{h}\hat{\mathbf{e}}_{k-1}$ der Residuumsgleichung durch:
  - Falls $k = k_{min}$:
    - Approximiere mit dem ungedämpften Jacobi-Verfahren in Half: ${}^{h}\hat{\mathbf{e}}_{k-1} = {}^{h}\mathbf{JACOBI}^{k_{lösen}}({}^{h}\mathbf{0}_{k-1}, {}^{h}\hat{\mathbf{r}}_{k-1})$
  - Falls $k > k_{min}$:
    - Approximiere durch die Anwendung eines Mehrgitterverfahrens in Half: ${}^{h}\hat{\mathbf{e}}_{k-1} = {}^{h}\mathbf{MG}(k-1, {}^{h}\mathbf{0}_{k-1}, {}^{h}\hat{\mathbf{r}}_{k-1})$
- Prolongiere die Korrektur in Half: ${}^{h}\hat{\mathbf{e}}_k = {}^{h}\mathbf{PROLONGATE}({}^{h}\hat{\mathbf{e}}_{k-1})$
- Korrigiere die Approximation in Half: ${}^{h}\mathbf{v}_k = {}^{h}\mathbf{UPDATE}({}^{h}\mathbf{v}_k, {}^{h}\hat{\mathbf{e}}_k)$

der Berechnung des Residuums auf dem feinsten Level erfolgen. Sofern das kleinste Level noch nicht erreicht ist ($k > k_{min}$) wird bei der Grobgitterkorrektur zum Lösen der Residuumsgleichung ein Mehrgitterverfahren in halber Genauigkeit verwendet. Das gesamte H3SMG-Verfahren ist in Algorithmus 9 aufgeführt. Da die Startlösung und rechte Seite in doppelter Genauigkeit vorliegen, müssen diese zunächst nach Single und Half konvertiert werden. Neben der Konvertierung zwischen den Single- und Double-Iterationen muss beim H3SMG-Verfahren auch zwischen den Half- und Single-Iterationen die neue Startlösung konvertiert werden. Das H3SMG-Verfahren besitzt dadurch einen geringfügig größeren Mehraufwand als das zweischichtige Horizontal-Verfahren.

Theoretischer Speedup

Für die Berechnung des theoretischen Speed-Ups werden wie beim H2SMG-Verfahren V-Zyklen betrachtet. Im Kapitel zuvor wurden bereits die Ausführungszeiten für die Single- und Double-V-Zyklen hergeleitet. Die Herleitung der theoretischen Ausführungszeit für die Durchführung eines V-Zyklus in halber Genauigkeit verläuft analog. Auf dem feinsten Level beträgt sie

$$
\begin{aligned}
{}^{h}T_{vzl}(k_{max}, k_{min}) = 6 \cdot {}^{h}MEM_{jac}(n_{max}) + {}^{dh}MEM_{def}(n_{max}) + {}^{dh}MEM_{res}(n_{max}) \\
+ {}^{h}MEM_{pro}(n_{max}) + {}^{h}MEM_{upd}(n_{max}).
\end{aligned}
$$

Algorithmus 9 H3SMG-Verfahren

Eingabe: Rechte Seite ${}^{d}\mathbf{f}_k$, Startlösung ${}^{d}\mathbf{v}_k^{(0)}$
Ausgabe: Approximation ${}^{d}\mathbf{v}_k$
Vorarbeit: Konvertiere ${}^{d}\mathbf{f}_k$ und ${}^{d}\mathbf{v}_k^{(0)}$ nach Single und Half $\longrightarrow {}^{s}\mathbf{f}_k, {}^{s}\mathbf{v}_k^{(0)}, {}^{h}\mathbf{f}_k, {}^{h}\mathbf{v}_k^{(0)}$

Für m = 0,1,2,...:
  Vorglättung:
    Berechne ${}^{h}\mathbf{v}_k^{(m+\frac{1}{3})}$ durch k_{vor}-malige Anwendung des gedämpften
      Jacobi-Verfahrens in Half: ${}^{h}\mathbf{v}_k^{(m+\frac{1}{3})} = {}^{h}\mathbf{JACOBI}^{k_{vor}}({}^{h}\mathbf{v}_k^{(m)}, {}^{h}\mathbf{f}_k)$
  Grobgitterkorrektur:
    Berechne ${}^{h}\mathbf{v}_k^{(m+\frac{2}{3})}$ durch Anwendung der Grobgitterkorrektur
      des H3SMG-Verfahrens: ${}^{h}\mathbf{v}_k^{(m+\frac{2}{3})} = {}^{dh}\mathbf{GGKH3SMG}(k, {}^{h}\mathbf{v}_k^{(m+\frac{1}{3})}, {}^{h}\mathbf{f}_k)$
  Nachglättung:
    Berechne ${}^{h}\mathbf{v}_k^{(m+1)}$ durch k_{nach}-malige Anwendung des gedämpften
      Jacobi-Verfahrens in Half: ${}^{h}\mathbf{v}_k^{(m+1)} = {}^{h}\mathbf{JACOBI}^{k_{nach}}({}^{h}\mathbf{v}_k^{(m+\frac{2}{3})}, {}^{h}\mathbf{f}_k)$

Konvertiere Approximation ${}^{h}\mathbf{v}_k$ nach Single $\longrightarrow {}^{s}\mathbf{v}_k^{(0)}$

Für m = 0,1,2,...:
  Vorglättung:
    Berechne ${}^{s}\mathbf{v}_k^{(m+\frac{1}{3})}$ durch k_{vor}-malige Anwendung des gedämpften
      Jacobi-Verfahrens in Single: ${}^{s}\mathbf{v}_k^{(m+\frac{1}{3})} = {}^{s}\mathbf{JACOBI}^{k_{vor}}({}^{s}\mathbf{v}_k^{(m)}, {}^{s}\mathbf{f}_k)$
  Grobgitterkorrektur:
    Berechne ${}^{s}\mathbf{v}_k^{(m+\frac{2}{3})}$ durch Anwendung der Grobgitterkorrektur
      des H2SMG-Verfahrens: ${}^{s}\mathbf{v}_k^{(m+\frac{2}{3})} = {}^{ds}\mathbf{GGKH2SMG}(k, {}^{s}\mathbf{v}_k^{(m+\frac{1}{3})}, {}^{s}\mathbf{f}_k)$
  Nachglättung:
    Berechne ${}^{s}\mathbf{v}_k^{(m+1)}$ durch k_{nach}-malige Anwendung des gedämpften
      Jacobi-Verfahrens in Single: ${}^{s}\mathbf{v}_k^{(m+1)} = {}^{s}\mathbf{JACOBI}^{k_{nach}}({}^{s}\mathbf{v}_k^{(m+\frac{2}{3})}, {}^{s}\mathbf{f}_k)$

Konvertiere Approximation ${}^{s}\mathbf{v}_k$ nach Double $\longrightarrow {}^{d}\mathbf{v}_k^{(0)}$

Für m = 0,1,2,...:
  Vorglättung:
    Berechne ${}^{d}\mathbf{v}_k^{(m+\frac{1}{3})}$ durch k_{vor}-malige Anwendung des gedämpften
      Jacobi-Verfahrens in Double: ${}^{d}\mathbf{v}_k^{(m+\frac{1}{3})} = {}^{d}\mathbf{JACOBI}^{k_{vor}}({}^{d}\mathbf{v}_k^{(m)}, {}^{d}\mathbf{f}_k)$
  Grobgitterkorrektur:
    Berechne ${}^{d}\mathbf{v}_k^{(m+\frac{2}{3})}$ durch Anwendung der Grobgitterkorrektur
      des MG-Verfahrens: ${}^{d}\mathbf{v}_k^{(m+\frac{2}{3})} = {}^{d}\mathbf{GGKMG}(k, {}^{d}\mathbf{v}_k^{(m+\frac{1}{3})}, {}^{d}\mathbf{f}_k)$
  Nachglättung:
    Berechne ${}^{d}\mathbf{v}_k^{(m+1)}$ durch k_{nach}-malige Anwendung des gedämpften
      Jacobi-Verfahrens in Double: ${}^{d}\mathbf{v}_k^{(m+1)} = {}^{d}\mathbf{JACOBI}^{k_{nach}}({}^{d}\mathbf{v}_k^{(m+\frac{2}{3})}, {}^{d}\mathbf{f}_k)$

Damit folgt für einen gesamten V-Zyklus mit halber Genauigkeit

$$
\begin{aligned}
{}^{h}T_{vz}(n_{max}, n_{min}) &= {}^{h}T_{vzl}(n_{max}) + \sum_{k=k_{min}+1}^{k_{max}-1} \left({}^{h}T_{vzl}(2^k - 1) \right) + 151 \cdot {}^{h}MEM_{jac}(n_{min}) \\
&= \frac{173}{2} \cdot n_{max}^2 + \frac{290}{2} \cdot n_{max} + \frac{1672}{2} \cdot n_{min}^2 + \frac{4536}{2} \cdot n_{min} + 46 \cdot \log_2 \frac{n_{max}+1}{n_{min}+1} + \frac{4829}{2}.
\end{aligned}
$$

Analog zum H2SMG-Verfahren wird zusätzlich ein Parameter β eingeführt, welcher das Verhältnis von Half-Iterationen zur Anzahl der insgesamt durchgeführten Iterationen charakterisiert:

$$
\beta = \frac{\text{Anzahl Half-Iterationen}}{\text{Anzahl gesamter Iterationen}}.
$$

Der Speed-Up für das H3SMG-Verfahren berechnet sich dadurch zu

$$
\text{Speed}-\text{Up} = \frac{{}^{d}T_{mg}}{{}^{dsh}T_{h3smg}} = \frac{{}^{d}T_{vz}}{\beta \cdot {}^{h}T_{vz} + \alpha \cdot {}^{s}T_{vz} + (1-\alpha-\beta) \cdot {}^{d}T_{vz}}.
$$

Für eine Wahl von $\alpha = \beta = 1/3$ resultiert $1-\alpha-\beta = 1/3$ und es werden je gleich viele Half-, Single- und Double-Iterationen durchgeführt. Der theoretische Speed-Up für unterschiedliche Werte von α und β ist in Tabelle 6 aufgeführt.

k_{max}	β	α	$1-\alpha-\beta$	Theo. Speed-Up
8	1.0	0.0	0.0	3,34
	0.8	0.2	0.0	2,89
	0.6	0.2	0.2	2,06
	0.4	0.4	0.2	1,88
	0.2	0.6	0.2	1,73
12	1.0	0.0	0.0	3,24
	0.8	0.2	0.0	2,82
	0.6	0.2	0.2	2,03
	0.4	0.4	0.2	1,85
	0.2	0.6	0.2	1,71

Tabelle 6: Theoretischer Speed-Up des H3SMG-Verfahrens in Abhängigkeit des Parameters α und β sowie der Problemgröße k_{max}.

Für $\beta = 1.0$ werden ausschließlich Half-Iterationen durchgeführt. Der Speed-Up beträgt hierfür knapp 3,34. Damit liegt der Wert deutlich unterhalb von 4,00, was dem reinen Verhältnis zwischen Double- und Half-Speicherplatz entspricht. Wie bereits beim H2SMG-Verfahren lässt sich dies anhand des Mehraufwands durch den gemischt genauen DEFECT- und RESTRICT-Kernel erklären. Sinkt der Wert von β, so reduziert sich auch der Speed-Up. Auf Level 8 beträgt der Wert ca. 1,73, wenn 20 Prozent der Iterationen in Half, 60 Prozent in Single und der Rest in Double durchgeführt werden. Für das 12. Level wird für dieselbe Wahl von α und β stets ein geringerer Speed-Up beobachtet. Auch dies lässt sich analog zum H2SMG-Verfahren auf Grund der Dominanz des feinsten Gitters erklären. Der kleinste Speed-Up für die betrachteten Fälle beträgt knapp 70 Prozent. An dieser Stelle sei erneut darauf hingewiesen, dass der tatsächliche Speed-Up sehr stark von der Wahl der Parameter α und β abhängt, welche wiederum Einfluss auf die Fehlerentwicklung haben.

Neben den Horizontal-Verfahren gibt es noch weitere Möglichkeiten, gemischte Genauigkeiten in Mehrgitterverfahren zu verwenden. Eine Herangehensweise besteht darin, den Wechsel der Datentypen vertikal stattfinden zu lassen. Dies führ zu den sog. *gemischt genauen Vertikal-Verfahren.*

3.4 Gemischt genaue Vertikal-Mehrgitterverfahren

Im Gegensatz zu den Horizontal-Verfahren, bei denen gesamte Mehrgitterzyklen in unterschiedlichen Datentypen durchgeführt werden, setzt sich bei den Vertikal-Verfahren bereits ein Zyklus aus unterschiedlichen Datentypen zusammen. Hierbei wird auf dem feinsten Level in doppelter Genauigkeit begonnen und ab einer definierten Grenze bei der Vergröberung der Datentyp gewechselt. In Abbildung 17 ist diese Denkweise schematisch dargestellt. Da der Wech-

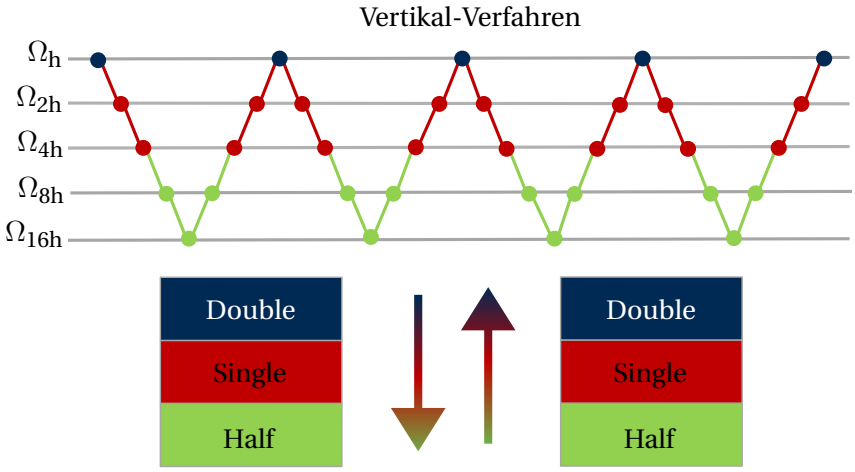


Abbildung 17: Visualisierung eines Vertikal-Verfahrens mit V-Zyklen. Der Wechsel der Datentypen findet vertikal gesehen statt.

sel der Datentypen bei dieser Herangehensweise vertikal stattfindet, werden die entstehenden Verfahren als gemischt genaue Vertikal-Mehrgitterverfahren bezeichnet. Für die betrachteten Datentypen Double, Single und Half entstehen dadurch wieder zwei unterschiedliche Verfahren. Analog zu der Notation bei den Horizontal-Verfahren wird im Folgenden das Vertikal-Verfahren mit den Datentypen Double und Single als *Vertikales-Zweischicht-Mehrgitterverfahren* (V2SMG-Verfahren) bezeichnet.

Vertikales-Zweischicht-Mehrgitterverfahren (V2SMG-Verfahren)

Das V2SMG-Verfahren ist ebenfalls iterativ aufgebaut, wobei für das Lösen der Residuumsgleichung rekursiv ein Mehrgitterverfahren in einer niedrigeren Genauigkeit verwendet wird. Im Folgenden steht k_s für die Tiefe, bei der der Wechsel von Double nach Single stattfindet. Hierbei wird das Residuum in doppelter Genauigkeit mit dem gemischt genauen RESTRICT-Kernel in einfache Genauigkeit konvertiert. Die erhaltene Approximation der Residuumsgleichung wird dann mit der gemischt genauen Prolongation wieder nach Double konvertiert. Das V2SMG-Verfahren ist in Algorithmus 10 aufgeführt. Der Befehl ${}^{ds}\mathbf{GGKV2SMG}(k, {}^{d}\mathbf{v}_k^{(m+1/3)}, {}^{d}\mathbf{f}_k)$ steht hierbei für die Anwendung der Grobgitterkorrektur des V2SMG-Verfahrens.

Im Gegensatz zu den Horizontal-Verfahren müssen beim V2SMG-Verfahren die Startlösung und die rechte Seite nicht konvertiert werden. Dies liegt daran, dass das größte Level stets in doppelter Genauigkeit gerechnet wird. Dadurch ist keinerlei Mehraufwand bei diesem Verfahren notwendig. Die Grobgitterkorrektur des V2SMG-Verfahrens ist in Algorithmus 11 aufgeführt. Die Konvergenzkontrolle kann wie bei den Horizontal-Verfahren direkt nach der Berechnung des Residuums auf dem größten Level erfolgen. Sofern das Level zum Wechseln der Datentypen noch nicht erreicht ist ($k > k_s$), wird zum Lösen der Residuumsgleichung das V2SMG-Verfahren rekursiv aufgerufen. Hierbei wurde angenommen, dass das kleinste Level echt kleiner als das Level zum Wechseln der Datentypen ist, d.h. $k_{min} < k_s$. Sobald das Level k_s erreicht

Algorithmus 10 V2SMG-Verfahren

Berechnung von ${}^{d}\mathbf{v}_k^{(m+1)} = {}^{ds}\mathbf{V2SMG}(k, {}^{d}\mathbf{v}_k^{(m)}, {}^{d}\mathbf{f}_k)$:

Vorglättung:

Berechne ${}^{d}\mathbf{v}_k^{(m+\frac{1}{3})}$ durch k_{vor}-malige Anwendung des gedämpften Jacobi-Verfahrens in Double: ${}^{d}\mathbf{v}_k^{(m+\frac{1}{3})} = {}^{d}\mathbf{JACOBI}^{k_{vor}}({}^{d}\mathbf{v}_k^{(m)}, {}^{d}\mathbf{f}_k)$

Grobgitterkorrektur:

Berechne ${}^{d}\mathbf{v}_k^{(m+\frac{2}{3})}$ durch Anwendung der Grobgitterkorrektur des V2SMG-Verfahrens: ${}^{d}\mathbf{v}_k^{(m+\frac{2}{3})} = {}^{ds}\mathbf{GGKV2SMG}(k, {}^{d}\mathbf{v}_k^{(m+\frac{1}{3})}, {}^{d}\mathbf{f}_k)$

Nachglättung:

Berechne ${}^{d}\mathbf{v}_k^{(m+1)}$ durch k_{nach}-malige Anwendung des gedämpften Jacobi-Verfahrens in Double: ${}^{d}\mathbf{v}_k^{(m+1)} = {}^{d}\mathbf{JACOBI}^{k_{nach}}({}^{d}\mathbf{v}_k^{(m+\frac{2}{3})}, {}^{d}\mathbf{f}_k)$

ist, wird der gemischt genaue RESTRICT-Kernel verwendet, um auf den Datentyp Single zu wechseln. Anschließend wird zum Lösen der Residuumsgleichung das Mehrgitterverfahren in einfacher Genauigkeit verwendet. Die erhaltene Lösung wird dann mit der gemischt genauen Prolongation wieder nach Double konvertiert.

Algorithmus 11 Grobgitterkorrektur beim V2SMG-Verfahren

Berechnung von ${}^{d}\mathbf{v}_k^{(m+\frac{2}{3})} = {}^{ds}\mathbf{GGKV2SMG}({}^{d}\mathbf{v}_k^{(m+\frac{1}{3})}, {}^{d}\mathbf{f}_k)$:

Berechne das Residuum in Double: ${}^{d}\mathbf{r}_k = {}^{d}\mathbf{DEFECT}({}^{d}\mathbf{v}_k^{(m+\frac{1}{3})}, {}^{d}\mathbf{f}_k)$

Falls $k = k_{max}$:

Berechne die Norm des Residuums in Double: $||{}^{d}\mathbf{r}_k|| = {}^{d}\mathbf{NORM}({}^{d}\mathbf{r}_k)$

Führe eine Konvergenzkontrolle durch

Falls $k > k_s$:

Restringiere das Residuum in Double: ${}^{d}\hat{\mathbf{r}}_{k-1} = {}^{d}\mathbf{RESTRICT}({}^{d}\mathbf{r}_k)$

Approximiere die Lösung der Residuumsgleichung durch die Anwendung des V2SMG-Verfahrens: ${}^{d}\hat{\mathbf{e}}_{k-1} = {}^{ds}\mathbf{V2SMG}(k-1, {}^{d}\mathbf{0}_{k-1}, {}^{d}\hat{\mathbf{r}}_{k-1})$

Prolongiere die Korrektur in Double: ${}^{d}\hat{\mathbf{e}}_k = {}^{d}\mathbf{PROLONGATE}({}^{d}\hat{\mathbf{e}}_{k-1})$

Falls $k = k_s$:

Restringiere das Residuum nach Single: ${}^{s}\hat{\mathbf{r}}_{k-1} = {}^{ds}\mathbf{RESTRICT}({}^{d}\mathbf{r}_k)$

Approximiere die Lösung der Residuumsgleichung durch die Anwendung des Mehrgitterverfahrens: ${}^{s}\hat{\mathbf{e}}_{k-1} = {}^{s}\mathbf{MG}(k-1, {}^{s}\mathbf{0}_{k-1}, {}^{s}\hat{\mathbf{r}}_{k-1})$

Prolongiere die Korrektur nach Double: ${}^{d}\hat{\mathbf{e}}_k = {}^{ds}\mathbf{PROLONGATE}({}^{s}\hat{\mathbf{e}}_{k-1})$

Korrigiere die Approximation in Double: ${}^{d}\mathbf{v}_k^{(m+\frac{2}{3})} = {}^{d}\mathbf{UPDATE}({}^{d}\mathbf{v}_k^{(m+\frac{1}{2})}, {}^{d}\hat{\mathbf{e}}_k)$

Theoretischer Speed-Up

Für die Berechnung des theoretischen Speed-Ups werden im Folgenden erneut V-Zyklen betrachtet. Aus Kapitel 3.3 ist bekannt, dass der Aufwand pro Level bei einem V-Zyklus gerade

$$ {}^{d}T_{vzl}(n) = 210 \cdot n^2 + 436 \cdot n + 410 $$

für die Verwendung von Double beträgt. Bei Verwendung von einfacher Genauigkeit gilt

$$ \begin{aligned} {}^{s}T_{vzl}(n) &= 6 \cdot {}^{s}MEM_{jac}(n) + {}^{s}MEM_{def}(n) + {}^{s}MEM_{res}(n) + {}^{s}MEM_{pro}(n) + {}^{s}MEM_{upd}(n) \\ &= 105 \cdot n^2 + 218 \cdot n + 205, \end{aligned} $$

was genau der Hälfte des Aufwands von Double entspricht, da im Gegensatz zum H2SMG-Verfahren nicht der gemischt genaue DEFECT- und RESTRICT-Kernel verwendet wird. Der Aufwand für das k_s-te Level muss separat berechnet werden, da hier die gemischt genaue Restriktion und Prolongation verwendet werden. Deshalb gilt

$$\begin{aligned}{}^{ds}T_{vzl}(n_s) &= 6 \cdot {}^{s}MEM_{jac}(n_s) + {}^{s}MEM_{def}(n_s) + {}^{ds}MEM_{res}(n_s) + {}^{ds}MEM_{pro}(n_s) + {}^{s}MEM_{upd}(n_s) \\ &= 112 \cdot n^2 + 224 \cdot n + 198,\end{aligned}$$

wobei die Verwendung der beiden Genauigkeiten auch durch das Superskript *ds* gekennzeichnet ist. Der Aufwand für die Durchführung eines V-Zyklus des V2SMG-Verfahrens berechnet sich dadurch zu

$$\begin{aligned}{}^{ds}T_{vz}(n_{max}, n_{min}, n_s) = &\sum_{k=k_s+1}^{k_{max}} {}^{d}T_{vzl}(2^k-1) + {}^{ds}T_{vzl}(n_s) + \sum_{k=k_{min}+1}^{k_s-1} {}^{s}T_{vzl}(2^k-1) + 151 \cdot {}^{s}MEM_{jac}(n_{min}) \\ &= 280 \cdot n_{max}^2 + 592 \cdot n_{max} + 1672 \cdot n_{min}^2 + 4536 \cdot n_{min} - 133 \cdot n_s^2 - 290 \cdot n_s \\ &+ 184 \cdot \log_2\Big(n_{max}+1\Big) - 92 \cdot \log_2\Big((n_{min}+1)(n_s+1)\Big) + 4825.\end{aligned}$$

Der theoretische Speed-Up berechnet sich als Quotient zwischen dem Aufwand des herkömmlichen Mehrgitterverfahrens und dem Aufwand des V2SMG-Verfahrens, d.h.

$$\text{Speed}-\text{Up} = \frac{{}^{d}T_{mg}}{{}^{ds}T_{v2smg}} = \frac{{}^{d}T_{vz}}{{}^{ds}T_{vz}}.$$

In Tabelle 7 ist der Speed-Up für unterschiedliche Problemgrößen und Möglichkeiten von n_s aufgetragen. Wie bereits beim H2SMG-Verfahren liegt der Speed-Up unter 2,00. Dies ist eben-

k_{max}	k_s	Theo. Speed-Up
8	8	1,91
	7	1,14
	6	1,03
12	12	1,90
	11	1,13
	10	1,03
	9	1,00

Tabelle 7: Theoretischer Speed-Up des V2SMG-Verfahrens in Abhängigkeit des Levels k_s und der Problemgröße k_{max}.

falls klar, da auch beim V2SMG-Verfahren stets ein Teil in doppelter Genauigkeit gerechnet wird. Mit 1,91 bei Level 8 bzw. 1,90 bei Level 12 liegen die theoretischen Werte leicht über denen des H2SMG-Verfahrens. Für eine kleinere Wahl von n_s fällt der Speed-Up stark ab. Findet der Wechsel der Datentypen bei einer Problemgröße von $k_{max} = 12$ bei Level 11 statt, so lässt sich lediglich ein Speed-Up von 1,13 erzielen. Dies zeigt erneut die Dominanz des größten Levels: Sobald dieses vollständig in doppelter Genauigkeit gerechnet wird, ist der Aufwand des gesamten Verfahrens maßgeblich davon abhängig. Der zeitliche Vorteil, den man bei der Verwendung von einfacher Genauigkeit für die kleineren Level erzielt, wird dadurch vernachlässigbar.

Wird neben doppelter und einfach auch mit halber Genauigkeit gearbeitet, so lässt sich aus dem V2SMG-Verfahren das dreischichtige Vertikal-Verfahren herleiten.

Vertikales-Dreischicht-Mehrgitterverfahren (V3SMG-Verfahren)

Beim V3SMG-Verfahren werden die untersten Level in halber Genauigkeit gerechnet. Hierzu wird neben k_s der Parameter k_h eingeführt, welcher den Wechsel von einfacher zu halber Genauigkeit charakterisiert. Im Folgenden wird angenommen, dass $k_s > k_h > k_{min}$ gilt. Das bedeutet, dass es stets Level gibt, die in einfacher Genauigkeit gerechnet werden. Es findet also kein direkter Wechsel von Double zu Half stattfindet ($k_s > k_h$). Zusätzlich bedeutet dies, dass es Level gibt, die in halber Genauigkeit gerechnet werden ($k_h > k_{min}$). Das V3SMG-Verfahren ist in Algorithmus 12 aufgeführt. Der einzige Unterschied zum V2SMG-Verfahren besteht in

Algorithmus 12 V3SMG-Verfahren

Berechnung von ${}^{d}\mathbf{v}_k^{(m+1)} = {}^{dsh}\mathbf{V3SMG}(k, k_s, k_h, {}^{d}\mathbf{v}_k^{(m)}, {}^{d}\mathbf{f}_k)$:
- Vorglättung:
  - Berechne ${}^{d}\mathbf{v}_k^{(m+\frac{1}{3})}$ durch k_{vor}-malige Anwendung des gedämpften Jacobi-Verfahrens in Double: ${}^{d}\mathbf{v}_k^{(m+\frac{1}{3})} = {}^{d}\mathbf{JACOBI}^{k_{vor}}({}^{d}\mathbf{v}_k^{(m)}, {}^{d}\mathbf{f}_k)$
- Grobgitterkorrektur:
  - Berechne ${}^{d}\mathbf{v}_k^{(m+\frac{2}{3})}$ durch Anwendung der Grobgitterkorrektur des V3SMG-Verfahrens: ${}^{d}\mathbf{v}_k^{(m+\frac{2}{3})} = {}^{dsh}\mathbf{GGKV3SMG}(k, k_s, k_h, {}^{d}\mathbf{v}_k^{(m+\frac{1}{3})}, {}^{d}\mathbf{f}_k)$
- Nachglättung:
  - Berechne ${}^{d}\mathbf{v}_k^{(m+1)}$ durch k_{nach}-malige Anwendung des gedämpften Jacobi-Verfahrens in Double: ${}^{d}\mathbf{v}_k^{(m+1)} = {}^{d}\mathbf{JACOBI}^{k_{nach}}({}^{d}\mathbf{v}_k^{(m+\frac{2}{3})}, {}^{d}\mathbf{f}_k)$

der Grobgitterkorrektur. Da das größte Level stets in doppelter Genauigkeit durchgeführt wird, wird für die Vor- und Nachglättungsschritte das ungedämpfte Jacobi-Verfahren in doppelter Genauigkeit verwendet. Der Befehl ${}^{dsh}\mathbf{GGKV3SMG}(k, k_s, k_h, {}^{d}\mathbf{v}_k^{(m+1/3)}, {}^{d}\mathbf{f}_k)$ beschreibt den Aufruf der Grobgitterkorrektur des V3SMG-Verfahrens. Das Superskript *dsh* charakterisiert die drei verwendeten Datentypen Double, Single und Half. Die Grobgitterkorrektur ist in Algorithmus 13 aufgeführt. Die Berechnung der Norm und die Durchführung einer Konvergenzkontrolle

Algorithmus 13 Grobgitterkorrektur beim V3SMG-Verfahren

Berechnung von ${}^{d}\mathbf{v}_k^{(m+\frac{2}{3})} = {}^{dsh}\mathbf{GGKV3SMG}(k, k_s, k_h, {}^{d}\mathbf{v}_k^{(m+\frac{1}{3})}, {}^{d}\mathbf{f}_k)$:
- Berechne das Residuum in Double: ${}^{d}\mathbf{r}_k = {}^{d}\mathbf{DEFECT}({}^{d}\mathbf{v}_k^{(m+\frac{1}{3})}, {}^{d}\mathbf{f}_k)$
- Falls $k = k_{max}$:
  - Berechne die Norm des Residuums in Double: $||{}^{d}\mathbf{r}_k|| = {}^{d}\mathbf{NORM}({}^{d}\mathbf{r}_k)$
  - Führe eine Konvergenzkontrolle durch
- Falls $k > k_s$:
  - Restringiere das Residuum in Double: ${}^{d}\hat{\mathbf{r}}_{k-1} = {}^{d}\mathbf{RESTRICT}({}^{d}\mathbf{r}_k)$
  - Approximiere die Lösung der Residuumsgleichung durch die Anwendung des V3SMG-Verfahrens: ${}^{d}\hat{\mathbf{e}}_{k-1} = {}^{dsh}\mathbf{V3SMG}(k-1, k_s, k_h, {}^{d}\mathbf{0}_{k-1}, {}^{d}\hat{\mathbf{r}}_{k-1})$
  - Prolongiere die Korrektur in Double: ${}^{d}\hat{\mathbf{e}}_k = {}^{d}\mathbf{PROLONGATE}({}^{d}\hat{\mathbf{e}}_{k-1})$
- Falls $k = k_s$:
  - Restringiere das Residuum nach Single: ${}^{s}\hat{\mathbf{r}}_{k-1} = {}^{ds}\mathbf{RESTRICT}({}^{d}\mathbf{r}_k)$
  - Approximiere die Lösung der Residuumsgleichung durch die Anwendung des V2SMG-Verfahrens: ${}^{s}\hat{\mathbf{e}}_{k-1} = {}^{sh}\mathbf{V2SMG}(k-1, k_h, {}^{s}\mathbf{0}_{k-1}, {}^{s}\hat{\mathbf{r}}_{k-1})$
  - Prolongiere die Korrektur nach Double: ${}^{d}\hat{\mathbf{e}}_k = {}^{ds}\mathbf{PROLONGATE}({}^{s}\hat{\mathbf{e}}_{k-1})$
- Korrigiere die Approximation in Double: ${}^{d}\mathbf{v}_k^{(m+\frac{2}{3})} = {}^{d}\mathbf{UPDATE}({}^{d}\mathbf{v}_k^{(m+\frac{1}{2})}, {}^{d}\hat{\mathbf{e}}_k)$

findet wieder auf dem größten Level, nach der Residuumsberechnung statt. Sofern das aktu-

elle Level noch größer als das Level für den Wechsel auf Single ist ($k > k_s$), wird rekursiv das V3SMG-Verfahren aufgerufen. Im Fall von $k = k_s$ wird zum Lösen der Residuumsgleichung das V2SMG-Verfahren mit den Datentypen Single und Half verwendet (vgl. Algorithmus 10).

Theoretischer Speed-Up

Analog wie bei der Analyse des V2SMG-Verfahrens beträgt der Aufwand pro Level bei Verwendung von halber Genauigkeit

$$^{h}T_{vzl}(n) = \frac{105}{2} \cdot n^2 + 109 \cdot n + 205.$$

Dies entspricht gerade der Hälfte des Aufwands bei Verwendung von Single. Auf dem n_h-ten Level werden die gemischt genauen RESTRICT- und PROLONGATE-Kernel verwendet. Dadurch berechnet sich der Aufwand für dieses Level zu

$$\begin{aligned} ^{sh}T_{vzl}(n_h) &= 6 \cdot {}^{s}MEM_{jac}(n_h) + {}^{s}MEM_{def}(n_h) + {}^{sh}MEM_{res}(n_h) + {}^{sh}MEM_{pro}(n_h) + {}^{s}MEM_{upd}(n_h) \\ &= 104 \cdot n^2 + 216 \cdot n + 200. \end{aligned}$$

Der Aufwand für das V3SMG-Verfahren setzt sich aus dem Aufwand der jeweiligen Level zusammen und beläuft sich dadurch auf

$$\begin{aligned} ^{dsh}T_{vz}(n_{max}, n_{min}, n_s, n_h) = &\sum_{k=k_s+1}^{k_{max}} {}^{d}T_{vzl}(2^k - 1) + {}^{ds}T_{vzl}(n_s) \\ &+ \sum_{k=k_h+1}^{k_s-1} {}^{s}T_{vzl}(2^k - 1) + {}^{sh}T_{vzl}(n_h) \\ &+ \sum_{k=k_{min}+1}^{k_h-1} {}^{h}T_{vzl}(2^k - 1) + 151 \cdot {}^{h}MEM_{jac}(n_{min}) \\ &= 280 \cdot n_{max}^2 + \frac{1841}{2} \cdot n_{max} + 836 \cdot n_{min}^2 + 2268 \cdot n_{min} \\ &- 133 \cdot n_s^2 - 290 \cdot n_s - \frac{109}{2} n_h^2 - 137 \cdot n_h \\ &+ 184 \cdot \log_2\Big((n_{max} + 1)\Big) - 92 \cdot \log_2\Big((n_s + 1)\Big) - 46 \cdot \Big((n_{min} + 1)(n_h + 1)\Big) \\ &+ \frac{4821}{2}. \end{aligned}$$

Der Speed-Up gegenüber dem herkömmlichen Mehrgitterverfahren in doppelter Genauigkeit berechnet sich damit zu

$$\text{Speed} - \text{Up} = \frac{^{d}T_{mg}}{^{dsh}T_{v3smg}} = \frac{^{d}T_{vz}}{^{dsh}T_{vz}}.$$

In Tabelle 8 ist der theoretische Speed-Up für unterschiedliche Möglichkeiten von k_s und k_h aufgetragen. Es zeigt sich, dass der beste Speed-Up auf Level 8 erreicht wird. Mit knapp 2,27 liegt er damit erwartungsgemäß über dem Speed-Up des V2SMG-Verfahrens für dasselbe Level. Allerdings beträgt der zusätzliche Gewinn gerade mal knapp 18 Prozent. Auf Grund der Unterschiede des Speicherplatzes hätte auch einen Gewinn in der Nähe von 50 Prozent erwartet werden können. Da der Datentyp Half jedoch erst auf den gröberen Gittern zur Anwendung kommt und das feinste Gitter den Aufwand dominiert, trägt seine Verwendung nur geringfügig zum Speed-Up bei. Je kleiner k_h gewählt wird, desto geringer fällt der theoretische Speed-Up aus. Mit steigender Problemgröße zeigt sich auch beim V3SMG-Verfahren dasselbe Resultat wie bereits bei den anderen gemischt genauen Verfahren: Durch die Dominanz des feinsten Levels

k_{max}	k_s	k_h	Theo. Speed-Up
8	8	7	2,27
	8	6	2,13
	7	6	1,30
12	12	11	2,10
	12	10	1,95
	11	10	1,15
	11	9	1,14

Tabelle 8: Theoretischer Speed-Up des V3SMG-Verfahrens in Abhängigkeit der Level k_s und k_h sowie der Problemgröße k_{max}.

sinkt der Wert mit steigender Problemgröße. Insgesamt zeigt sich ein Speed-Up zwischen 14 und 127 Prozent, der stark von der Wahl der Level k_s und k_h abhängt. Hierauf wird insbesondere bei der numerischen Untersuchung des V2SMG- und V3SMG-Verfahrens in Kapitel 4.5 eingegangen.

Nachdem die vier gemischt genauen Mehrgitterverfahren theoretisch hergeleitet und der zu erwartende Speed-Up untersucht wurde, steht als nächstes die numerische Untersuchung der Verfahren im Fokus.

4 Numerische Experimente und Diskussion

Nachdem im Kapitel zuvor der zu erwartenden Speed-Up hergeleitet wurde, liegt der Fokus im Folgenden auf der numerischen Untersuchung der Implementierung und der Analyse der Ausführungszeiten. Hierzu werden zunächst die Messmetriken erläutert, welche im Zuge dieser Arbeit verwendet werden. Anschließend erfolgt die Untersuchung der Kernel und des Mehrgitterverfahrens mit nur einem Datentyp, die als Ausgangspunkt für die weitere Analyse der gemischt genauen Verfahren dient.

4.1 Messmetriken

Im Zuge dieser Arbeit werden prinzipiell zwei Aspekte der Verfahren verglichen: die Fehlerentwicklung und die Ausführungszeit. Hierfür bedarf es je einer geeigneter Metrik. In der Praxis wird für die Fehlermessung häufig das sog. *relative Residuum* (RR) verwendet. Hierbei wird die Norm des Residuums $\mathbf{r}^{(m)}$ im m-ten Schritt durch die Norm der rechten Seite $\mathbf{f}$ geteilt:

$$\mathrm{RR(m)} = \frac{||\mathbf{r}^{(\mathrm{m})}||}{||\mathbf{f}||} = \frac{||\mathbf{f}-\mathbf{A}\mathbf{v}^{(\mathrm{m})}||}{||\mathbf{f}||}.$$

Da in vielen Verfahren die Berechnung des Residuums bereits Bestandteil des Algorithmus ist, liefert die Berechnung des relativen Residuums keinen nennenswerten Mehraufwand.

Für theoretische Untersuchungen und numerische Experimente kann auch der *relative Fehler* (RF) verwendet werden. Während das Residuum Aufschluss darüber gibt, wie gut eine Approximation $\mathbf{v}^{(m)}$ das Gleichungssystem löst, ist der relative Fehler ein Maß für die Abweichung der Approximation von der exakten Lösung. Der RF im m-ten Schritt berechnet sich zu

$$\mathrm{RF(m)} = \frac{||\mathbf{v}-\mathbf{v}^{(\mathrm{m})}||}{||\mathbf{v}||}.$$

Da in der Praxis die exakte Lösung $\mathbf{v}$ unbekannt ist, kann der RF nicht als Abbruchkriterium verwendet werden. Im Zuge dieser Arbeit wird die exakte Lösung für ein Problem vorgegeben und die daraus resultierende rechte Seite analytisch bestimmt. Damit kann der RF für numerische Experimente verwendet werden. Zusätzlich lässt sich mit dem RF als Fehlermaß ein Zusammenhang mit dem Diskretisierungsfehler herstellen, welcher in Kapitel 4.3 hergeleitet wird.

Für die implementierten Verfahren werden im Folgenden sowohl der Verlauf des relativen Residuums als auch der Verlauf des relativen Fehlers diskutiert. Letzteres erlaubt die Beurteilung der Qualität eines Verfahrens unabhängig vom verwendeten Abbruchkriterium. Das relative Residuum wird dabei exemplarisch als ein mögliches Abbruchkriterium betrachtet.

Um den Speed-Up der Verfahren zu bestimmen, wird auf Zeitmessungen zurückgegriffen. Hierbei kann zwischen der Ausführungszeit eines einzelnen Kernels, einzelner Mehrgitterzyklen und der Ausführungszeit eines gesamten Verfahrens unterschieden werden. Letzteres ist abhängig von der rechten Seite, den gewählten Datentypen, der Abbruchbedingung und der Problemgröße. Die Zeitmessungen einzelner Zyklen und Kernel sind unabhängig von der rechten Seite und einer Abbruchbedingung. Daher sind sie besser für einen allgemeinen Vergleich der Verwendung unterschiedlicher Datentypen geeignet. Da durch die Verwendung gemischter Datentypen ein Mehraufwand in Form von Kopier- und Konvertierungsoperationen entsteht, werden im Zuge dieser Arbeit auch Ausführungszeiten der gesamten Verfahren betrachtet. Als exakte Lösung wird hierfür die trigonometrische Funktion

$$\mathrm{v(x,y)} \;=\; \mathrm{TRIGO(x,y)} \;=\; 5\cdot\sin(\pi\cdot\mathrm{x})\cdot\sin(\pi\cdot\mathrm{y})$$

vorgegeben. Die rechte Seite folgt dann durch zweimaliges, analytisches Ableiten nach x und y:

$$f(x,y) = -\Delta \mathrm{TRIGO}(x,y) = 10 \cdot \pi^2 \cdot \sin(\pi \cdot x) \cdot \sin(\pi \cdot y)$$

In Anhang C befindet sich eine kurze Untersuchung der Verfahren mit einer unstetigen rechten Seite. Im Folgenden wird bei allen Untersuchungen *TRIGO*(x, y) als analytische Lösung verwendet. In Abbildung 18 ist sie in Form eines Surfplots dargestellt.

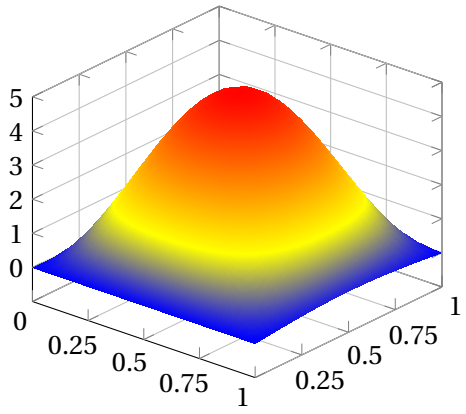


Abbildung 18: 3D-Darstellung der verwendeten analytischen Lösung TRIGO(x,y).

4.2 Numerische Aufwandsanalyse der Kernel

Als Nächstes werden die Rechen- und Datendurchsätze der implementierten Kernel analysiert und diskutiert. Da es sich beim JACOBI-, DEFECT-, und RESTRICT-Kernel jeweils um die Anwendung eines Stencils handelt, wird im Folgenden lediglich der JACOBI-Kernel genauer untersucht. Die Ergebnisse lassen sich jedoch auf die anderen beiden Kernel übertragen. Die gemessenen Durchsätze sind in Abbildung 19 aufgeführt.

Es zeigt sich, dass die Kurven der Rechendurchsätze streng nach den Datentypen geordnet sind. Dies gilt für alle betrachteten Kernel, wobei die Half-Implementierung stets den größten und die Double-Implementierung stets den kleinsten Rechendurchsatz aufweist. Außerdem ist auffällig, dass sich die Kurven erst ab einem mittleren Level voneinander unterscheiden. Für die Datendurchsätze ist die Situation umgekehrt. Hier verlaufen die Kurven mit doppelter Genauigkeit stets oberhalb der Single- und Half-Varianten. Zusätzlich beginnen sie getrennt voneinander und nähern sich einem gemeinsamen Grenzwert an.

Die Implementierung des JACOBI-Kernels in halber Genauigkeit erreicht einen Rechendurchsatz von knapp 1.147 GFLOP/s. Mit rund 679 GFLOP/s erreicht die Single-Implementierung etwas mehr als die Hälfte. Bei Verwendung von doppelter Genauigkeit halbiert sich der Rechenaufwand erneut auf ca. 387 GFLOP/s. An dieser Stelle sei erwähnt, dass alle Spitzenwerte der Rechendurchsätze weniger als ein Prozent des theoretischen Maximalwerts entsprechen. Erklären lässt sich dies anhand der Datendurchsätze. Mit steigendem Level unterscheidet sich der gemessene Wert für die Double-Implementierung kaum von den beiden anderen. Der Speichercontroller ist für diese Problemgrößen bereits voll ausgelastet und kann bei Verwendung von Single in gleicher Zeit lediglich die doppelte Menge an Daten transferieren. Dies führt zu dem Faktor 2 beim zugehörigen Rechendurchsatz. Dasselbe gilt für den Vergleich zwischen Single und Half. Der limitierende Faktor des Jacobi-Verfahrens ist demnach der Speicherzugriff, was die Anwendung eines solchen Stencils zu einem speichergebundenen Kernel macht. Aus diesem Grund sind keine Rechendurchsätze in der Nähe des theoretischen Maximums zu erwarten. Mit ca. 715 GByte/s erreicht die Double-Variante knapp 80 Prozent des theoretischen

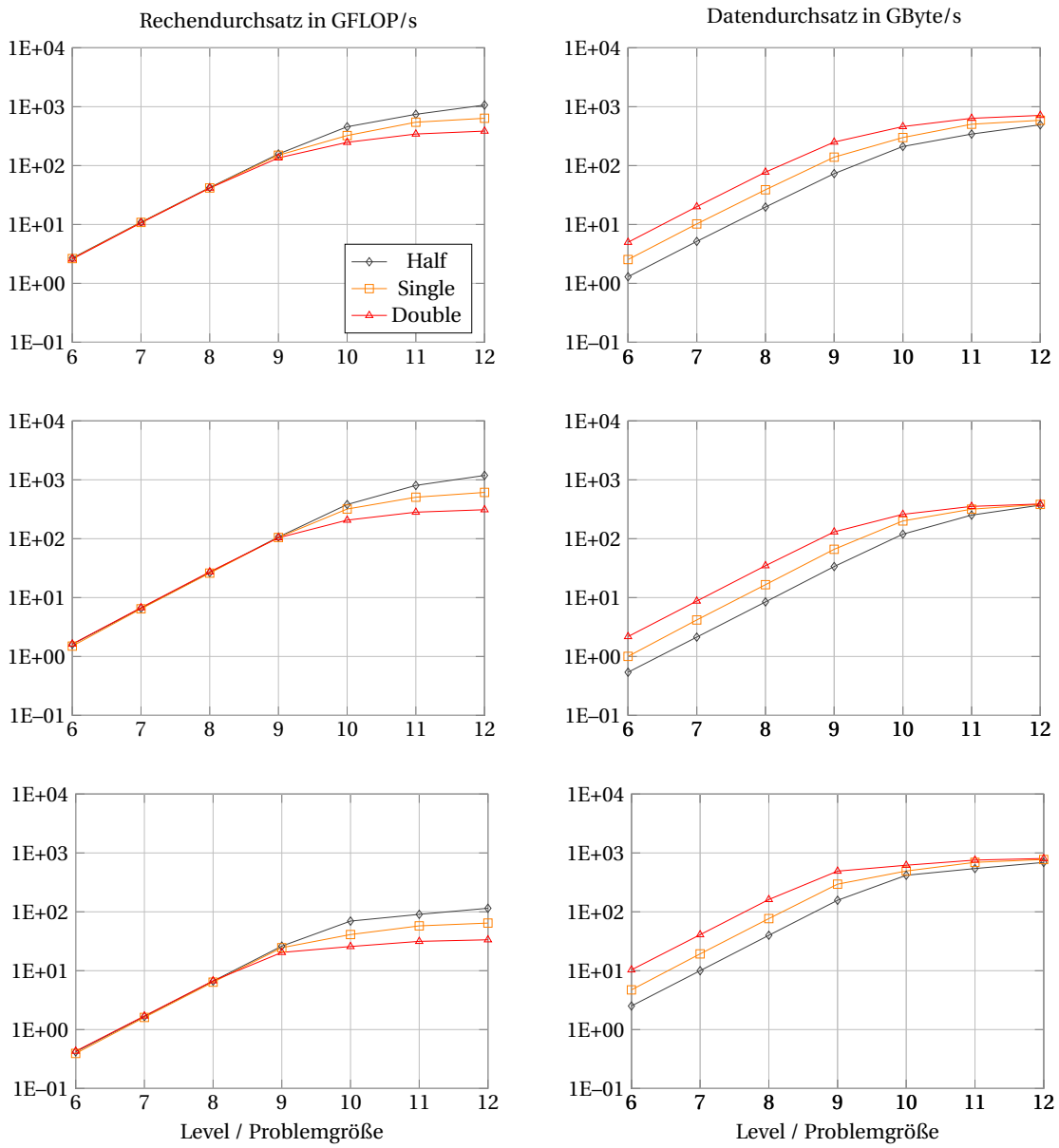


Abbildung 19: Rechendurchsätze (links) und Datendurchsätze (rechts) der implementierten Kernel. Von oben nach unten: JACOBI-, PROLONGATE- und ADD-Kernel. Die gemessenen Werte sind abhängig von der Problemgröße aufgetragen.

Maximalwerts von 900 GByte/s.

Die gemessenen Werte für die Interpolation können analog interpretiert werden. Auch hier lässt sich ein Verhältnis von 1:2 zwischen dem Double- und Single- bzw. zwischen dem Single- und Half-Rechendurchsatz messen. Die Spitzenwerte liegen dabei in derselben Größenordnung wie beim JACOBI-Kernel. Der Datendurchsatz ist jedoch deutlich geringer. Den besten Wert erzielte die Half-Implementierung mit knapp 400 GByte/s, was rund 44 Prozent des theoretischen Maximums entspricht.

Die Double-Implementierung des ADD-Kernels liefert mit 33 GFLOP/s den geringsten Rechendurchsatz für das größte Level. Das Verhältnis zu den anderen Datentypen beträgt dabei ebenfalls 1:2. Der Datendurchsatz für die Double-Implementierung liegt mit knapp 805 GByte/s an erster Stelle und entspricht fast 90 Prozent des theoretischen Maximalwerts. Dies zeigt, dass

auch der ADD-Kernel speichergebunden ist.

Nachdem die Implementierung der einzelnen Kernel analysiert wurde, erfolgt als Nächstes die Betrachtung eines gesamten Mehrgitterverfahrens.

4.3 Analyse des Mehrgitterverfahrens mit einem Datentyp

Im Folgenden wird zunächst ein Mehrgitterverfahren mit nur einem Datentyp betrachtet. Die Implementierung in doppelter Genauigkeit wird später für den Vergleich mit den gemischt genauen Verfahren herangezogen. Die Single- und Half-Implementierung stellt hingegen den Ausgangspunkt der Analyse von gemischt genauen Mehrgitterverfahren dar. Wir beginnen mit der Analyse des Mehrgitterverfahrens in doppelter Genauigkeit.

Mehrgitterverfahren in doppelter Genauigkeit

In Abbildung 20 sind die Fehlerentwicklungen für ein Mehrgitterverfahren in doppelter Genauigkeit dargestellt. Betrachtet man das relative Residuum, so fällt auf, dass man für alle Problemgrößen einen monoton fallenden Verlauf erhält. In den ersten 30 Iterationen ist der Verlauf für alle betrachteten Problemgrößen identisch. Dies zeigt die h-unabhängige Konvergenz des Mehrgitterverfahrens. Danach teilen sich die Kurven auf, wobei das größte Level als Erstes zu stagnieren beginnt und gegen einen Wert von knapp $3,49\text{E}-10$ konvergiert. Mit sinkender Problemgröße fällt die Größe des RR weiter ab. Auf 6 sechs beträgt das kleinste gemessene Residuum knapp $8,00\text{E}-14$. Erklären lässt sich dieses Verhalten anhand der Problemgröße: Bei Verwendung von doppelter Genauigkeit beträgt der Rundungsfehler einer arithmetischen Rechenoperation maximal $\varepsilon_d = 1,1\text{E}{-16}$. Für die Addition zweier Vektoren auf Level 12 erhält man damit $(2^{12}+1)^2 \cdot \varepsilon_d \approx 1,85\text{E}{-09}$ als akkumulierten Rundungsfehler. Damit liegt der Fehler bereits in der zu messenden Größenordnung. Analog erhält man für die Addition auf Level 6 einen akkumulierten Fehler von $(2^6+1)^2 \cdot \varepsilon_d \approx 4,65\text{E}-13$.

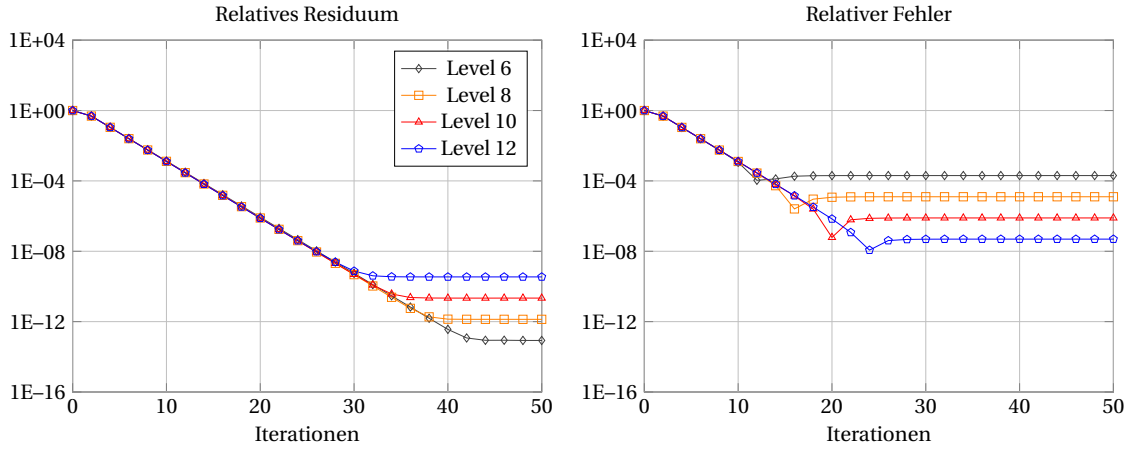


Abbildung 20: Fehlerentwicklungen des Mehrgitterverfahrens in doppelter Genauigkeit. Das relative Residuum (links) und der relative Fehler (rechts) sind für unterschiedliche Problemgrößen in Abhängigkeit der durchlaufenden Iterationen aufgetragen.

Betrachtet man den relativen Fehler, so zeigt sich, dass sich die Werte ebenfalls asymptotisch gegen einen Grenzwert annähern. Die Kurven stagnieren deutlich früher, als es beim relativen Residuum der Fall ist. Mit steigender Problemgröße sinkt der Grenzwert. Für das kleinste betrachtete Problem erhält man einen Grenzwert von knapp $2,00\text{E}-04$. Bei Level 12 beträgt er rund $4,90\text{E}{-08}$. Auf Grund der Diskretisierung mit der Finite-Differenzen-Methode erhält man

einen *Diskretisierungsfehler* (DF). Je feiner das Gebiet diskretisiert wird, desto geringer ist auch der Diskretisierungsfehler, weshalb der Grenzwert mit Zunahme der Problemgröße sinkt. Formal kann der relative Fehler wie folgt abgeschätzt werden:

Sei $\tilde{\mathbf{v}}$ die exakte Lösung des diskretisierten Gleichungssystems, d.h. $\mathbf{A}\tilde{\mathbf{v}} = \mathbf{f}$ und $\mathbf{v}$ die analytische Lösung des Problems. Dann folgt mit der Dreiecksungleichung

$$\mathrm{RF}(\mathrm{m}) = \frac{||\mathbf{v}-\mathbf{v}^{(\mathrm{m})}||}{||\mathbf{v}||} = \frac{||\mathbf{v}-\mathbf{v}^{(\mathrm{m})}+\tilde{\mathbf{v}}-\tilde{\mathbf{v}}||}{||\mathbf{v}||} \leq \underbrace{\frac{||\tilde{\mathbf{v}}-\mathbf{v}^{(\mathrm{m})}||}{||\mathbf{v}||}}_{\text{rel. Verfahrensfehler}} + \underbrace{\frac{||\mathbf{v}-\tilde{\mathbf{v}}||}{||\mathbf{v}||}}_{\text{rel. Diskretisierungsfehler}}. \tag{11}$$

Der hier als relativer Verfahrensfehler bezeichnete Term beinhaltet seinerseits den Abbruchfehler sowie weitere Fehler auf Grund der Kondition der Systemmatrix. Der Diskretisierungsfehler lässt sich selbst wiederum mit dem relativen Fehler abschätzen:

$$\frac{||\mathbf{v}-\tilde{\mathbf{v}}||}{||\mathbf{v}||} = \frac{||\mathbf{v}-\tilde{\mathbf{v}}+\mathbf{v}^{(\mathrm{m})}-\mathbf{v}^{(\mathrm{m})}||}{||\mathbf{v}||} \leq \frac{||\mathbf{v}^{(\mathrm{m})}-\tilde{\mathbf{v}}||}{||\mathbf{v}||} + \frac{||\mathbf{v}-\mathbf{v}^{(\mathrm{m})}||}{||\mathbf{v}||}.$$

Hieraus folgt

$$\frac{||\mathbf{v}-\tilde{\mathbf{v}}||}{||\mathbf{v}||} - \frac{||\mathbf{v}^{(\mathrm{m})}-\tilde{\mathbf{v}}||}{||\mathbf{v}||} \leq \frac{||\mathbf{v}-\mathbf{v}^{(\mathrm{m})}||}{||\mathbf{v}||} = \mathrm{RF}(\mathrm{m}). \tag{12}$$

Insgesamt folgt somit aus (11) und (12)

$$\frac{||\mathbf{v}-\tilde{\mathbf{v}}||}{||\mathbf{v}||} - \frac{||\mathbf{v}^{(\mathrm{m})}-\tilde{\mathbf{v}}||}{||\mathbf{v}||} \leq \mathrm{RF}(\mathrm{m}) \leq \frac{||\mathbf{v}-\tilde{\mathbf{v}}||}{||\mathbf{v}||} + \frac{||\tilde{\mathbf{v}}-\mathbf{v}^{(\mathrm{m})}||}{||\mathbf{v}||}. \tag{13}$$

Der Term $||\mathbf{v}^{(\mathrm{m})}-\tilde{\mathbf{v}}||$ wird in der Literatur häufig auch als *Vorwärtsfehler* (VF) bezeichnet. Sofern dieser Term klein wird, befindet sich der relative Fehler in der Größenordnung des Diskretisierungsfehlers. Um dies zu zeigen, kann der VF wie folgt abgeschätzt werden:

$$\begin{aligned}
\frac{||\tilde{\mathbf{v}}-\mathbf{v}^{(\mathrm{m})}||}{||\tilde{\mathbf{v}}||} &= \frac{||\mathbf{A}||\cdot||\mathbf{A}^{-1}||}{||\mathbf{A}||\cdot||\mathbf{A}^{-1}||} \cdot \frac{||\tilde{\mathbf{v}}-\mathbf{v}^{(\mathrm{m})}||}{||\tilde{\mathbf{v}}||} \\
&\leq \frac{||\mathbf{A}||\cdot||\mathbf{A}^{-1}||}{||\mathbf{A}^{-1}||} \cdot \frac{||\tilde{\mathbf{v}}-\mathbf{v}^{(\mathrm{m})}||}{||\mathbf{A}\tilde{\mathbf{v}}||} \\
&= \frac{||\mathbf{A}||\cdot||\mathbf{A}^{-1}||}{||\mathbf{A}^{-1}||} \cdot \frac{||\mathbf{A}^{-1}\mathbf{A}(\tilde{\mathbf{v}}-\mathbf{v}^{(\mathrm{m})})||}{||\mathbf{A}\tilde{\mathbf{v}}||} \\
&\leq ||\mathbf{A}||\cdot||\mathbf{A}^{-1}|| \cdot \frac{||\mathbf{A}\tilde{\mathbf{v}}-\mathbf{A}\mathbf{v}^{(\mathrm{m})}||}{||\mathbf{A}\tilde{\mathbf{v}}||} \\
&= \varkappa_{\mathbf{A}} \cdot \frac{||\mathbf{f}-\mathbf{A}\mathbf{v}^{(\mathrm{m})}||}{||\mathbf{f}||} \\
&= \varkappa_{\mathbf{A}} \cdot \frac{||\mathbf{r}^{(\mathrm{m})}||}{||\mathbf{f}||}.
\end{aligned}$$

Hierbei steht $\varkappa_{\mathbf{A}} = ||\mathbf{A}||\cdot||\mathbf{A}^{-1}||$ für die Kondition der Systemmatrix $\mathbf{A}$. Der Term $||\mathbf{r}^{(\mathrm{m})}||/||\mathbf{f}||$ entspricht gerade dem relativen Residuum. Damit folgt für den Vorwärtsfehler

$$||\tilde{\mathbf{v}}-\mathbf{v}^{(\mathrm{m})}|| \leq \varkappa_{\mathbf{A}} \cdot ||\tilde{\mathbf{v}}|| \cdot \mathrm{RR}(\mathrm{m}) \to 0 \ (\mathrm{m} \to \infty),$$

sofern das Problem nicht singulär ist und arithmetische Fehler vernachlässigt werden. Nach 30 Iterationen befinden sich die RR aller hier betrachteten Problemgrößen unterhalb $1{,}00\mathrm{E}-08$ und aus 13) folgt

$$\mathrm{RF}(\mathrm{m}) \approx \frac{||\mathbf{v}-\tilde{\mathbf{v}}||}{||\mathbf{v}||},$$

sofern m groß genug gewählt wird. In diesem Sinne können die Grenzwerte des relativen Fehlers aus Abbildung 20 als Diskretisierungsfehler aufgefasst werden.

In Tabelle 9 sind die Problemgrößen und zugehörigen Diskretisierungsfehler aufgelistet. Es fällt auf, dass das Produkt aus Problemgröße und Diskretisierungsfehler nahezu konstant ist. Für eine Problemgröße N und zugehörigem Diskretisierungsfehler DF(N) ergibt sich damit der folgende Zusammenhang:

$$\mathrm{N} \cdot \mathrm{DF(N)} \approx \text{konstant} \longrightarrow \mathrm{DF(N)} \propto \frac{1}{\mathrm{N}}.$$

In anderen Worten: Verdoppelt sich die Problemgröße, so halbiert sich der Diskretisierungsfehler.

Level	Problemgröße N	rel. Diskretisierungsfehler DF(N)	N · DF(N)
6	4.225	2,00E−04	0,845
8	66.049	1,25E−05	0,827
10	1.050.625	7,84E−07	0,824
12	16.785.409	4,90E−08	0,822

Tabelle 9: Auflistung des relativen Diskretisierungsfehlers in Abhängigkeit von der Problemgröße.

Betrachtet man die Kurven des relativen Fehlers für mittlere Iterationszahlen, so lässt sich je ein globales Minimum erkennen. Die Erklärung dafür liegt in der Definition des relativen Fehlers. Der Vorwärtsfehler misst den Abstand zwischen Approximation und der Lösung des Gleichungssystems im Urbildraum. Das Residuum misst denselben Abstand im Bildraum. Der relative Fehler hingegen misst den Abstand zwischen Approximation und exakter Lösung. Deshalb steigt der RF nach Erreichen des Minimums erneut an, da das Verfahren nicht den Abstand zur exakten Lösung, sondern den Abstand zur Lösung des Gleichungssystem minimiert.

Fazit Die Analyse des Mehrgitterverfahrens in doppelter Genauigkeit zeigt auf, dass bereits nach wenigen Iterationen der relative Fehler bis auf den Diskretisierungsfehler reduziert wird. Das relative Residuum wird durch das Verfahren zwar weiterhin reduziert, die Approximation jedoch nicht verbessert. Da der Diskretisierungsfehler ohnehin in einer Größenordnung zwischen $\mathcal{O}(10^{-4})$ und $\mathcal{O}(10^{-8})$ liegt, kann vermutet werden, dass hierfür bereits einfache oder gar halbe Genauigkeit ausreichend ist. Deshalb wird als Nächstes das Mehrgitterverfahren in einfacher Genauigkeit betrachtet.

Mehrgitterverfahren in einfacher Genauigkeit

In Abbildung 21 sind die Fehlerentwicklungen des Mehrgitterverfahrens mit einfacher Genauigkeit dargestellt. Es fällt auf, dass sich auch hier die Kurven der Residuen einem Grenzwert annähern. Wie zuvor sinkt dieser Grenzwert mit Absinken des Levels. Zahlenmäßig befinden sich die relativen Residuen bei Verwendung von Single deutlich oberhalb der von Double. Der Quotient zwischen $\mathrm{RR_d}$ und $\mathrm{RR_s}$ beträgt dabei knapp 1,80E−09 und ist nahezu konstant für alle betrachteten Problemgrößen. Für das 12. Level konvergiert das RR gegen einen Wert von knapp 1,93E−01. Bei der kleinsten Problemgröße sind es noch ca. 4,34E−05. Zum Vergleich: Die Berechnung einer einzelnen Vektoraddition in einfacher Genauigkeit liefert einen akkumulierten maximalen Rundungsfehler von $(2^{12}+1)^2 \cdot \varepsilon_s \approx 0{,}99$ auf Level 12 und $(2^6+1)^2 \cdot \varepsilon_s \approx 2{,}49\text{E}{-}04$ auf Level 6. Damit liegt der Fehler wieder in der gemessenen Größenordnung und es ist klar, dass keine kleineren Zahlen für das relative Residuum erwartet werden können.

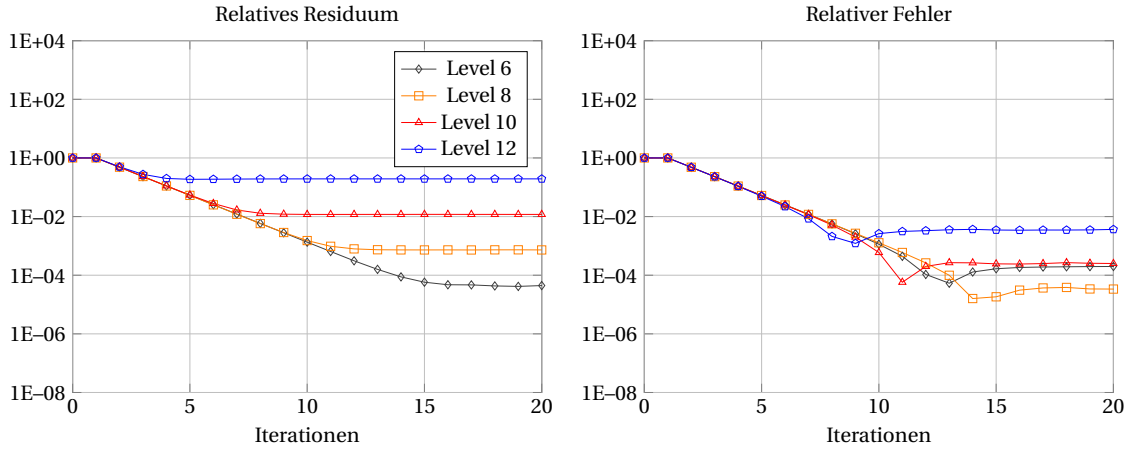


Abbildung 21: Fehlerentwicklungen des Mehrgitterverfahrens in einfacher Genauigkeit. Das relative Residuum (links) und der relative Fehler (rechts) sind für unterschiedliche Problemgrößen in Abhängigkeit der durchlaufenden Iterationen aufgetragen.

Betrachtet man den relativen Fehler, so fällt auf, dass sich die Kurven auch hier einem Grenzwert annähern. Im Vergleich zu den Messungen in doppelter Genauigkeit erhält man jedoch keine klare Abstufung zwischen den einzelnen Problemgrößen. Während der Grenzwert zwischen dem 6. und 8. Level absinkt, steigt er beim Übergang auf Level 10 und 12 wieder an. Der kleinste gemessene Wert liegt bei rund $7,28\text{E}{-}04$ auf Level 8. Für Level 12 konnte lediglich ein Wert von ca. $3,71\text{E}{-}03$ erreicht werden.

Dieses Verhalten lässt sich wie Folgt erklären: Wie bei der Berechnung des Residuums liegt der maximale Rundungsfehler durch die endliche Gleitkommaarithmetik in der Größenordnung der zu messenden Größe. Für das 6. Level spielen diese arithmetischen Fehler noch keine entscheidende Rolle, weshalb sich der relative Fehler in etwa in derselben Größenordnung befindet, wie bei der Messung in doppelter Genauigkeit. Ab dem 8. Level fangen die arithmetischen Fehler an zu dominieren und der relative Fehler steigt alleine durch die Ungenauigkeiten in den Rechnungen wieder an. Auch lässt sich bei Verwendung von einfacher Genauigkeit aus der Konvergenz des Residuums nicht zwingend auf die Konvergenz des Fehlers schließen. Für Level 12 fängt das Residuum bereits nach vier Iterationen an zu stagnieren. Der Fehler befindet sich jedoch erst nach knapp neun Iterationen in der Nähe des Grenzwerts.

An dieser Stelle sei darauf hingewiesen, dass es sich bei den Grenzwerten des relativen Fehlers in einfacher Genauigkeit nicht um den Diskretisierungsfehler handelt. Dieser ist per se unabhängig vom gewählten Datentyp. Da sich das RR in einfacher Genauigkeit in derselben oder einer höheren Größenordnung als der relative Fehler befindet, kann der Vorwärtsfehler nicht vernachlässigt werden. Ungleichung 13 ist zwar nach wie vor gültig, jedoch folgt daraus nicht unmittelbar, dass sich der RF in derselben Größenordnung wie der Diskretisierungsfehler befindet.

Fazit Auf Grund der größeren arithmetischen Fehler lässt sich das Residuum nicht beliebig reduzieren. Auch der relative Fehler liegt auf Grund von Rechenungenauigkeiten nicht in der Größenordnung des Diskretisierungsfehlers. Zudem impliziert die Konvergenz des Residuums nicht die Konvergenz des echten Fehlers. In den ersten Iterationen stimmen jedoch die Fehlerverläufe in einfacher Genauigkeit mit denen in doppelter Genauigkeit überein. Hier sollte es praktisch keinen Unterschied machen, ob in einfacher oder doppelter Genauigkeit gerechnet wird. Dieses Resultat motiviert die Untersuchung des Verfahrens mit dem nur halb so großen Datentyp Half.

Mehrgitterverfahren in halber Genauigkeit

Für die Analyse des Mehrgitterverfahrens in halber Genauigkeit werden im Folgenden nur die Level 6 bis 8 betrachtet. Mit einem maximalen Exponenten von 15 ist die größte Zahl, die in Half dargestellt werden kann, 65.504. Auf Level 9 beträgt die Gitterweite $h = 2^{-9} = 1/512$. Der Vorfaktor der Systemmatrix beläuft sich auf Grund der Diskretisierung mit der Finite-Differenzen-Methode somit auf $1/h^2 = 512^2 = 262.144$. Damit liegt er bereits deutlich oberhalb der größten darzustellenden Zahl.

Betrachtet man die Entwicklung des relativen Residuums aus Abbildung 22, so fällt auf, dass lediglich das Residuum auf Level 6 gegen einen Grenzwert kleiner Eins konvergiert. Für die an-

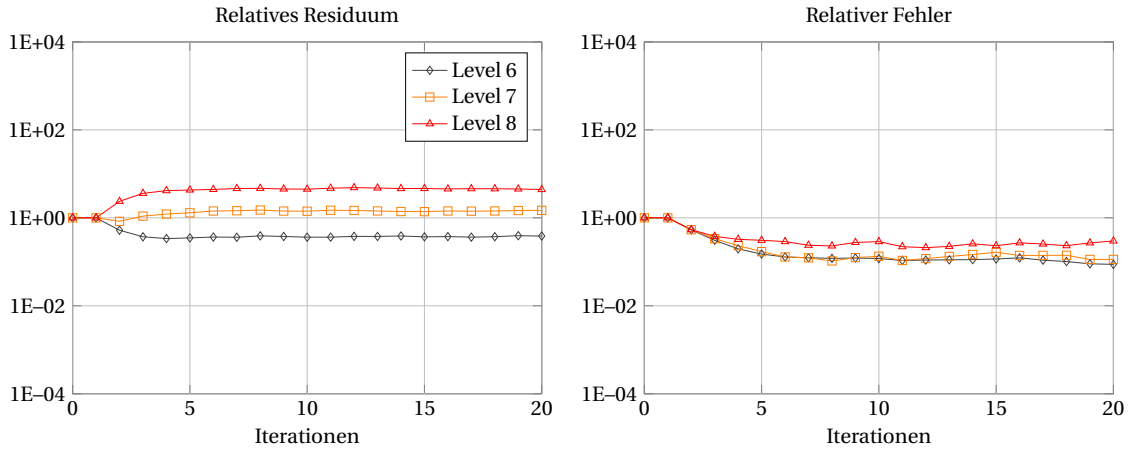


Abbildung 22: Fehlerentwicklungen des Mehrgitterverfahrens in halber Genauigkeit. Das relative Residuum (links) und der relative Fehler (rechts) sind für unterschiedliche Problemgrößen in Abhängigkeit der durchlaufenden Iterationen aufgetragen.

deren Level sind die arithmetischen Fehler bereits zu groß. Auf Level 7 beträgt der maximale akkumulierte Rundungsfehler für eine Vektoraddition bereits $(2^7+1)^2 \cdot \varepsilon_h \approx 7,99$. Das relative Residuum konvergiert für diese Problemgröße gegen einen Wert von knapp $1,42$.

Die Entwicklung des relativen Fehlers verläuft hingegen monoton fallend. Auffällig ist, dass sich die Reihenfolge der Grenzwerte im Vergleich zur doppelten Genauigkeit umgekehrt hat: Bei Verwendung von Half steigt der Grenzwert mit dem Level an. Dies kann ebenfalls auf die arithmetischen Fehler zurückgeführt werden. Bei der kleinsten gemessenen Problemgröße konvergiert der relative Fehler gegen $1,13\text{E}{-}01$. Im Gegensatz zur einfachen und doppelten Genauigkeit sind keine globalen Minima zu erkennen. Eine mögliche Erklärung lautet wie folgt: Da auch die exakte Lösung des Problems in Half ausgewertet wird, beinhaltet die Lösung bereits einen Fehler in der Größenordnung von $\varepsilon_h \in \mathcal{O}(10^{-4})$. Aus Sicht des Datentyps Half unterscheidet sich die exakte Lösung $\mathbf{v}$ des Problems deshalb kaum von der exakten Lösung $\tilde{\mathbf{v}}$ des Gleichungssystems.

Fazit Durch die Verwendung von halber Genauigkeit können nur wenige Problemgrößen betrachtet werden. Zudem lässt sich das Residuum bereits ab einem moderaten Level nicht mehr unter Eins reduzieren. Der relative Fehler hingegen fällt monoton fallend ab und konvergiert gegen einen Grenzwert kleiner Eins. Letzteres zeigt durchaus, dass selbst durch die Verwendung von Half eine gewisse Genauigkeit erzielt werden kann, sofern geeignete Problemgrößen verwendet werden.

Analyse der Ausführungszeit

Zum Ende dieses Unterkapitels werden die Ausführungszeiten zu je 1.000 Durchführungen für die unterschiedlichen Datentypen analysiert. In Abbildung 23 sind hierfür die Zeiten pro Level t_{level} und pro V-Zyklus t_{zyklus} in Abhängigkeit der Problemgröße aufgetragen. Ersteres beschreibt dabei die Zeit, welche für die Rechnungen auf einem Level benötigt wird, wobei der rekursive Aufruf nicht dazu gezählt wird. Es wird zunächst die Ausführungszeiten pro Level betrachtet. Es stellt sich heraus, dass für kleine Problemgrößen (Level 6 - Level 9) alle drei betrachteten Datentypen gleich viel Zeit auf dem jeweiligen Level benötigen. Die einzelnen Kurven liegen nahezu identisch übereinander. Dieses Verhalten ändert sich erst ab Level 10. Ab hier benötigt die Single-Implementierung dauerhaft weniger Zeit pro Level. Zwischen dem 9. und 10. Level steigt die Kurve der Double-Implementierung steiler an, als die der Single-Implementierung. Dies lässt sich anhand der Hardwarespezifikation erklären: Für Level 10 ergibt sich eine Problemgröße von $(2^{10}+1)^2 = 1.050.625$. Bei Verwendung von einfacher Genauigkeit benötigt ein Vektor auf diesem Level $1.050.625 \cdot 4$ Byte $\approx 4,2$ MByte Speicher. Damit passt dieser vollständig in den 6 MByte großen L2-Zwischenspeicher. Bei Verwendung von doppelter Genauigkeit benötigt ein Vektor derselben Größe entsprechend doppelt soviel Speicherplatz und kann damit nicht vollständig aus dem Zwischenspeicher geladen werden. Im Umkehrschluss bedeutet dies, dass bis zum 9. Level der Zwischenspeicher für alle Datentypen effizient ausgenutzt werden kann. Die Kurven unterscheiden sich bis zu dieser Problemgröße deshalb kaum voneinander. Für die größte betrachtete Problemgröße (Level 12) benötigen 1.000 Single-Durchführungen ca. 3,29 Sekunden. Bei der Double-Implementierung sind es knapp 5,30 Sekunden. Damit ist die Single-Implementierung für diese Problemgröße um den Faktor 1,6 schneller.

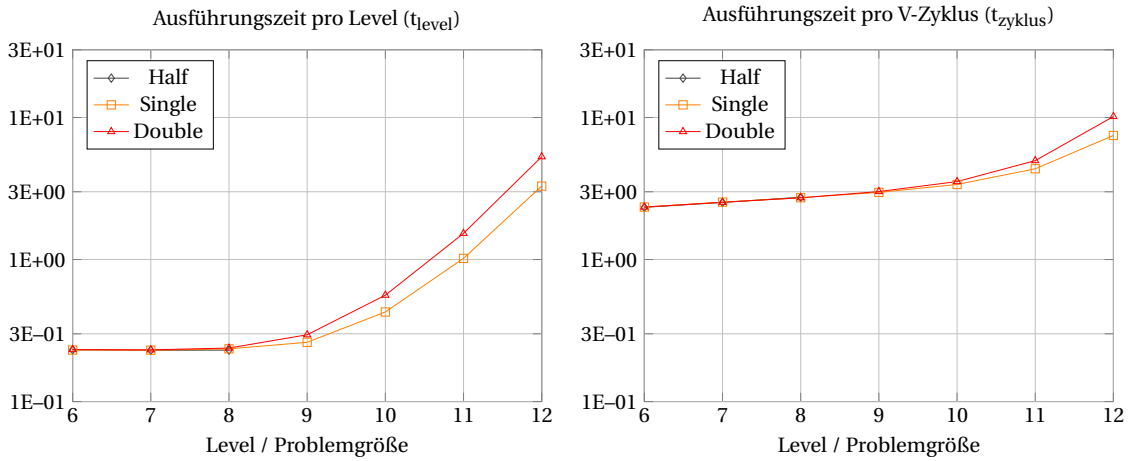


Abbildung 23: Ausführungszeiten des Mehrgitterverfahrens mit einem Datentyp. Die Gemessenen Zeiten sind in Sekunden angegeben und über die Problemgröße aufgetragen. Links: Ausführungszeit pro Level. Rechts: Ausführungszeit pro V-Zyklus. Die Zeiten beziehen sich auf je 1.000 Durchführungen.

Da in den gemischt genauen Verfahren auch ganze V-Zyklen in einem Datentyp ausgeführt werden, wird als Nächstes die Ausführungszeit pro V-Zyklus betrachtet. Die gemessenen Zeiten belaufen sich wieder auf je 1.000 Durchführungen. Für kleine Problemgrößen zeigt sich dasselbe Bild wie bereits zuvor: Die Kurven für Half, Single und Double unterscheiden sich kaum. Dies ist anschaulich klar, da sich bereits die Zeiten pro Level für kleine Problemgrößen kaum unterscheiden. Für das größte Level benötigt die Single-Implementierung im Schnitt 7,47 Sekunden. Bei Verwendung von doppelter Genauigkeit liegt der gemessene Wert bei knapp 10,16 Sekunden, was einem Faktor von 1,36 entspricht. Da die Zeiten pro Level für kleine Problemgrößen nahezu identisch sind, ist ein Faktor in der Nähe von 1,6 an dieser Stelle nicht zu er-

warten.

Nachdem das Mehrgitterverfahren mit einem Datentyp analysiert wurde, stehen als Nächstes die gemischt genauen Verfahren im Fokus. Hierbei wird zunächst mit den Horizontal-Verfahren begonnen.

4.4 H2SMG- und H3SMG-Verfahren

Im Folgenden werden die beiden Horizontal-Verfahren näher betrachtet und analysiert. Dabei wird mit dem H2SMG-Verfahren begonnen und zunächst die verwendeten Abbruchkriterien und die Anzahl an Iterationen diskutiert. Im Anschluss werden dann die Fehlerentwicklungen und Ausführungszeiten mit dem Mehrgitterverfahren in doppelter Genauigkeit verglichen.

H2SMG-Verfahren

Wie bereits in Kapitel 3.3 erläutert, wird für das H2SMG-Verfahren ein geeignetes Abbruchkriterium für die ersten Iterationen in einfacher Genauigkeit benötigt. Während für die Double-Iterationen dasselbe Abbruchkriterium wie für ein herkömmliches Mehrgitterverfahren in doppelter Genauigkeit verwendet werden kann, gilt dies nicht zwingend für die Single- und Half-Iterationen. Bei Verwendung des Residuums als Abbruchkriterium muss beachtet werden, dass auf Grund der größeren Rundungsfehler eine Bedingung der Form $||\mathbf{r}|| < \varepsilon_s$ für Single und Half möglicherweise nicht erreicht werden kann. Dies gilt selbst dann, wenn das Residuum in doppelter Genauigkeit berechnet wird. Hierfür sind a posteriori-Abbruchkriterien besser geeignet, da sie die Stagnation des Fehlers beurteilen. Ein mögliches a posteriori-Fehlermaß ist die *relative Residuumsrate* (RRR). Diese berechnet sich zu

$$\mathrm{RRR(k)} = \frac{||\mathbf{r}^{(\mathrm{m})} - \mathbf{r}^{(\mathrm{m}-1)}||}{||\mathbf{f}||}.$$

Neben dem relativen Residuum lässt sich auch der relative Fehler in eine Rate überführen. Die sog. *relative Fehlerrate* (RFR) berechnet sich zu

$$\mathrm{RFR(k)} = \frac{||\mathbf{v}^{(\mathrm{m})} - \mathbf{v}^{(\mathrm{m}-1)}||}{||\mathbf{v}^{(\mathrm{m})}||}$$

und lässt sich im Gegensatz zum relativen Fehler während dem Verfahren berechnen. Da für die in dieser Arbeit betrachteten Probleme die analytische Lösung bekannt ist, lässt sich die optimale Anzahl an Iterationen numerisch bestimmen. Hierzu wird die Entwicklung des relativen Fehlers beim Mehrgitterverfahren in einfacher Genauigkeit betrachtet (vgl. Abbildung 21). Aus den Daten lässt sich die Anzahl an Iterationen bis zu dem Punkt bestimmen, an dem sich der relative Fehler nicht mehr weiter reduzieren lässt. Pro Level ergibt dies eine feste Zahl, welche die optimale Anzahl an Single-V-Zyklen des H2SMG-Verfahrens beschreibt.

Für den Vergleich mit dem Mehrgitterverfahren in doppelter Genauigkeit wird ebenfalls ein Abbruchkriterium benötigt. Hierfür kann das relative Residuum verwendet werden. Auf Grund der Untersuchungen aus Kapitel 4.3 ist neben Single auch die optimale Anzahl an Iterationen für das Mehrgitterverfahren in doppelter Genauigkeit bekannt (vgl. Abbildung 20). Deshalb wird im Folgenden nicht das Residuum verwendet, sondern analog zu den Single-Iterationen eine feste Anzahl an Iterationen des herkömmlichen Mehrgitterverfahrens durchgeführt. Für den Vergleich mit dem H2SMG-Verfahren wird dabei wie folgt vorgegangen: Insgesamt werden bei beiden Verfahren gleich viele Iterationen betrachtet. Beim H2SMG-Verfahren wird ein Teil davon in einfacher Genauigkeit und die übrigen in doppelter Genauigkeit durchgeführt. Nach der Durchführung werden die Ausführungszeiten miteinander verglichen. Die Anzahl an Iterationen des Mehrgitter- und H2SMG-Verfahrens sind in Abbildung 24 zusammengefasst. Die

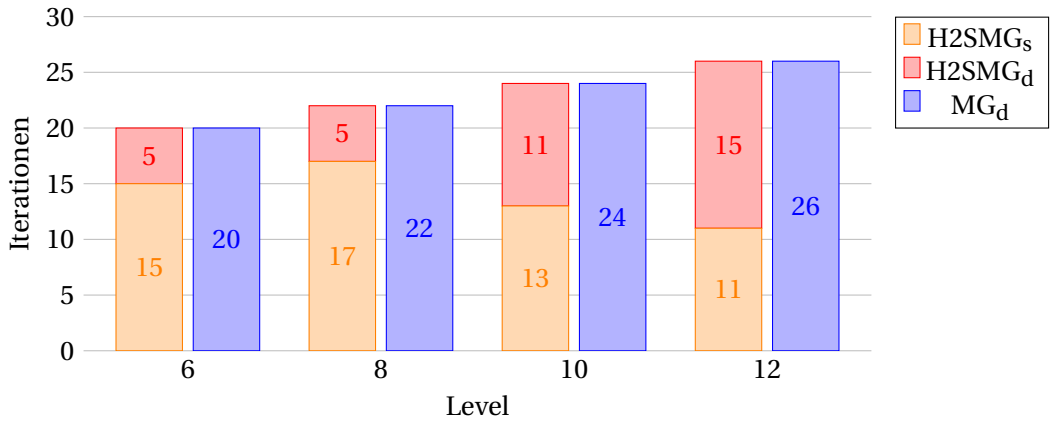


Abbildung 24: Anzahl an Iterationen für das H2SMG- und Mehrgitterverfahren in doppelter Genauigkeit. Die Anzahl an Single- (H2SMG$_s$) und Double- (H2SMG$_d$) Iterationen des H2SMG-Verfahrens ergeben in Summe die Anzahl an durchgeführten Iterationen des Mehrgitterverfahrens in doppelter Genauigkeit (MG$_d$).

Notation *H2SMG$_s$* steht dabei für die Single-Iterationen des H2SMG-Verfahrens. Für den Vergleich der Fehlerentwicklungen werden bei beiden Verfahren mehr Iterationen durchgeführt, um die Verläufe des RR und des RF beurteilen zu können.

In Abbildung 25 sind die Fehlerentwicklungen des H2SMG-Verfahrens aufgetragen. In der zugrundeliegenden Implementierung wurden hierbei nicht die gemischt genauen DEFECT- und RESTRICT-Kernel verwendet. Bei der Betrachtung des Residuums (links) fällt deshalb auf, dass an den Stellen, an denen horizontal gesehen der Wechsel von Single nach Double stattfindet, ein Sprung zu erkennen ist. Dies gilt für alle betrachteten Problemgrößen. Im weiteren Verlauf fallen die Residuen monoton ab und beginnen ab ca. 40 Iterationen zu stagnieren. Für die größte betrachtete Problemgröße beträgt der gemessene Wert knapp $7,00\text{E}-05$. Mit sinkender Problemgröße sinkt auch das Residuum. Für das kleinste Level beträgt das relative Residuum nach 50 Iterationen rund $2,86\text{E}-10$. Vergleicht man die Werte mit denen des Mehrgitterverfahrens in doppelter Genauigkeit, so ist das Residuum beim H2SMG-Verfahren um den Faktor $1,00\text{E}+04$ bis $1,00\text{E}+05$ größer. Dies resultiert aus dem sprunghaften Verhalten beim Wechsel

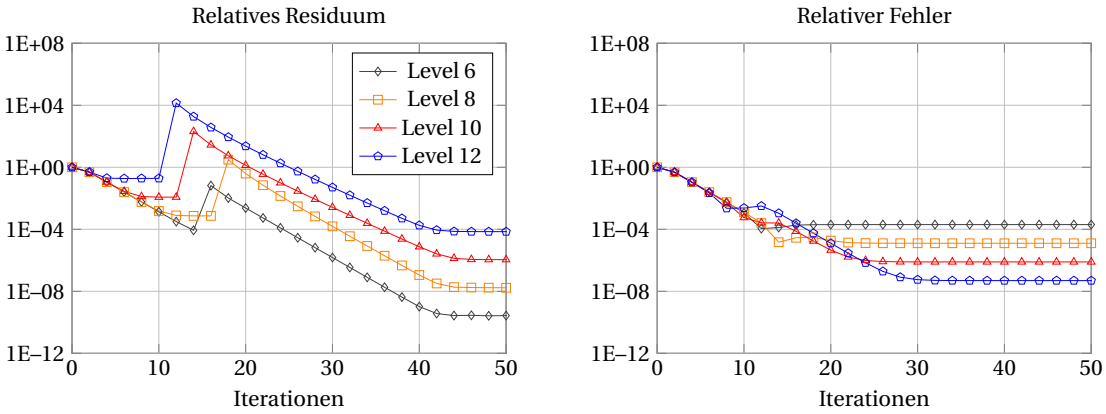
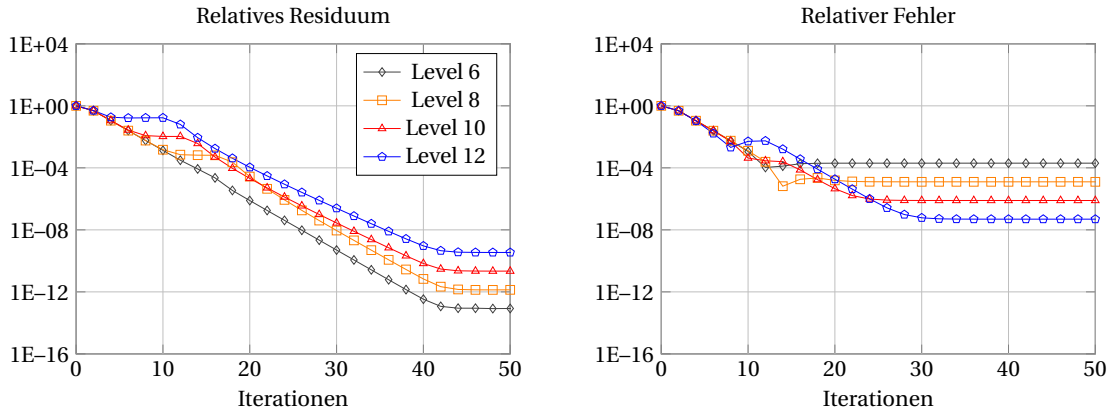


Abbildung 25: Fehlerentwicklungen des H2SMG-Verfahrens ohne den gemischt genauen DEFECT- und RESTRICT-Kernel. Das relative Residuum (links) und der relative Fehler (rechts) sind für unterschiedliche Problemgrößen in Abhängigkeit der durchlaufenden Iterationen aufgetragen.

der Datentypen und kann wie folgt interpretiert werden: Für die V-Zyklen in einfacher Genau-

igkeit wird das Residuum auch in einfacher Genauigkeit berechnet. Zusätzlich wird die rechte Seite ${}^{s}\mathbf{f}$ im Datentyp Single für die Berechnung verwendet. Dies führt dazu, dass die Single-Iterationen das Residuum zu einem gegenüber dem Ausgangsproblem ${}^{d}\mathbf{f}$ leicht veränderten Problem minimieren. Beim Übergang zur doppelten Genauigkeit verändert sich die rechte Seite (${}^{s}\mathbf{f} \rightarrow {}^{d}\mathbf{f}$) und dadurch auch die Qualität des Residuums (${}^{s}\mathbf{r} \rightarrow {}^{d}\mathbf{r}$).

Der relative Fehler weist hingegen keine Sprünge auf, was anschaulich klar ist, da seine Berechnung unabhängig von der rechten Seite erfolgt. Da der RF nur für die numerische Analyse verwendet werden kann, wird er in der Implementierung stets in doppelter Genauigkeit berechnet. Dies gilt zumindest für die Berechnung der Normen. Die Approximation ${}^{s}\mathbf{v}^{(m)}$ wird dabei von Single nach Double konvertiert. Um das RR dennoch als mögliches Abbruchkriterium verwenden zu können, kann die Berechnung des Residuums beim H2SMG-Verfahren in doppelter Genauigkeit durchgeführt werden. Hierfür werden der gemischt genaue DEFECT- und RESTRICT-Kernel verwendet (vgl. Algorithmus 7). In Abbildung 26 sind die Fehlerentwicklungen für diese Implementierung aufgeführt. Es zeigt sich, dass das relative Residuum durch diese Anpassung keine Sprünge mehr aufweist und sehr glatt verläuft. Die gemessenen Residuen befinden sich nach 40 Iterationen in derselben Größenordnung wie beim Mehrgitterverfahren in doppelter Genauigkeit. Wie zu erwarten war, haben sich die Verläufe des relativen Fehlers nicht geändert. Nach ca. 30 Iterationen liegen auch hier die gemessenen Werte in derselben Größenordnung wie die relativen Fehler des herkömmlichen Mehrgitterverfahrens.



Level	T_{h2smg}	T_{mg}	Exp. Speed-Up	Theo. Speed-Up	Δ_{rf}
6	4,15E–02	4,37E–02	1,05	1,58	8,80E–08
8	5,38E–02	5,73E–02	1,07	1,56	8,72E–07
10	7,42E–02	8,11E–02	1,09	1,33	2,29E–07
12	2,14E–01	2,68E–01	1,25	1,24	2,22E–07

Abbildung 26: Fehlerentwicklungen, Ausführungszeiten und Speed-Up des H2SMG-Verfahrens mit dem gemischt genauen DEFECT- und RESTRICT-Kernel. Das relative Residuum (links) und der relative Fehler (rechts) sind für unterschiedliche Problemgrößen in Abhängigkeit der durchlaufenden Iterationen aufgetragen. Unten: Ausführungszeiten T in Sekunden und Differenz der relativen Fehler Δ_{rf} des H2SMG- und Mehrgitterverfahrens in doppelter Genauigkeit (MG). Der experimentelle Speed-Up berechnet sich aus dem Quotienten der beiden gemessenen Ausführungszeiten, d.h. $\text{Speed}-\text{Up} = T_{mg}/T_{h2smg}$. Der theoretische Speed-Up kann gemäß Kapitel 3.3 aus dem Verhältnis zwischen Double- und Single-Iterationen bestimmt werden. Die Fehlerdifferenz berechnet sich zu $\Delta_{rf} = RF_{h2smg} - RF_{mg}$.

In Abbildung 26 sind ebenfalls die gemessenen Ausführungszeiten mit zugehöriger Fehlerdifferenz aufgeführt. Der gemessene Speed-Up gegenüber dem herkömmlichen Mehrgitterverfahren beträgt knapp 1,05 auf dem kleinsten Level und wächst monoton bis auf 1,25 für das größte Level an. Da für kleine Problemgrößen die Ausführungszeit eines V-Zyklus nahezu identisch für alle Datentypen ist, lässt sich zumindest sagen, dass der zusätzliche Mehraufwand durch die Konvertierungen vernachlässigbar ist. Für große Problemgrößen steigert sich der gemessene Speed-Up deutlich. Die Berechnungen auf dem obersten Level dominieren sowohl die Lade- als auch Rechenzeit. Durch die Verwendung von einfacher Genauigkeit entsteht hier ein zeitlicher Vorteil, welcher sich stark auf die gesamte Ausführungszeit auswirkt. Auf dem 12. Level liegt der gemessene Speed-Up sogar über der theoretischen Schranke. Dies kann auf die effizientere Verwendung der Zwischenspeicher zurückgeführt werden, welche bei der Herleitung des theoretischen Speed-Ups vernachlässigt wurden.

Die relativen Fehler des H2SMG-Verfahrens unterscheiden sich kaum von denen des herkömmlichen Mehrgitterverfahrens. Mit ca. $8,72\text{E}-07$ auf Level 8 liegt die größte Differenz noch in derselben Größenordnung wie der kleinste gemessen relative Fehler.

Fazit Sofern das RR als Abbruchkriterium für das H2SMG-Verfahren verwendet werden soll, muss die Berechnung des Residuums auf dem feinsten Level in doppelter Genauigkeit durchgeführt werden. Die Entwicklung des relativen Fehlers ist unabhängig davon und verläuft nahezu identisch wie beim herkömmlichen Mehrgitterverfahren. Der gemessene Speed-Up liegt zwischen 1,05 und 1,25 und wächst mit steigender Problemgröße an.

H3SMG-Verfahren

Beim dreischichtigen Horizontal-Verfahren stellt sich ebenfalls die Frage, wie viele Iterationen in halber bzw. einfacher Genauigkeit durchgeführt werden sollen. Hierzu kann analog zum H2SMG-Verfahren vorgegangen und die Fehlerentwicklung des Mehrgitterverfahrens in halber Genauigkeit herangezogen werden (vgl. Abbildung 22). Nach 8 Iterationen zeigt sich, dass der relative Fehler nicht mehr weiter reduziert wird. Deshalb werden beim H3SMG-Verfahren stets 8 Iterationen in halber Genauigkeit durchgeführt. Dies führt dazu, dass die Anzahl an Single-Iterationen im Gegensatz zum H2SMG-Verfahren sinkt. Mit anderen Worten bedeutet das, dass das H3SMG-Verfahren aus dem H2SMG-Verfahren entsteht, indem die ersten 8 Single-Iterationen in halber Genauigkeit durchgeführt werden. Die verwendete Anzahl an Iterationen beim H3SMG-Verfahren sind in Abbildung 27 zusammengefasst. Da bei der Verwendung von halber Genauigkeit nur Problemgrößen bis Level 8 betrachtet werden können, werden im Folgenden nur die Level 6 und 8 des H3SMG-Verfahrens analysiert. Analog zum H2SMG-Verfahren wird in den Half- und Single-Iterationen das Residuum auf dem größten Level stets in doppelter Genauigkeit berechnet.

In Abbildung 28 sind die Fehlerentwicklungen des H3SMG-Verfahrens aufgeführt. Im Gegensatz zum H2SMG-Verfahren fällt das RR für das 8. Level nicht monoton ab, was gerade dem Verlauf der ersten acht Half-Iterationen entspricht. Die Folge davon ist, dass deutlich mehr Iterationen notwendig sind, bis das RR zu stagnieren beginnt. Nach knapp 50 Iterationen liegt der Wert bei rund $1,36\text{E}{-12}$ und damit in derselben Größenordnung wie beim Mehrgitterverfahren in doppelter Genauigkeit. Beim Verlauf des RR auf dem 6. Level lassen sich die drei Schichten deutlich erkennen. In den ersten acht Iterationen wird das Residuum kaum reduziert. Für die nächsten sieben Iterationen verläuft die Kurve sehr steil, was durch die Single-V-Zyklen zustande kommt. Nach 15 Iterationen wächst sie wieder leicht an. Letzteres entspricht dem Übergang von Single zu Double. Das Residuum auf dem 6. Level fängt bereits ab 40 Iterationen an zu stagnieren und befindet sich ebenfalls in derselben Größenordnung wie beim herkömmlichen

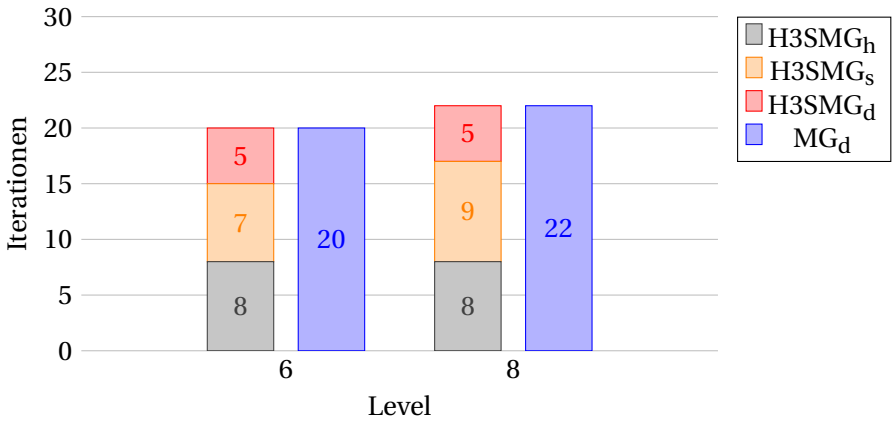
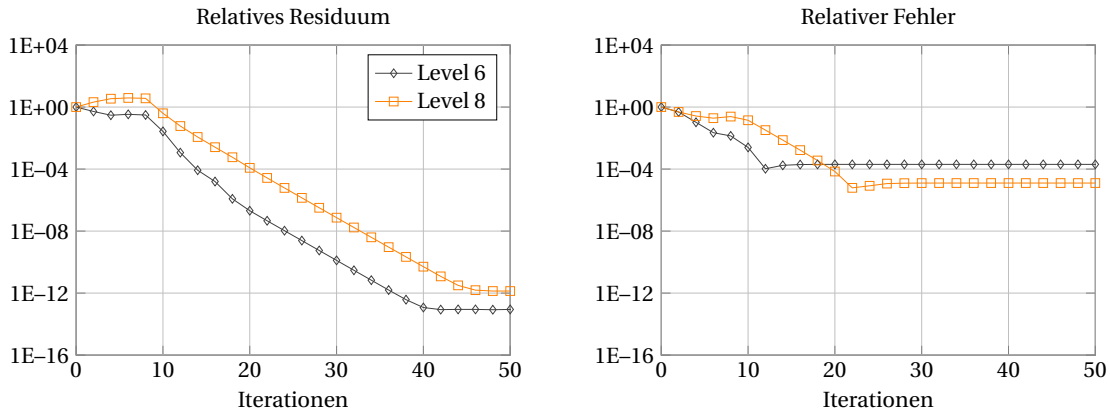


Abbildung 27: Anzahl an Iterationen für das H3SMG- und Mehrgitterverfahren in doppelter Genauigkeit. Die Anzahl an Half- (H3SMG$_h$), Single- (H3SMG$_s$) und Double- (H2SMG$_d$) Iterationen des H3SMG-Verfahrens ergeben in Summe die Anzahl an durchgeführten Iterationen des Mehrgitterverfahrens in doppelter Genauigkeit (MG$_d$).

Mehrgitterverfahren.

Für den relativen Fehler zeigt sich ein ähnliches Resultat. Auch hier werden auf dem 8. Level mehr Iterationen benötigt, bis der Fehler stagniert. Beim 6. Level wird der RF durch die Anwen-



Level	T$_{H3SMG}$	T$_{MG}$	Exp. Speed-Up	Theo. Speed-Up	Δ_{RF}
6	4,16E–02	4,37E–02	1,05	1,87	5,73E–07
8	5,40E–02	5,73E–02	1,06	1,81	-5,23E–06

Abbildung 28: Fehlerentwicklungen und Ausführungszeiten des H3SMG-Verfahrens. Das relative Residuum (links) und der relative Fehler (rechts) sind für unterschiedliche Problemgrößen in Abhängigkeit der durchlaufenden Iterationen aufgetragen. Unten: Ausführungszeiten T in Sekunden und Differenz der relativen Fehler Δ_{rf} des H3SMG- und Mehrgitterverfahrens in doppelter Genauigkeit (MG). Der experimentelle Speed-Up berechnet sich aus dem Quotienten der beiden gemessenen Ausführungszeiten, d.h. Speed−Up = T_{mg}/T_{h3smg}. Der theoretische Speed-Up kann gemäß Kapitel 3.3 aus dem Verhältnis zwischen Double- und Single- und Half-Iterationen bestimmt werden. Die Fehlerdifferenz berechnet sich zu $\Delta_{rf} = RF_{h3smg} - RF_{mg}$.

dung der Single-V-Zyklen sehr schnell reduziert. Die anschließenden Double-V-Zyklen schaffen es nicht, den Fehler weiter zu reduzieren, da der relative Fehler zu diesem Zeitpunkt bereits bis auf den Diskretisierungsfehler reduziert wurde.

Für die Ausführungszeiten des H3SMG-Verfahrens (Abbildung 28) zeigt sich ein ähnliches Resultat wie bereits beim H2SMG-Verfahren. Der Speed-Up beträgt 1,05 für das 6. bzw. 1,06 für das 8. Level. Durch die zusätzliche Verwendung von halber Genauigkeit lässt sich kein besseres Resultat erzielen. Da die Ausführungszeiten der V-Zyklen bei kleinen Problemgrößen und für alle Datentypen nahezu identisch sind, ist das Resultat nicht weiter verwunderlich. Einzig kann festgehalten werden, dass auch beim dreischichtigen Horizontal-Verfahren der Mehraufwand durch Konvertierungen vernachlässigbar ist. Die erzielten relativen Fehler unterscheiden sich nur geringfügig von denen des herkömmlichen Mehrgitterverfahrens. Mit $\Delta_{\mathrm{RF}} = -5,23\mathrm{E}{-}06$ für das 8. Level liegt die Differenz bereits eine Größenordnung unterhalb der gemessenen Werte.

Fazit Durch die zusätzliche Verwendung von halber Genauigkeit kann der Speed-Up nicht weiter verbessert werden. Mit steigender Problemgröße werden mehr Iterationen zum Erreichen derselben Genauigkeit für das relative Residuum benötigt. Der relative Fehler reduziert sich nach gleich vielen Iterationen auf denselben Wert, wie beim herkömmlichen Mehrgitterverfahren.

4.5 V2SMG- und V3SMG-Verfahren

Als Nächstes werden die beiden gemischt genauen Vertikal-Verfahren betrachtet. Hierbei wird mit dem zweischichtigen Verfahren begonnen, welches die Datentypen Single und Double verwendet. Zunächst werden wie beim H2SMG-Verfahren die verwendeten Abbruchkriterien erläutert. Im Anschluss daran erfolgt die Analyse der Fehlerentwicklungen und der Ausführungszeiten. Für den Vergleich mit dem Mehrgitterverfahren in doppelter Genauigkeit werden gleich viele Iterationen durchgeführt, wie dies bei den Horizontal-Verfahren der Fall war (vgl. Abbildung 24 bzw. 27).

V2SMG-Verfahren

Im Gegensatz zu den Horizontal-Verfahren werden bei den Vertikal-Verfahren nicht ganze V-Zyklen, sondern einzelne Level in unterschiedlichen Datentypen durchgeführt. Hierbei stellt sich die Frage, ab welchem Level der Wechsel der Datentypen stattfinden soll. Für das V2SMG-Verfahren haben die numerischen Experimente gezeigt, dass es ausreichend ist, das erste Level in doppelter und die restlichen in einfacher Genauigkeit durchzuführen. Deshalb wird beim V2SMG-Verfahren das erste Level stets in doppelter Genauigkeit gerechnet. Im Folgenden werden die Problemgrößen beginnend mit Level 8, 10 und 12 für die numerische Analyse verwendet. Die entsprechenden Konfigurationen der Level sind in Abbildung 29 aufgeführt. Die Notation $V2SMG_{8d,7s}$ beschreibt dabei ein V2SMG-Verfahren, welches auf dem 8. Level Double und ab dem 7. Level Single verwendet.

In Abbildung 30 sind die Fehlerentwicklungen des V2SMG-Verfahrens für die unterschiedlichen Konfigurationen aufgetragen. Es wird zunächst das relative Residuum betrachtet. Die Verläufe decken sich nahezu perfekt mit denen des herkömmlichen Mehrgitterverfahrens (vgl. Abbildung 20). Für alle Konfigurationen wird das RR gleich schnell minimiert und nach knapp 40 Iterationen haben sich die Kurven an die jeweiligen Grenzwerte angenähert. Dasselbe gilt für die Verläufe des relativen Fehlers. Auch hier ist kein sichtbarer Unterschied zu erkennen. Dieses Verhalten lässt sich wie folgt erklären: Auf jedem Level wird bei einem V-Zyklus die Residuumsgleichung gelöst. Hierzu wird rekursiv ein weiterer V-Zyklus verwendet. Beim V2SMG-Verfahren besteht der Unterschied darin, dass der rekursiv aufgerufene V-Zyklus in einfacher

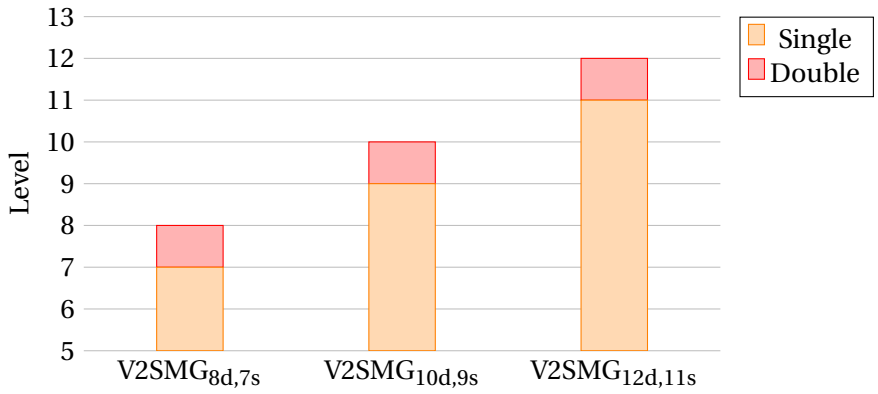
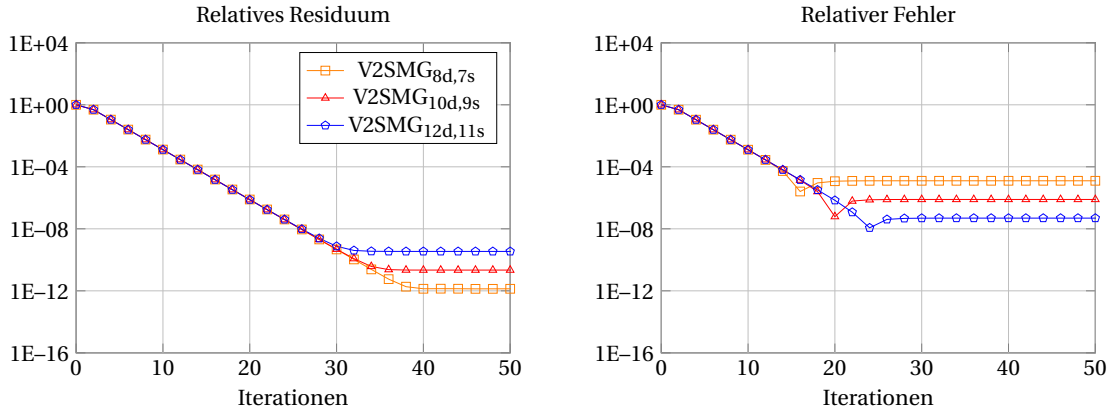


Abbildung 29: Konfigurationen für die einzelnen Level beim V2SMG-Verfahren. Das erste Level wird stets in doppelter Genauigkeit durchgeführt, die restlichen Level in einfacher Genauigkeit.

Genauigkeit durchgeführt wird. In jeder Iteration wird das Residuum neu berechnet, weshalb die Residuumsgleichung in jedem Schritt ein leicht verändertes Problem darstellt. Der rekursive Aufruf kann deshalb als die erste Iteration eines Mehrgitterverfahrens, angewandt auf die Residuumsgleichung, verstanden werden. Da sich die Fehlerverläufe zwischen Double und Single in den ersten Iterationen nicht unterscheiden (vgl. Kapitel 4.3), wird die Residuumsgleichung in beiden Fällen „gleich gut“ gelöst.



Konfiguration	T_{V2SMG}	T_{MG}	Exp. Speed-Up	Theo. Speed-Up	Δ_{RF}
8d, 7s	5,40E–02	5,73E–02	1,06	1,92	1,23E–12
10d, 9s	7,56E–02	8,11E–02	1,07	1,91	1,22E–12
12d, 11s	2,22E–01	2,68E–01	1,21	1,90	1,30E–12

Abbildung 30: Fehlerentwicklungen und Ausführungszeiten des V2SMG-Verfahrens. Das relative Residuum (oben, links) und der relative Fehler (oben, rechts) sind für unterschiedliche Level-Konfigurationen in Abhängigkeit der durchlaufenden Iterationen aufgetragen. Unten: Ausführungszeiten T in Sekunden und Differenz der relativen Fehler Δ_{rf} des V2SMG- und Mehrgitterverfahrens in doppelter Genauigkeit (MG). Der experimentelle Speed-Up berechnet sich aus dem Quotienten der beiden gemessenen Ausführungszeiten, d.h. $Speed-Up = T_{mg}/T_{v2smg}$. Der theoretische Speed-Up kann gemäß Kapitel 3.4 bestimmt werden. Die Fehlerdifferenz berechnet sich zu $\Delta_{rf} = RF_{v2smg} - RF_{mg}$.

Neben den Fehlerverläufen sind in Abbildung 30 auch die Ausführungszeiten des V2SMG-Verfahrens aufgeführt. Hierbei zeigt sich lediglich ein geringer Speed-Up. Mit 1,06 bei Level 8 und 1,20 bei Level 12 befindet sich der Wert in derselben Größenordnung wie schon beim H2SMG-Verfahren. Mit steigender Problemgröße wächst auch der gemessene Speed-Up, da insbesondere die Level 10 und 11 in einfacher Genauigkeit deutlich schneller durchgeführt werden können. Im Gegensatz zum H2SMG-Verfahren steigt der Wert jedoch nicht so schnell an, da das feinste Level den Aufwand dominiert, welches beim V2SMG-Verfahren in doppelter Genauigkeit durchgeführt wird. Die Differenz der relativen Fehler befinden sich in der Größenordnung von $1,00\mathrm{E}-12$. Damit stimmen die Werte nach gleich vielen Iterationen nahezu perfekt mit denen des herkömmlichen Mehrgitterverfahrens überein.

Fazit Beim V2SMG-Verfahren ist es ausreichend, das feinste Level in doppelter Genauigkeit zu berechnen. Die Verwendung von einfacher Genauigkeit bei den anderen Level hat keine Auswirkung auf die Fehlerentwicklung. Der Speed-Up liegt zwischen 1,06 und 1,20 und wächst mit steigender Problemgröße an.

V3SMG-Verfahren

Beim dreischichtigen Vertikal-Verfahren stellt sich ebenfalls die Frage, wann zwischen den Datentypen gewechselt wird. Analog zum V2SMG-Verfahren wird beim V3SMG-Verfahren das feinste Level stets in doppelter Genauigkeit gerechnet. Die verbleibenden Level teilen sich auf in je gleich viele Level für Single und Half. In Abbildung 31 sind die entstehenden Konfigurationen aufgeführt. Die Notation $\mathrm{V3SMG}_{8\mathrm{d},7\mathrm{s},6\mathrm{h}}$ steht dabei für ein V3SMG-Verfahren, welches auf dem 8. Level in doppelter, auf dem 7. Level in einfacher und ab dem 6. Level in halber Genauigkeit rechnet.

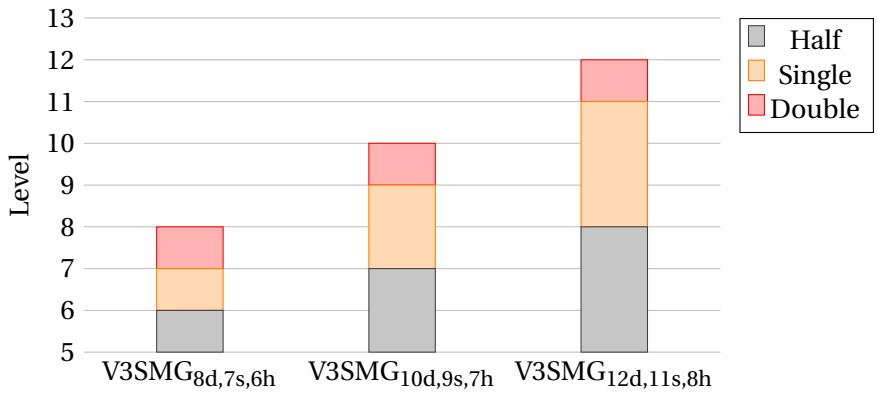
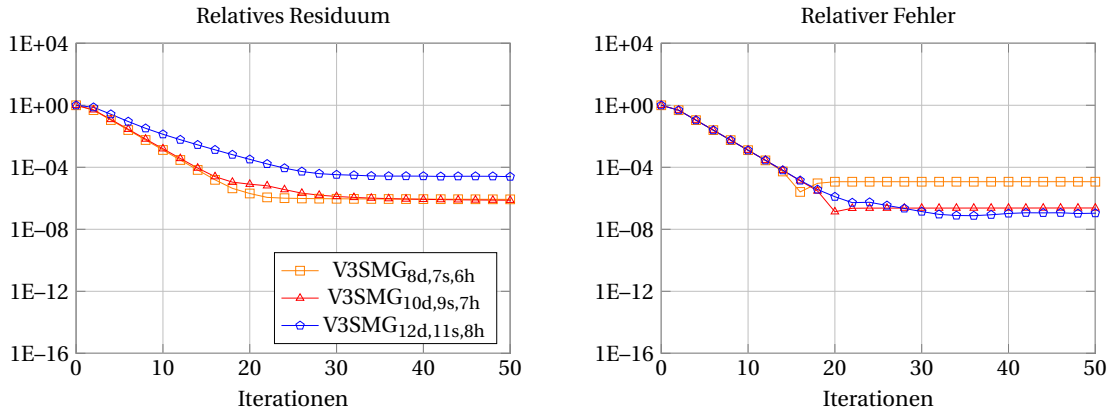


Abbildung 31: Konfigurationen für die einzelnen Level beim V3SMG-Verfahren. Das erste Level wird stets in doppelter Genauigkeit durchgeführt. Die restlichen Level teilen sich auf in einfache und halbe Genauigkeit.

Abbildung 32 zeigt die Verläufe des relativen Residuums und des relativen Fehlers. Zunächst wird der relative Fehler betrachtet. Die Verläufe entsprechen im Wesentlichen denen des V2SMG-Verfahrens und damit denen des herkömmlichen Mehrgitterverfahrens. Lediglich für das Verfahren mit der größten Problemgröße schwankt der Fehler bei den hinteren Iterationen etwas. Er befindet sich allerdings in derselben Größenordnung. Die zusätzliche Verwendung von halber Genauigkeit wirkt sich demnach nicht wesentlich auf die Entwicklung des relativen Fehlers aus.

Die Verläufe der relativen Residuen unterscheiden sich stark von denen des herkömmlichen

Mehrgitterverfahrens. Bei allen Konfigurationen ist die Konvergenz langsamer. Mit ca. 8,36E–07 für das kleinste und knapp 2,09E–05 für das größte Level sind die Werte um fünf Größenordnungen größer, als beim Mehrgitterverfahren in doppelter Genauigkeit. Erklären lässt sich dies anhand der Fehlerentwicklungen des Mehrgitterverfahrens in halber Genauigkeit (vgl. Abbildung 22). Hier zeigte sich, dass bei der Berechnung des Residuums in halber Genauigkeit viele Rundungsfehler auftreten, die nicht vernachlässigt werden können. Dasselbe gilt für die Anwendung der 151 Jacobi-Iterationen auf dem feinsten Level. Dies führt dazu, dass die entsprechende Lösungsapproximation mit vielen Rundungsfehlern behaftet ist. Bei der Berechnung des RR auf dem feinsten Gitter werden die Rundungsfehler aufsummiert, was zu einer schlechten Konvergenz führt. Die Fehler in den Rechnungen wirken sich, analog zum Mehrgitterverfahren in halber Genauigkeit, nicht so stark auf den relativen Fehler aus, da bei seiner Berechnung weniger Rechenoperationen durchgeführt werden müssen. Aus diesem Grund befindet sich das relative Residuum auch in derselben Größenordnung wie der relative Fehler. In Anhang B befindet sich eine weitere Untersuchung dieses Verhaltens mit dem V3SMG-Verfahren, welches lediglich die 151 Jacobi-Iterationen auf dem gröbsten Gitter in halber Genauigkeit durchführt.



Level	T_{V3SMG}	T_{MG}	Exp. Speed-Up	Theo. Speed-Up	Δ_{RF}
8	5,40E–02	5,73E–02	1,06	2,27	-6,76E–07
10	7,59E–02	8,11E–02	1,07	1,96	-5,14E–07
12	2,23E–01	2,68E–01	1,20	1,91	3,04E–07

Abbildung 32: Fehlerentwicklungen und Ausführungszeiten des V3SMG-Verfahrens. Das relative Residuum (oben, links) und der relative Fehler (oben, rechts) sind für unterschiedliche Level-Konfigurationen in Abhängigkeit der durchlaufenden Iterationen aufgetragen. Unten: Ausführungszeiten T in Sekunden und Differenz der relativen Fehler Δ_{rf} des V3SMG- und Mehrgitterverfahrens in doppelter Genauigkeit (MG). Der experimentelle Speed-Up berechnet sich aus dem Quotienten der beiden gemessenen Ausführungszeiten, d.h. $Speed-Up = T_{mg}/T_{v3smg}$. Der theoretische Speed-Up kann gemäß Kapitel 3.4 bestimmt werden. Die Fehlerdifferenz berechnet sich zu $\Delta_{rf} = RF_{v3smg} - RF_{mg}$.

Die Ausführungszeiten des V3SMG-Verfahrens sind ebenfalls in Abbildung 32 aufgeführt. Der Speed-Up liegt in derselben Größenordnung wie beim zweischichtigen Vertikal-Verfahren. Die zusätzliche Verwendung von halber Genauigkeit hat sich demnach nicht sonderlich auf die Ausführungszeit ausgewirkt. Da für kleine Problemgrößen die Zeit pro Level für alle Datentypen nahezu identisch ist, ist dieses Resultat nicht weiter verwunderlich. Der relative Fehler liegt in der Größenordnung von 1,00E–07. Damit erreicht auch das V3SMG-Verfahren nach gleich vielen Iterationen dieselbe Genauigkeit wie das V2SMG- und Mehrgitterverfahren in doppelter

Genauigkeit.

Faizt Auf Grund der Rundungsfehler durch die Verwendung von halber Genauigkeit kann das RR nicht unter 1,00E−07 reduziert werden. Der relative Fehler verhält sich hingegen weitestgehend gleich wie beim herkömmlichen Mehrgitterverfahren. Außerdem lässt sich durch die Verwendung von Half kein zusätzlicher Speed-Up erreichen.

5 Fazit

Die Analyse der gemischt genauen Mehrgitterverfahren hat gezeigt, dass die Verwendung von einfacher und halber Genauigkeit bei allen Verfahren keinen Einfluss auf die Entwicklung des echten Fehlers hat. Um das relative Residuum als Abbruchkriterium verwenden zu können, sollte die Residuumsberechnung auf dem feinsten Gitter in doppelter Genauigkeit erfolgen. Der Verlauf des Residuums verändert sich jedoch stark unter Hinzunahme von halber Genauigkeit. Dadurch ist es beim H3SMG-Verfahren schwierig das Residuum als Abbruchkriterium zu verwenden. Hier könnten Alternativen wie die relative Residuumsrate oder die relative Fehlerrate Abhilfe schaffen. Im Gegensatz zum H2SMG-Verfahren lässt sich das H3SMG-Verfahren auf Grund des Darstellungsbereichs von Half nicht für beliebig große Probleme verwenden.

Bei den gemischt genauen Vertikal-Verfahren hat sich gezeigt, dass es ausreichend ist, wenn das feinste Gitter in doppelter Genauigkeit berechnet wird. Beim V3SMG-Verfahren kann das relative Residuum, wie schon beim H2SMG-Verfahren, nicht zuverlässig als Abbruchkriterium verwendet werden.

Der Speed-Up gegenüber dem herkömmlichen Mehrgitterverfahren in doppelter Genauigkeit beträgt bis zu 25 Prozent. Er hängt stark von der Implementierung ab und wächst mit steigender Problemgröße an. Auf Grund der theoretischen Untersuchungen lässt sich vermuten, dass der Speed-Up deutlich weiter ausgebaut werden kann. Hierfür muss viel Zeit in die Optimierung der Kernel gesteckt werden, sodass die Zwischenspeicher effizient ausgenützt werden können. Ob der zusätzliche Aufwand bei der Implementierung den Speed-Up rechtfertigt, lässt sich a priori nicht eindeutig sagen. Hierbei muss beachtet werden, dass die Implementierung mit dem Datentyp Half technisch aufwändiger ist, da der Datentyp nur auf der Beschleunigerkarte verwendet werden kann. Die Verwendung von C++-Templates im Zusammenhang mit Half stellt dabei ebenfalls als eine Herausforderung dar.

6 Ausblick

Da bei den herkömmlichen Mehrgitterverfahren jede Iteration mit einer verbesserten Startlösung beginnt, können die gemischt genauen Verfahren auch als vorkonditionierte Mehrgitterverfahren betrachtet werden. Der Vorkonditionierer ist in diesem Sinne ein – oder die Kombination mehrerer – Mehrgitterverfahren in einer geringerer Genauigkeit. Diese Interpretation erlaubt die Untersuchung von gemischt genauen Mehrgitterverfahren als Vorkonditionierer für andere Lösungsverfahren. Herkömmliche Mehrgitterverfahren werden heutzutage bereits erfolgreich als Vorkonditionierer eingesetzt. Beispiele hierfür sind die Verwendung eines Mehrgitterverfahrens für ein GMRES-Verfahren [23] oder für ein CG-Verfahren [24].

Über eine ähnliche Argumentation lassen sich gemischte Genauigkeiten für FMG-Verfahren verwenden. Ausgehend vom gröbsten Gitter, auf dem eine Lösung in halber Genauigkeit berechnet wird, kann die Approximation sukzessive nach oben prolongiert werden. Hierbei werden die Mehrgitterzyklen auf den unteren Levels in halber und die restlichen in einfacher Genauigkeit durchgeführt. Sobald das feinste Gitter erreicht ist, wird die Approximation nach Double konvertiert und anschließend ein paar wenige Mehrgitterzyklen durchgeführt.

Die Untersuchung der Rechen- und Datendurchsätze für die implementierten Kernel (vgl. Kapitel 4.2) zeigt einen weiteren Anwendungsbereich für gemischt genaue Mehrgitterverfahren: Die Tatsache, dass für kleine Problemgrößen ein geringer Speed-Up gemessen wird, liegt maßgeblich daran, dass die Ausführungszeiten der V-Zyklen für unterschiedliche Datentypen nahezu identisch sind. Dies resultiert aus den Datendurchsätzen der Kernel, welche sich erst ab einer großen Problemgröße dem gemeinsamen Grenzwert annähern. Für Beschleunigerkarten im mittleren und unteren Preissegment ist zu erwarten, dass sich die Kurven bereits schneller dem Grenzwert annähern. Dies führt dazu, dass bereits die V-Zyklen mit einer mittleren Problemgröße in Single oder Half schneller ausgeführt werden können, als die V-Zyklen in Double. Insgesamt kann dadurch ein höherer Speed-Up erwartet werden.

Neben den Datentypen Double, Single und Half existiert noch der 16 Byte große Datentyp Quadruple. Dieser kann derzeit meist nur als emulierter Datentyp verwendet werden. Die Rechenoperationen sind dadurch deutlich teurer als die in doppelter Genauigkeit. Im Zusammenhang mit gemischt genauen Mehrgitterverfahren könnte dadurch ein deutlicher zeitlicher Vorteil erreicht werden.

Eine vollkommen andere Herangehensweise, um gemischte Genauigkeiten für Mehrgitterverfahren zu verwenden, besteht in einer gemischt genauen Approximation des Differentialoperators. Hierbei kann beispielsweise ein 9-Punkte-Stern verwendet werden, welcher ausschließlich in einfacher Genauigkeit zur Anwendung kommt. Dies betrifft die Implementierung des JACOBI- und DEFECT-Kernels und könnte dadurch weitreichende Auswirkungen auf die Ausführungszeit haben.

A

Nachfolgend werden die Resultate dargestellt, die sich bei der Untersuchung des Mehrgitterverfahrens in doppelter Genauigkeit mit unterschiedlichen Tiefen und einer unterschiedlichen Anzahl an Jacobi-Schritten auf dem kleinsten Level ergeben haben.

In Abbildung A.1 sind oben die Fehlerentwicklungen für eine maximale Tiefe von 5, bei einer Durchführung von drei Jacobi-Schritten dargestellt. Es zeigt sich, dass insgesamt sehr viele Ite-

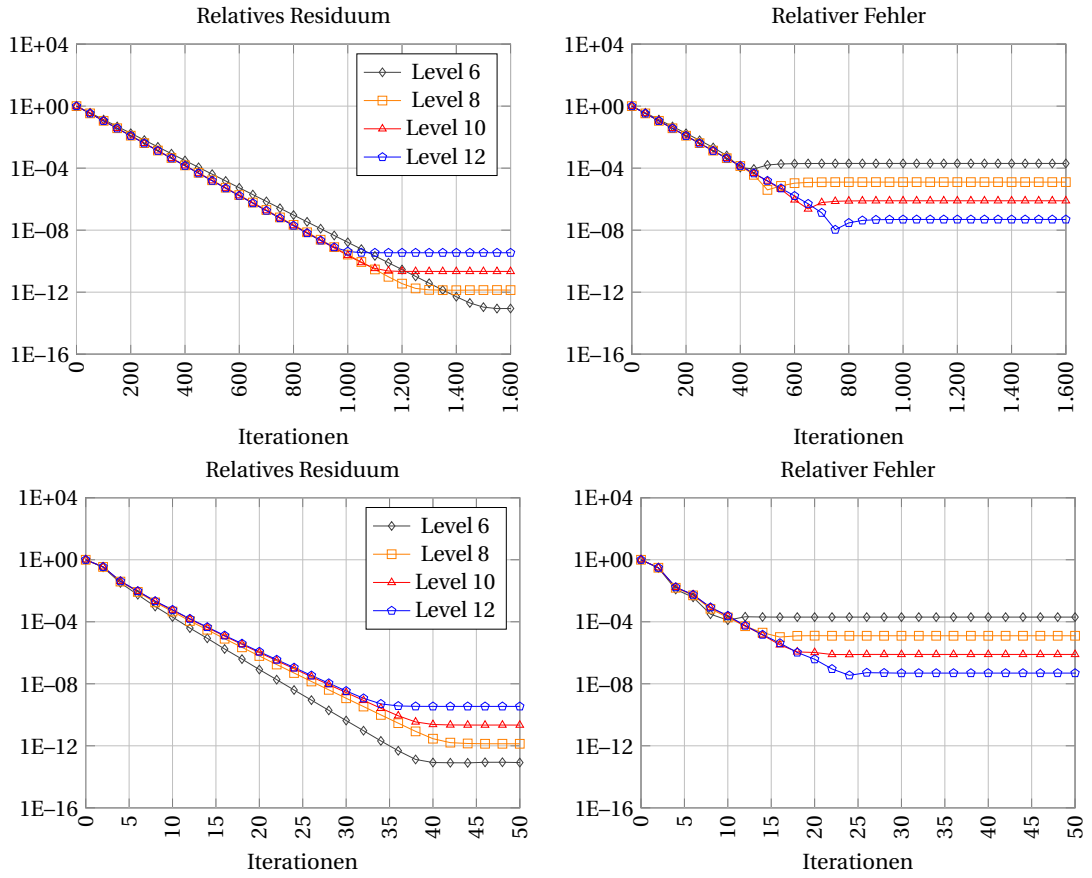


Abbildung A.1: Fehlerentwicklungen des Mehrgitterverfahrens in doppelter Genauigkeit. Bei den beiden oberen Fehlerentwicklungen wird jeweils bis zum 5. Level vergröbert. Bei den unteren bis zum 2. Level. Auf dem gröbsten Gitter werden jeweils drei Schritte des (ungedämpften) Jacobi-Verfahrens durchgeführt.

rationen benötigt werden, bis die Fehler anfangen zu stagnieren. Erst nach ca. 1.500 Iterationen beim relativen Residuum und nach knapp 800 Iterationen beim relativen Fehler haben sich die Werte den jeweiligen Grenzwerten angenähert. Dies liegt vorwiegend an der schlechten Approximation auf dem gröbstem Gitter: Durch die Verwendung von lediglich drei Schritten des Jacobi-Verfahrens beinhaltet die Approximation einen zu hohen Abbruchfehler. Dadurch sind sehr viele Iterationen notwendig, bis dieser Fehler ausreichend reduziert wird.

Im Gegensatz dazu ist unten in Abbildung A.1 das Verfahren mit einer Tiefe von 2 aufgeführt. Es zeigt sich, dass deutlich weniger Iterationen benötigt werden. Bereits nach 40 Iterationen beim relativen Residuum bzw. nach 25 Iterationen beim relativen Fehler verändern sich die Kurven nicht mehr. Es werden zwar nach wie vor lediglich drei Schritte des Jacobi-Verfahrens ausgeführt, allerdings ist die Problemgröße bereits so gering, sodass diese Anzahl ausreicht, um einen vernachlässigbaren Abbruchfehler zu beinhalten. Die Problemgröße auf Level 2 beläuft

sich auf $(2^2+1)^2 = 3$. Für diese Größe weisen die implementierten Kernel allerdings einen sehr geringer Rechen- und Datendurchsatz auf. Die Beschleunigerkarte wird dadurch nicht optimal ausgenutzt. Wird hingegen bis zum 5. Level vergröbert und dann 151 Iterationen des Jacobi-Verfahrens ausgeführt, so zeigt sich ein ähnlicher Fehlerverlauf wie in Abbildung A.1 (vgl. Abbildung 20). Aus diesem Grund wird im Zuge dieser Arbeit nicht bis ganz in die Tiefe vergröbert, sondern bereits ab dem 5. Level das ungedämpfte Jacobi-Verfahren angewandt.

B

Im Folgenden werden unterschiedliche Konfigurationen des V3SMG-Verfahrens näher analysiert. Bei allen werden lediglich die 151 Jacobi-Iterationen auf dem kleinsten Level in halber Genauigkeit durchgeführt. In Abbildung B.1 sind die zugehörigen Fehlerentwicklungen aufgeführt. Es wird zunächst das relative Residuum betrachtet. Hierbei zeigt sich, dass die Verläufe sich nicht voneinander unterschieden werden können. Unabhängig davon, wie groß die Problemgröße ist, wird das relative Residuum in allen Konfigurationen gleich schnell reduziert. Beim relativen Fehler können die Kurven nach knapp 20 Iterationen unterschieden werden.

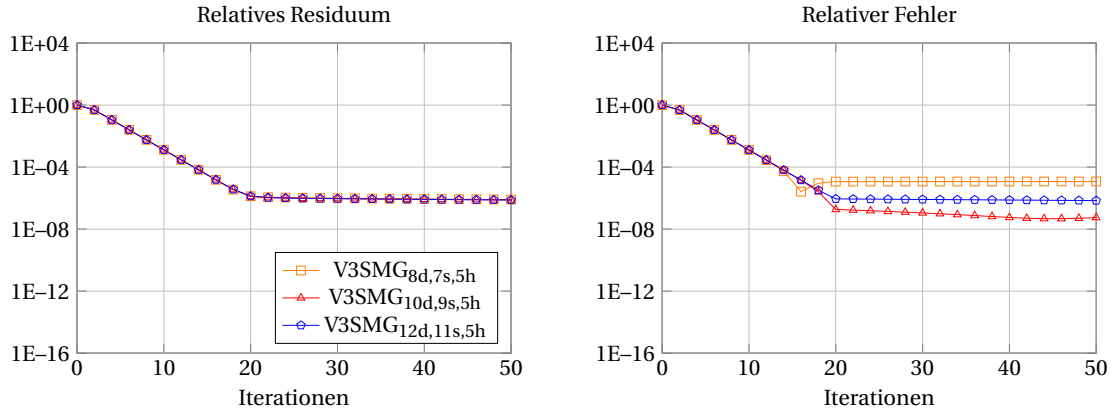


Abbildung B.1: Fehlerentwicklungen und Ausführungszeiten des V3SMG-Verfahrens. Das relative Residuum (links) und der relative Fehler (rechts) sind für unterschiedliche Level-Konfigurationen in Abhängigkeit der durchlaufenden Iterationen aufgetragen.

Die Grenzwerte befinden sich dabei in derselben Größenordnung, wie für die Konfigurationen aus Abbildung 32. Dies zeigt, dass der relative Fehler nicht von der Wahl des Levels n_h abhängt. Aus diesem Grund kann das Level zum Wechseln von Single auf Half so groß gewählt werden, wie der Darstellungsbereich von Half dies zulässt.

C

Nachfolgend wird die Auswirkung einer unstetigen rechten Seite auf die gemischt genauen Verfahren untersucht. Hierbei wird als rechte Seite die Funktion

$$\mathrm{DC}(x,y) = \begin{cases} 0, & \text{für } (x,y) \in \left(\left[0,1\right] \times \left[0,1\right]\right) \setminus \left(\left[\frac{1}{4},\frac{3}{4}\right] \times \left[\frac{1}{4},\frac{3}{4}\right]\right) \\ 2, & \text{für } (x,y) \in \left(\left[\frac{1}{4},\frac{3}{4}\right] \times \left[\frac{1}{4},\frac{3}{4}\right]\right) \setminus \left(\left[\frac{3}{8},\frac{5}{8}\right] \times \left[\frac{3}{8},\frac{5}{8}\right]\right) \\ 4, & \text{für } (x,y) \in \left[\frac{3}{8},\frac{5}{8}\right] \times \left[\frac{3}{8},\frac{5}{8}\right] \end{cases}$$

verwendet. In Abbildung C.1 ist $DC(x,y)$ in Form eines Surfplots dargestellt. Da auf Grund der

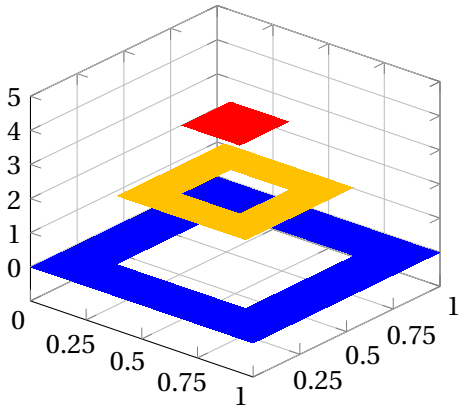


Abbildung C.1: 3D-Darstellung der unstetigen Rechten Seite DC(x, y)

Unstetigkeit die Lösung nicht eindeutig sein muss, wird im Folgenden nur die Entwicklung des relativen Residuums betrachtet. In Abbildung C.2 ist auf der linken Seite der Verlauf des relativen Residuums für ein herkömmliches Mehrgitterverfahren in halber, einfacher und doppelter Genauigkeit aufgeführt. Das Subskript kennzeichnet dabei das Level des Ausgangsproblems. Es zeigt sich, dass bei Verwendung von halber Genauigkeit das Residuum wächst und gegen

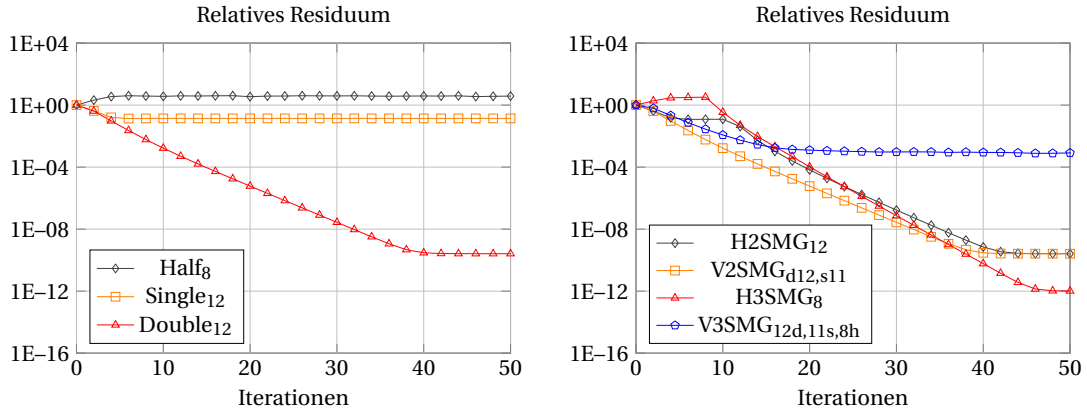


Abbildung C.2: Links: Entwicklung des relativen Residuums beim Mehrgitterverfahren in doppelter, einfacher und halber Genauigkeit für unterschiedliche Problemgrößen. Rechts: Verlauf des relativen Residuum für die gemischt genauen Verfahren. Als rechte Seite wurde DC(x, y) verwendet.

einen Wert von knapp 3,91E+00 konvergiert. Bei Verwendung von einfacher Genauigkeit wird das RR zwar reduziert, allerdings nur bis zu einem Wert von knapp 1,39E–01. Lediglich bei Verwendung von doppelter Genauigkeit wird das Residuum deutlich minimiert. Verglichen mit den Verläufen bei einer stetigen rechten Seite (Abbildung 20, 21 und 22) liegt der Fehler in derselben Größenordnung. Die Regularität hat damit keinen Einfluss auf den Verlauf des relativen Residuums beim herkömmlichen Mehrgitterverfahren. Dies gilt zumindest, für die in dieser Arbeit betrachteten Probleme. Auf der rechten Seite sind in Abbildung C.2 die Verläufe des Residuums für die gemischt genauen Verfahren dargestellt. Das Subskript bei den Horizontal-Verfahren beschreibt dabei ebenfalls das Level des Ausgangsproblems. Auch bei den gemischt genauen Verfahren zeigt sich, dass die Verläufe keine wesentlichen Unterschiede zu den Verläufen mit einer stetigen rechten Seite aufweisen (vgl. Abbildung 26, 28, 30 und 32). Lediglich beim V3SMG-Verfahren liegt der gemessene Wert um ca. 7,3E–04 höher. Allerdings befinden sich beide Grenzwerte in derselben Größenordnung, weshalb der Unterschied durch Rundungsfehler erklärt werden kann.

Literatur

[1] Jinn-Liang Liu. Poisson's Equation in Electrostatics. http://www.nhcue.edu.tw/~jinnliu/proj/Device/3DPoisson.pdf. Abgerufen: Juni 2019.

[2] IEEE Computer Society. IEEE Standard for Floating-Point Arithmetic. http://www.dsc.ufcg.edu.br/~cnum/modulos/Modulo2/IEEE754_2008.pdf. Abgerufen: Juni 2019.

[3] Thomas Jahn. "Implementierung numerischer Algorithmen auf CUDA–Systemen". Abgerufen: Juni 2019. Diplomarbeit. Universität Bayreuth.

[4] NVIDIA Corporation. NVIDIA CUDA - Programming Guide - Version 1.0. http://developer.download.nvidia.com/compute/cuda/1.0/NVIDIA_CUDA_Programming_Guide_1.0.pdf. Abgerufen: Mai 2019.

[5] The Khronos Group Inc. Khronos OpenCL Registry. https://www.khronos.org/registry/OpenCL/. Abgerufen: Juni 2019.

[6] Dominik Göddeke. "Fast and Accurate Finite-Element Multigrid Solvers for PDE Simulations on GPU Clusters". Diss. Technische Universität Dortmund, Feb. 2010.

[7] NVIDIA Corporation. Whitepaper - CUDA™Compute Architecture: Fermi™. https://www.nvidia.com/content/PDF/fermi_white_papers/NVIDIA_Fermi_Compute_Architecture_Whitepaper.pdf. Abgerufen: Februar 2019.

[8] NVIDIA Corporation. NVIDIA Tesla V100 GPU Architecture. https://images.nvidia.com/content/volta-architecture/pdf/volta-architecture-whitepaper.pdf. Abgerufen: Februar 2019.

[9] James H. Wilkinson. Rundungsfehler. pub-Springer:adr-B: pub-Springer, 1969.

[10] Cleve B. Moler. "Iterative Refinement in Floating Point". In: J. ACM 14.2 (Apr. 1967), S. 316–321. ISSN: 0004-5411. DOI: 10.1145/321386.321394. URL: http://doi.acm.org/10.1145/321386.321394.

[11] Alfredo Buttari et al. "Mixed Precision Iterative Refinement Techniques for the Solution of Dense Linear Systems". In: The International Journal of High Performance Computing Applications 21.4 (2007), S. 457–466. DOI: 10.1177/1094342007084026. URL: https://doi.org/10.1177/1094342007084026.

[12] Alfredo Buttari et al. "Using Mixed Precision for Sparse Matrix Computations to Enhance the Performance while Achieving 64-bit Accuracy". In: ACM Transactions on Mathematical Software 34 (Juli 2008). DOI: 10.1145/1377596.1377597.

[13] Dominik Göddeke, Robert Strzodka und Stefan Turek. "Performance and accuracy of hardware-oriented native-, emulated- and mixed-precision solvers in FEM simulations". In: International Journal of Parallel, Emergent and Distributed Systems 22.4 (2007), S. 221–256. DOI: 10.1080/17445760601122076. URL: https://doi.org/10.1080/17445760601122076.

[14] Dominik Göddeke. Wissenschaftliches Rechnen. Vorlesungsskript. März 2017.

[15] Wolfgang Hackbusch. Multi-grid methods and applications. Springer, 1985.

[16] NVIDIA Corporation. NVIDIA TESLA V100 GPU ACCELERATOR. https://images.nvidia.com/content/technologies/volta/pdf/tesla-volta-v100-datasheet-letter-fnl-web.pdf. Abgerufen: Juni 2019.

[17] NVIDIA Corporation. NVIDIA TESLA V100 GPU ACCELERATOR. https://www.anandtech.com/show/12576/nvidia-bumps-all-tesla-v100-models-to-32gb. Abgerufen: Juni 2019.

[18] NVIDIA Corporation. NVIDIA Tesla V100 GPU Architecture. https://images.nvidia.com/content/volta-architecture/pdf/volta-architecture-whitepaper.pdf. Abgerufen: September 2019.

[19] Dominik Göddeke und Malte Shirwon. GPU Programming with CUDA. Doktorandenkurs. Sep. 2016.

[20] Martin Bernreuther. High Performance Computing. Vorlesungsskript. Juli 2019.

[21] NVIDIA Corporation. CUDA C Programming Guide - Design Guide. https://docs.nvidia.com/pdf/CUDA_C_Programming_Guide.pdf. Abgerufen: September 2019.

[22] NVIDIA Corporation. CUBLAS Library - User Guide. https://docs.nvidia.com/cuda/pdf/CUBLAS_Library.pdf. Abgerufen: August 2019.

[23] T. Washio C.W. Oosterlee. On the Use of Multigrid as a Preconditioner. https://pdfs.semanticscholar.org/dcc1/9ad91450753e7f47157e323fade5c2b4e320.pdf. Abgerufen: Juli 2019.

[24] Osamu Tatebe. The Multigrid Preconditioned Conjugate Gradient Method. http://www.hpcs.cs.tsukuba.ac.jp/~tatebe/research/paper/CM93-tatebe.pdf. Abgerufen: Juli 2019.