Scientific Computing

Masterarbeit

PISM Performance Profiling – Analyse einer Eisschild Simulation

Alexander Van Craen

Studiengang:	Informatik
Prüfer/in:	Prof. Dr. rer. nat. Dirk Pflüger
Betreuer/in:	M.Sc. Gregor Daiß
Beginn am:	15. Juni 2020
Beendet am:	15. Dezember 2020

Kurzfassung

In der vorliegenden Masterarbeit wird der aktuelle Performancestand des Parallel Ice Sheet Model (PISM) evaluiert. Eine allgemeine Übersichtsanalyse ergibt, dass die Performance von PISM sowohl durch die Speicherbandbreite, also auch Netzwerkbandbreite und -latenz limitiert ist. Auf dieser Analyse aufbauend, werden ebenfalls mittels, sowohl automatisch generierten, als auch theoretisch hergeleiteten, Roofline Modellen, sowie Laufzeitmessungen einzelner Programmabschnitte die laufzeitkritischen Abschnitte identifiziert und analysiert. Für die herauskristallisierten Flaschenhälse und Hotspots, werden verschiedene Verbesserungsansätze dargelegt und diskutiert. Die detaillierte Analyse der Stressgleichungen ergibt beispielsweise eine mögliche Laufzeitreduzierung eben dieser um über 80 %. Anhand einer Implementierung wird die Umsetzbarkeit ausgewählter Verbesserungsansätze überprüft.
Die Ergebnisse dieser Masterarbeit sind dabei von primärem Interesse für die PISM Entwickler. Das Vorgehen in dieser Arbeit ist exemplarisch für eine Performanceanalyse großer und kleiner Projekte und kann damit als Grundlage für weitere Analysen dienen, sodass die Masterarbeit auch für Leute interessant ist, die sich mit der Performance von wissenschaftlichem Programmcode auseinandersetzen.

Danksagung

Zu aller erst möchte ich mich bei Prof. Dirk Pflüger und Gregor Daiß bedanken, die mir durch ihr Vertrauen die Herausforderung an so einem interessanten Thema ermöglicht, und durch ihr kritisches Hinterfragen und ihre Motivation die Arbeit bereichert haben. Auch geht ein großes Dankeschön an Moritz Kreuzer, Torsten Albrecht, Constantine Khrulev, Brigitta Krukenberg, Andy Aschwanden und Ricarda Winkelmann, die mir ausführlich meine Fragen zur Eissimulation im Allgemeinen und zu PISM im Speziellen beantwortet haben, hilfreiche Anregungen und Fragen hatten sowie die Rechenzeit auf dem PIK-Cluster ermöglicht haben. Ich danke allen, die sich die Zeig genommen haben, um einzelne Abschnitte oder sogar die ganze Arbeit Korrektur zu lesen. Außerdem bedanke ich mich für die Unterstützung des Landes Baden-Württemberg durch bwHPC, und damit für die zur Verfügungsstellung von Rechenzeit auf einem Tier 3 Computer-Cluster.

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

Verzeichnis der Listings

Verzeichnis der Algorithmen

Abkürzungsverzeichnis

ALE Arbitrary Lagrangian-Eulerian. 22

CPU Central Processing Unit (engl. Prozessor). 41

FD finiten Differenzen. 22

FE finiten Elemente. 22

FLOPs Floating Point Operations Per Second (engl. Gleitkommaoperationen pro Sekunde). 35

FMA Fused Multiply Add. 42

GPU Graphics Processing Unit (engl. Grafikprozessor). 23

MPI Message Passing Interface [For15]. 19

PETSc Portable Extensible Toolkit for Scientific Computing. 9

PISM Parallel Ice Sheet Model. 19

SIA Shallow Ice Approximation. 22

SMT Simultaneous Multithreading. 46

SSA Shallow Shelf Approximation. 23

1 Einleitung

Der Klimawandel ist eines der Schlüsselprobleme, mit denen die Menschheit aktuell konfrontiert wird. Der daraus resultierende Anstieg des Meeresspiegels wird die Küstenlinien auf der ganzen Welt prägen. Die beiden letzten Eisschilde der Erde liegen auf der Insel Grönland, sowie dem Kontinent Antarktika und weitere Teile der Antarktis. Zusammen bestehen diese Eisschilde aus $3 \times 10^7\,\mathrm{km}^3$ Eis [FPV+13; MWR+17]. Würde dieses Eis komplett abschmelzen, hätte dies verheerende Folgen. Fretwell et al. gehen davon aus, dass bei einem kompletten Abschmelzen des antarktischen Eisschildes die Wassermassen einen Meeresspiegelanstieg von 58 m nach sich ziehen würden [FPV+13]. Bei einem kompletten Abschmelzen des grönländischen Eisschildes, gehen Morlighem et al. von einem zusätzlichen Anstieg von 7,4 m aus [MWR+17]. Das würde dazu führen, dass sich die Küstenlinie so weit zurückgedrängt würde, dass man beispielsweise vom Kölner Dom aus ins Meer springen könnte.

Jedoch verursacht nicht einzig der Anstieg des Meeresspiegels enorme Probleme für die Menschheit, auch die riesigen Wassermengen, die beim Schmelzen in Bewegung kommen, bergen Gefahren. Die Eisschilde speichern über 70 % des globalen Süßwassers [Rig11]. Ein Abschmelzen würde zu einer Abnahme des Salzgehalts im Meerwasser führen und somit dessen Dichte senken. Das saline Gleichgewicht ist in dieser Region allerdings enorm wichtig. Ein Beispiel hierfür ist die Tiefenströmung im Nordatlantik, deren Voraussetzung in einem hohen Salzgehalt des Gewässers besteht. Die Antarktis ist ein Motor der globalen thermohaline Zirkulation [LOC95]. Die Verdünnung des Meerwassers durch Wasser aus dem grönländischen Eis, und der damit verbundene abnehmende Salzgehalt, hat die Verlangsamung des atlantischen Teils der globalen thermohaline Zirkulation zur Folge. Zwischen Mitte des 20. Jahrhunderts und dem Jahr 2018 hat sich die Fließgeschwindigkeit der Nordatlantikströmung nach einer Veröffentlichung von Caesar et al. um 15 % verringert [CRR+18]. Diese Veränderungen haben das Potenzial, das Weltklima aus dem Gleichgewicht zu bringen und damit gravierend zu beeinflussen [SRR+18].

Damit sollte die Wissenschaft und die gesamte Menschheit das Verhalten der Eisschilde mit großem Interesse beobachten. Um das zukünftige Verhalten, oder das Verhalten unter bestimmten Bedingungen untersuchen zu können, werden Programme, wie das Parallel Ice Sheet Model (PISM) benötigt, die Eisschildsimulationen durchführen können. Diese Simulationen sind dabei nicht nur ausschlaggebend für die Abschätzung des Meeresspiegelanstiegs und die damit einhergehende Neuformung der Küstenlinien, sie sind auch unerlässlich als Grundlage für komplexe Klimasimulationen. Beides dient als Analysegrundlage, um mögliche Handlungsoptionen für die Herausforderungen des Klimawandels evaluieren zu können. Um bei einer kontinental-skaligen Simulation eine hinreichende Auflösung zu ermöglichen, ist die Voraussetzung an solche Eisschildsimulationen eine effiziente und parallele Ausnutzung aktueller Hardware. Parallel Ice Sheet Model (PISM) verwendet dafür PETSc, welches auf dem Message Passing Interface [For15] (MPI) basiert, um die Simulationen zu parallelisieren und die Arbeitslast auf mehrere Rechner zu verteilen. Bessere Performance ermöglicht dabei nicht nur die Simulation in höherer Auflösung, welche für eine aussagekräftige Analyse unerlässlich sind, sondern trägt auch zur allgemeinen Beschleunigung der Berechnungen

von aktuellen Szenarien bei. Damit ermöglicht sie die Berechnung von großen Ensembles. Aufgrund der chaotischen Natur des Wetters, und den vielen Parametern und Unbekannten, sind diese nötig, um ein zuverlässiges Risikomaß liefern zu können.

Ziel dieser Arbeit ist es deshalb, das Leistungsverhalten von PISM zu analysieren, und Möglichkeiten zur Leistungsverbesserung aufzuzeigen. Außerdem werden diese Möglichkeiten auf ihr tatsächliches Potenzial der Leistungsverbesserungen hin untersucht.

Die Arbeit beginnt mit einem Vergleich von PISM mit anderen Eisschild Simulationsprogrammen in Kapitel 2 „Verwandte Arbeiten“, gefolgt von Kapitel 3 „Grundlagen“. Die Grundlagen geben eine detaillierte Einführung in PISM und PETSc. Darauf folgt eine Erläuterung des Szenarios, auf welchem die anschließenden Analysen basieren sowie die Beschreibung der, für diese Analysen genutzten, Hardware. Das Grundlagenkapitel wird, nach einer Übersicht über die genutzten Profilingprogramme, mit einer Unterscheidung von Skalierungstypen abgeschlossen. Im nächsten Kapitel folgen Untersuchungen des allgemeinen Leistungsverhaltens. Im Speziellen wird hier eine allgemeine Laufzeitanalyse durchgeführt, auf den verbrauchten Hauptspeicher eingegangen sowie die Menge der geschriebenen Ergebnisse betrachtet. Kapitel 5 „Hotspot Analysen“ beschäftigt sich, wie der Name schon vermuten lässt, mit den in Kapitel 4 herauskristallisierten laufzeitintensiven Codeabschnitten. Hier wird analysiert, was genau die limitierenden Faktoren sind und wie viel Verbesserungspotential in den Hotspots besteht. In Kapitel 6 „Verbesserungsvorschläge“ werden Strategien diskutiert, womit die aktuelle Performance verbessert werden kann. Im Weiteren folgt in Kapitel 7 „Verbesserungsansatz“ die Beschreibung erster Umsetzungsversuche einiger Verbesserungsansätze. Abschließend folgt ein Fazit in Kapitel 8.

2 Verwandte Arbeiten

Die Simulationsmodelle für Eisschilde unterstehen einem stetigen Wandel. Da die Eisschildsimulation eine physikalisch komplexe Simulation ist, die dazu noch ein großes Gebiet nachbildet, ist sie nur über zum Teil große Vereinfachungen und Annahmen möglich. Infolge dessen haben sich über die Zeit sehr viele unterschiedliche Bibliotheken, wie z. B. GLIMMER [TZS+08], Glimmer[1], CISM[2], GLAM [LPH+19], SICPOLIS[3], IcIES [GSAO11], Elmer/Ice[4], GRISLI [QDR+18], AIF [WZH+03], ISSM[5], University of Maine Ice Sheet Model[6] [Fas94], BISICLES[7], f.ETISH [Pat17] mit unterschiedlichen Modellvereinfachungen gebildet. Des Weiteren gibt es Veröffentlichungen von anderen Modellen, die noch nicht in öffentlich zugängliche Bibliotheken implementiert wurden, wie zum Beispiel das Modell der Pennsylvania State University [FWP+11].

Mit ansteigender Hardwareperformance und zunehmend leistungsfähigeren Höchstleistungsrechenzentren ergeben sich kontinuierlich neue Möglichkeiten. So entstehen einerseits Abspaltungen von Bibliotheken, weil manche Szenarien auf eine neue Art und Weise simuliert werden können, die mit der alten Hardware nicht umsetzbar war. Andererseits fusionieren Bibliotheken auch wiederum, weil, bedingt durch stärkere Hardware, es möglich wird komplexere Modelle miteinander zu kombinieren. Im Folgenden sind einige Beispiele dafür aufgeführt.
Das ANICE-Modell, als Teil des IMAU-ICE (Institute for Marine and Atmospheric Research Utrecht) Eisschildmodellpaketes [BDB+15], entwickelte sich aus dem Regional Atmospheric Climate Modell RACMO [MVUB+08].
Das GLIMMER-Modell hat sich aus dem GENIE-Erdsystemmodell heraus gebildet [LFV+13; LMP+07] und besteht aus mehreren Bibliotheken, die zur Simulation der Eisschildentwicklung verwendet werden können [RHHP09]. Es kann unabhängig oder als Untermodul des Earth System Modeling Frameworks (ESM)[8] für unterschiedliche Simulationen genutzt werden [TZS+08]. Aus diesem Framework hat sich das Community Earth System Modeling Framework (CESM) abgespalten [LPH+19]. Das ursprüngliche Glimmer-Modell berechnet Masse, Impuls und Energieerhaltung für Eisschilde und Gletscher unter Verwendung einer seriellen, thermomechanisch gekoppelten, flachen Eisdarstellung der Eisdynamik.
In CESM kommt das Community Ice Sheet Model (CISM)[9] als Modul zur Land-Eis-Simulation zum Einsatz. CISM erweitert das ursprüngliche Glimmer-Modell durch das Hinzufügen von Näherungen höherer Ordnung für die Impulsbilanz, skalierbare Parallelität, erweiterte Unterstützung

[1] https://github.com/glimmer-cism
[2] https://cism.github.io/
[3] http://www.sicopolis.net/, https://gitlab.awi.de/sicopolis/sicopolis/
[4] https://elmerice.elmerfem.org/
[5] https://issm.jpl.nasa.gov/
[6] http://www2.umaine.edu/climatechange/Research/Contrib/html/15.html
[7] https://commons.lbl.gov/display/bisicles/BISICLES
[8] https://www.esm-project.net/
[9] https://cism.github.io/

von Drittanbieter-Löser-Bibliotheken und zusätzliche Physik [LPH+19]. CISM rechnet dabei auf einem regulären Gitter unter Verwendung einer Mischung aus Finite-Differenz-, Finite-Volumen- und Finite-Elemente-Methoden und ist hauptsächlich in Fortran 90 geschrieben [LS12]. Für die Gleichungen höherer Ordnung nutzt CISM jedoch nicht die Gleichungen der Glimmer-Modelle, sondern eine parallelisierte und mit Trilinos[10] beschleunigte Version des CISMs Dycore [HBH+05], auch bekannt als Glam [ESW+12].

Die Bibliotheken basieren jedoch keineswegs alle auf dem gleichen zugrundeliegenden Modellansatz, sondern unterscheiden sich teilweise stark in den Modellen, mit denen das Eisschild simuliert wird. Sie unterscheiden sich in der Art und Weise, wie das Rechengebiet aufgeteilt wird. Hier besteht ein Unterschied, sowohl in der Dimensionalität (2D/3D), als auch in der zum Teil adaptiven Gitterstruktur, die gleichmäßig, oder variabel ist. Auf diesen Gittern wird je nach Bibliothek mit den Methoden der finiten Differenzen (FD) (z.B. Glimmer, GLAM, IcIES, GRISLI, AIF), oder der Methode der finiten Elemente (FE) (z. B. MAINE, Elmer/Ice, ISSM) gerechnet. Zum Teil werden diese Methoden mit weiteren wie der Euler Methode (z.B. GRISLI, Elmer/Ice) kombiniert, so auch bei PISM. In PISM kann gewählt werden, ob eine Simulation basierend auf einer FD Picard Grundlage oder FE Euler Grundlage durchgeführt werden soll. ISSM koppelt zu den FE die Arbitrary Lagrangian-Eulerian (ALE) Methode, also eine Kopplung der eulerschen mit der lagrangeschen Formulierung.

Als Grundlage für die Flussgleichungen dienen bei den meisten Bibliotheken Modelle, die auf Stokes-Stress-Modellen basieren und durch Shallow Ice Approximation (SIA) vereinfacht werden ([BB09b; Bas10; EKCL11; Gol11; PD12; PGB12; SH10; TPS+15]). Die zugrundeliegenden vollständigen Stokes-Modelle ([BJ13; GZ08; GZGC+13; ISG15; JPRB08; Jar08; LJG+12; LJGP14; LSMR12]) liefern eine vollständige mechanische Beschreibung der Verformung. Eine, neben weiteren Unterscheidungsmerkmalen, zusätzliche Herausforderung in vollständigen Stokes-Modellen, ist die nicht lineare Thermomechanik von Eis [ABKB12], die von den Modellen mit unterschiedlichen Konzepten angegangen wird [BGAO10]. In drei Dimensionen stellen die Berechnungen der vollständigen Stokes-Gleichungen allerdings einen sehr hohen Bedarf an Rechenressourcen dar, und schränken so die Gebietsauflösung ein.
Durch eine Kombination mit der SIA wird das Modell vereinfacht, indem folgende Phänomene vernachlässigt werden: Längsspannungen entlang der Fließdehnung, Kompression und Querspannungen; seitlicher Widerstand gegen langsameres Eis für einen Eisstrom oder Talwände für einen Talgletscher; sowie vertikale Spannungsgradienten [AM12; MC98]. Das SIA ist dabei ein vertikal integriertes (2D) Modell mit einer tiefen-gemittelten Eisgeschwindigkeit. Die Reduktion der Dimensionalität sorgt für eine starke Reduktion der Rechenintensität. Die Verwendung der SIA in der Eisschildmodellierung geht dabei auf die Arbeiten von Mahaffy [Mah76] und das thermo-mechanische Modell von Jenssen [Jen77] zurück. Die Anwendbarkeit der SIA wurde in der Literatur ausführlich diskutiert (z. B. [HM01; Hut83; Pat03]). Obwohl es keine Darstellung von Spannungen höherer Ordnung im Eis gibt, hat sich gezeigt, dass SIA im Vergleich zu vollen Stokes-Stress-Modellen in einer Vielzahl glaziologischer Situationen vergleichbare Ergebnisse liefert [VG04]. Elmer/Ice ist eine der wenigen Bibliotheken die aktuell noch rein auf den Stokes Glei-

[10] https://trilinos.github.io/

chungen basiert [CLS20]. ISSM greift zusätzlich zu SIA auf eine Auswahl der Stokes Gleichungen zurück.

PISM wiederum nutzt keinen reinen SIA Ansatz, sondern als ein hybrides Strömungsgesetz, eine Konkatenation von SIA und eine Shallow Shelf Approximation (SSA) [BKA+09]. Die SSA wurde entwickelt, um Schelfeisbewegungen zu modellieren, bei denen die basale Scherspannung gleich null ist und die Längsspannungen dominieren [BB09a]. Die SSA ist ein vertikal integriertes (2D) Modell mit einer tiefen gemittelten Eisgeschwindigkeit. Es ist daher nicht möglich sie für Regionen zu implementieren, in denen vertikale Geschwindigkeitsschwankungen von Bedeutung sind, wie z. B. über Grundlinien oder komplexe Eisströmungen in der Nähe von Gletscherspalten [LSMR12]. Hier kommt PISM ohne Gleichungen höherer Ordnung aus, auf die die anderen Modelle zurückgreifen müssen, um ähnliche Genauigkeiten zu erreichen.

Weitere Unterschiede, in der Modellierungsgrundlage sowie das unterschiedliche Verhalten dieser in bestimmten Szenarien, zu untersuchen, haben sich unter anderem SeaRISE[11] [BNAO+13; NBAO+13a; NBAO+13b] und das Ice Sheet Model Intercomparison Project for CMIP6 (ISMIP6)[12] [GNE+18; SNS+19] zur Aufgabe gemacht.

Der vierte Bericht des Weltklimarates [SQM+07] bemängelt allerdings, dass die existierenden Eisschild-Strömung-Modelle den polaren Eisschildabfluss nicht genau beschreiben [GZGC+13], da sie aufgrund der Modellvereinfachungen, oder der benötigten Rechenressourcen nicht in der Lage sind, gleichzeitig langsame und schnelle Eisströme in hinreichender Auflösung zu modellieren [BB09b; GZGC+13]. Eine Möglichkeit, trotz der deutlich rechenintensiveren Lösung der kompletten Stokes Gleichungen eine nutzbare Auflösung zu erhalten, ist der Einsatz von Beschleunigerkarten wie Graphics Processing Units (engl. Grafikprozessoren) (GPUs) [BE12; SG16]. Auch andere Modelle, wie die auf SIA basierenden [BDE14] oder pseudo-transiente Modelle [RLH+19], lassen sich damit beschleunigen. Die PISM Implementierung profitiert aktuell jedoch nicht von Beschleunigerkarten.

Performance Vergleiche zwischen den unterschiedlichen Modellen sind nicht aussagekräftig, bedingt durch die unterschiedlichen zugrundeliegenden Annahmen. Auch sind sowohl die räumliche, als auch die zeitliche Auflösung nur bedingt miteinander vergleichbar, da für die Modelle unterschiedliche Vereinfachungen verwendet werden, wodurch diese nicht zwingend dieselben physikalische Phänomene abbilden.

Über das Performanceverhalten von PISM gibt es aktuell noch keine Veröffentlichungen.

[11] http://websrv.cs.umt.edu/isis/index.php/SeaRISE_Assessment
[12] http://www.climate-cryosphere.org/mips/ismip6

3 Grundlagen

Dieses Kapitel beschäftigt sich mit den Grundlagen, die für die Performance Analyse benötigt werden. Dafür wird zuerst eine Übersicht über das zu analysierende Parallel Ice Sheet Model (PISM) gegeben. Darauf folgt ein Einblick in PETSc, welches PISM als Parallelisierungsgrundlage dient. Nach dem Simulationsprogramm folgt das Szenario, welches als Basis für die Analysen dieser Arbeit dient. In Abschnitt 3.4 werden die Eigenschaften der Hardware, auf welcher die Analysen durchgeführt wurden aufgezeigt. Darauf folgen die genauen Softwareversionen, bezogen auf die Hardware, mit denen simuliert wurde, gefolgt von einer Beschreibung der genutzten Analyseprogramme. Abgeschlossen wird das Kapitel der Grundlagen mit unterschiedlichen Skalierungsdefinitionen.

3.1 Parallel Ice Sheet Model - PISM

Das Parallel Ice Sheet Model (PISM) wurde entwickelt, um Eisschildmodellierung von heutigen, Paläoeisschilden und zukünftigen Szenarien auf der Größe von ganzen Kontinenten durchzuführen. Das heißt, es wird der zeitliche Verlauf von unterschiedlichen Klima bezogenen Parametern der jeweiligen Region berechnet. Die von PISM berechneten Parameter sind hier zum Beispiel die Temperatur, Dichte und Geschwindigkeiten unterschiedlicher Medien, die Eisdicke sowie das Alter des Eises in jedem Datenpunkt der mit Eis bedeckt ist, die Meerestemperatur, die Schmelzmenge, der gesamten Eisfläche, oder auch die Beschaffenheit des Untergrundes. Dieser hebt sich zum Beispiel an oder senkt sich ab, abhängig von dem Gewicht des Eises, welches auf ihm lastet.

PISM ist dabei ein C++ -Eisschildmodellierungscode, der für die Skalierung und Parallelisierung das PETSc [BAA+19; BGMS97] einsetzt. Dies ermöglicht es PISM, hochauflösende Simulationen nichtlinearer Eisströmungen im eisbedeckten Teil eines rechteckigen Rechengebietes parallel auf großen Supercomputern auszuführen [BB09b]. Die Entwicklung sowohl der Temperatur als auch des Wasseranteils im Eis, wird durch den Energiezustand unter Verwendung eines auf polythermaler Enthalpie basierenden Schemas, erfasst [ABKB12].
Eisschildanfangsbedingungen sind typischerweise nicht einfach zu ermitteln [DG11]. Zum Beispiel ist es mit aktuellen Messverfahren nicht möglich in einer feinen Auflösung die Wassermenge sowie deren thermische Energie unter dem Eisschild oder die initiale Strömungsgeschwindigkeit in jedem Punkt zu bestimmen. Die Eisschildanfangsbedingungen, die in einem realen Eisschild nicht messbar sind, werden in der sogenannten „Bootstrapping“ Phase extrapoliert. Während des Bootstrappings wird ein Modelllauf unter Verwendung beobachtbarer Größen wie Eisdicke und Oberflächentemperatur initiiert, während Heuristiken verwendet werden, um die Temperaturen in der Tiefe auszufüllen und die basalen Gleitbedingungen abzuschätzen. Nach dem Bootstrapping wird ein langer Spin-up-Lauf (typischerweise zehntausende von Jahren) durchgeführt, damit sich das Eis zu einem physikalisch stabileren Zustand entwickeln kann, der ein kompatibles Temperaturfeld, Altersfeld und Geschwindigkeiten aufweist. Dieses Verfahren erlaubt es, alle Anfangsbedingungen im Eis zu erhalten, die für eine Simulation verwendet werden. Darauf folgt dann die eigentliche

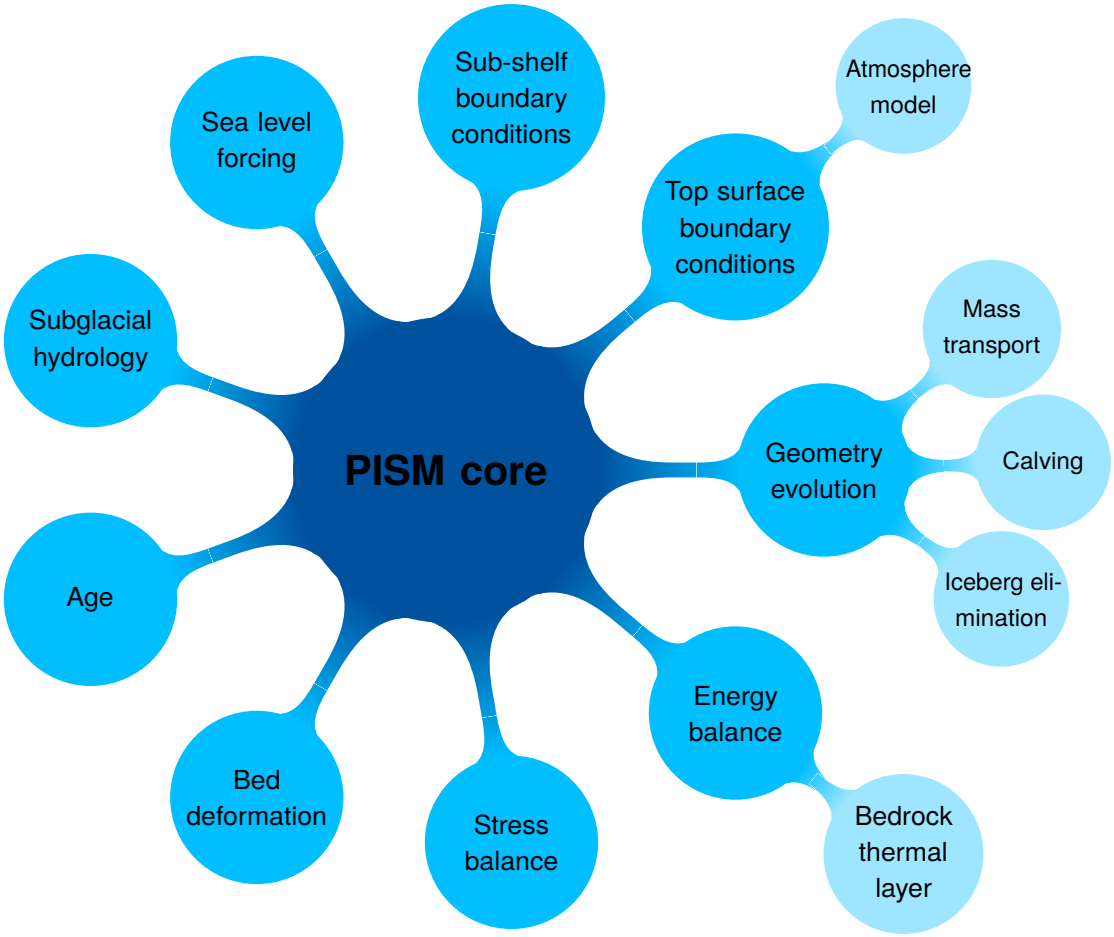


Abbildung 3.1: Übersicht der einzelnen PISM Abschnitte.
Quelle: In Anlehnung an [Khr19, S. 2]

Simulation des Szenarios.
Für eine genaue Modellbeschreibung siehe „The Potsdam Parallel Ice Sheet Model (PISM-PIK) Part 1: Model description“ [WMH+11] und „The Potsdam Parallel Ice Sheet Model (PISM-PIK) – Part 2: Dynamic equilibrium simulation of the Antarctic ice sheet“ [MWH+11] sowie *PISM, a Parallel Ice Sheet Model* [Pa15].

Wie in Abbildung 3.1 schematisch dargestellt, ist PISM zum Teil modularisiert. Zusätzlich zu einem Hauptmodul, dem sogenannten PISM Core Modul, welches den eigentlichen Modellkern enthält und für die Simulation verantwortlich ist, sind weitere Module implementiert, die zum Beispiel zusätzliche Randwerte ermöglichen oder Modellverfeinerungen darstellen und zum eigentlichen Modellkern dazu- oder abgeschaltet werden können. Bei einer ersten Analyse werden die einzelnen Module von PISM betrachtet, um herauszufinden in welchen die Hotspots liegen. Nachfolgend wird dafür eine Übersicht über diese Module gegeben.

3.1.1 PISM core

Das Core Module dient als Schnittstelle zwischen den einzelnen Modulen. Hier ist der Ablauf der Kompletten Simulation implementiert. Zu Beginn, startet das Core Modul beispielsweise das Einlesen des Ausgangszenarios. Hier ist das eigentliche Zeitschrittverfahren implementiert, von welchem aus dann die einzelnen eingeschalteten Module aufgerufen werden. Zudem sind an dieser Stelle die noch nicht vollständig modularisierten Modellerweiterungen enthalten.

Zusätzlich dazu beinhaltet es noch die utility Funktionen, mit denen zum Beispiel die Analysedaten gesammelt werden, Zwischenergebnisse ausgegeben werden, oder die ganze Optionsbehandlung durchgeführt werden.

3.1.2 Age

Da das Eisalter im Strömungsgesetz, das PISM standardmäßig verwendet, keinen Einfluss hat, berechnet PISM das Eisalter nur, wenn der Laufzeitparameter `-age` angegeben wurde. Wenn `-age` gesetzt ist und die Variable `age` in der Eingabedatei nicht vorhanden ist, wird das Anfangsalter von Null angenommen. Wenn die Variable `age` in der Eingabedatei vorhanden ist, der Parameter aber nicht gesetzt wurde, wird diese ignoriert. Das Alter des Eises kann in zwei Stellen im SIA-Modell verwendet werden. In PISM verwendet das Goldsby-Kohlstedt'sche Fließgesetz die Korngröße [Pa15, S. 54], die vom Alter beeinflusst wird [CCLD98; LBDP89]. Außerdem kann ein Flussfaktor an das Alter des Eises gekoppelt werden [Gre97; Pa15, S. 62].

In allen Analysen dieser Arbeit wird das Alter des Eises nicht berücksichtigt und berechnet.

3.1.3 Bed deformation

Dieses Modul beinhaltet den Code, der die Untergrundverformung modelliert. PISM implementiert dafür zwei unterschiedliche Modelle, zwischen denen gewählt werden kann. Ein punktweises Isostasie-Modell [WMH+11] sowie das Lingle-Clark-Modell [BLB07; LC85; Pa15, S. 70].

Für die Analysen dieser Arbeit wurde keines der beiden Modelle für die Untergrundverformung genutzt.

3.1.4 Stress balance

In jedem Zeitschritt eines typischen PISM-Laufs werden die Geometrie, die Temperatur und die Grundfestigkeit der Eisdecke über Spannungs-(impuls-)Gleichungen einbezogen, um die Geschwindigkeit des fließenden Eises zu bestimmen. Diese Impulsgleichungen für fließendes Eis bilden ein nicht-Newtonsches Stokes-Modell [AB09]. PISM löst jedoch nicht die dreidimensionalen Stokes-Gleichungen, stattdessen greift es zurück auf zwei verschiedene zweidimensionale Annäherungen, die sich gut für Eisschild- und Schelfeis-Systeme eignen.

SIA Die Shallow Ice Approximation (SIA) [Hut83] beschreibt das Eis durch Scherung in Ebenen parallel zum Geoid fließend, bei einer starken Verbindung der Eisbasis mit dem Grundgestein [Pa15, S. 37f]. Sie beschreiben die vertikale Scherspannung als eine lokale Funktion [Cuf10]. Die SIA-Gleichungen sind lokal in jeder Eissäule, also jedem Datenpunkt in der zweidimensionalen Modellierung, und damit unabhängig von Nachbarzellen, was vorteilhaft für die Parallelisierung ist. Sie kann für die geerdeten Teile von Eisschilden angewendet werden, also für die Bereiche, in denen das Basaleis bis zum Grundgestein gefroren ist oder nur geringfügig rutscht und die Topografie des Untergrunds relativ langsam in der Kartenebene schwankt [FFF97; HM01; Hut83; Pat03]. Diese Merkmale gelten für große Bereiche der beiden Eisschilde in Grönland und der Antarktis.

SSA Die Shallow Shelf Approximation (SSA) [WGH99] beschreibt eine membranartige Eisströmung, als ein vertikal integriertes zweidimensionales Modell, mit einer tiefengemittelten Eisgeschwindigkeit. Die SSA-Gleichungen können damit für große, schwimmende Eisschelfe, die ein kleines Verhältnis von Tiefe zu Breite haben, oder für Landeis, welches nicht fest auf dem Grundgestein sitzt, eingesetzt werden [MZ87; Mac89; Mor87; Sch06].

In PISM können diese beiden Modelle getrennt voneinander eingesetzt werden, oder als ein hybrider SSA- und SIA-Modellansatz [BKA+09; WMH+11; Pa15, S. 37f, S. 80]. Die in dieser Arbeit vorgestellten Ergebnisse beziehen sich auf Simulationen mit eben diesem hybriden Modellansatz. Dieses Modell wird dabei im Programmcode sowohl mithilfe der finiten Elemente Methode, also der finiten Differenzen Methode implementiert. Zum Stand dieser Arbeit, ist allerdings der Wechsel von der finiten Differenzen Methode hin zu der finiten Elemente Implementierung noch nicht vollständig implementiert, sodass die Analysen sich auf die Implementierung auf Basis der finiten Differenzen bezieht.

3.1.5 Subglacial hydrology

Mit diesem Modul sind zwei einfache, subglaziale Hydrologie-Modelle in PISM implementiert. Also Modelle, die beschreiben wie sich das Wasser, welches sich immer unterhalb von Eisschilden oder Gletschern befindet, verhält [Pa15, S. 67f].

3.1.6 Energy balance

PISM modelliert in Energy balance das Energieerhaltungsproblem innerhalb des Eises, der dünnen subglazialen Schicht und einer Schicht aus thermischem Grundgestein [Pa15, S. 250ff]. Für das Eis und die subglaziale Schicht verwendet es ein auf Enthalpie basierendes Schema [ABKB12], das es erlaubt, die Energie auch dann zu konservieren, wenn die Temperatur am Druckschmelzpunkt liegt.

Eis, das sich direkt am Schmelzpunkt befindet, nennt man temperiertes Eis. Ein Teil der Wärmeenergie von temperiertem Eis liegt dabei in der Wärme des flüssigen Wassers, das sich zwischen den Kristallen des temperierten Eises befindet [FW98; Fou11]. Ein weiterer Teil der Wärmeenergie des gesamten Gletschers ist in der Wärme des flüssigen Wassers, welches sich unter den Gletschern sammelt, enthalten. Das Enthalpie Schema von PISM modelliert diese Speicher thermischer Energie und erlaubt so die Modellierung von polythermalen sowie voll temperierten Gletschern [AB09;

ABKB12]. Als temperierte Gletscher oder warme Gletscher werden Gletscher bezeichnet, deren Eistemperatur sich überall mit Ausnahme der oberflächennahen Schichten, am Druckschmelzpunkt befindet. Gletscher, die sowohl Bereiche mit temperiertem als auch kälterem Eis aufweisen, werden als polythermal bezeichnet. Das Standard-Energieerhaltungsmodell (`-energy enthalpy`) kann mit `-energy cold` durch die Temperatur basierte Methode [BB09a; BBL07] ersetzt oder mit `-energy none` abgeschaltet werden.

Das thermische Grundgesteinsschichtmodell wird durch die Einstellung von `-Mbz 1` ausgeschaltet. Bei der expliziten Wahl einer Tiefe und Anzahl von Punkten, wie z. B. mit -Lbz 1000 -Mbz 21, wird dieses zugeschaltet [Pa15, S. 61f].

3.1.7 Geometry evolution

In PISM sind alle Variablen, die die Eigenschaften des Eises innerhalb einer Eisfluiddomäne, also je nach Parametern z. B. Land Eis, Gletscher oder Schelfeis, beschreiben, nicht ortsgebunden, sondern einer wechselnden Geometrie unterzogen [WMH+11; Pa15, S. 46ff] . Insbesondere bewegt sich die obere und untere Oberfläche des Eises. PISM rechnet dabei intern mit Koordinaten in orthogonalen Koordinatensystemen, wobei z in antiparalleler Richtung zur Schwerkraft steht, also mit einer Flach-Erd-Näherung. Die vertikalen Koordinaten relativ zum Geoid werden dafür durch vertikale Koordinaten relativ zur Eisbasis ersetzt. Außerdem ist es für einige Modellannahmen wichtig, dass keine Eisberge, im Speziellen keine frei schwimmenden Eismassen, die keine direkte oder indirekte Verbindung zu festem Grund haben, vorhanden sind. Bilden sich diese, werden sie aus dem Rechengebiet eliminiert. Dazu wird auch in dieser Modellkomponente das Kalben der Gletscher berücksichtigt [Pa15, S. 80].

3.1.8 Top surface boundary conditions

Mit dieser Modellkomponente ist es PISM möglich, abgetastete Oberflächenwerte, zum Beispiel aus Klimadaten, einzulesen und in den Simulationen als Randwerte zu berücksichtigen [WMH+11].

3.1.9 Sub-shelf boundary conditions

Das Modul der Sub-shelf boundary conditions ist ein Interface des `ocean`-Modells für den Eisdynamik-Code. Diese Modellkomponente dient zur Behandlung der Randwerte zwischen Eis und Meerwasser [Pa15, S. 229ff].

3.1.10 Sea level forcing

Sea level forcing ist ein Teil des `ocean`-Modells und behandelt die Kräfte, die in Bezug auf den Meeresspiegel stehen.

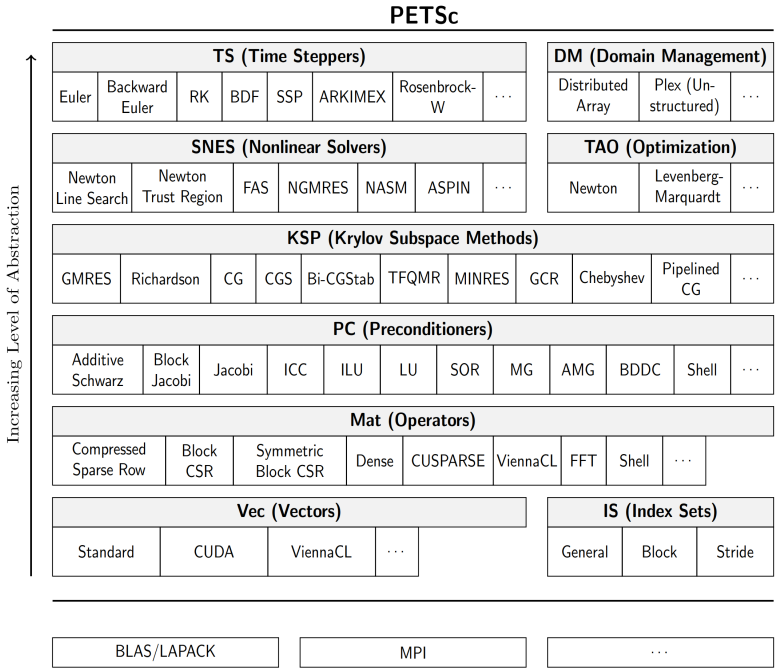


Abbildung 3.2: Übersicht der von PETSc genutzten Grundlagen, so wie darauf aufbauenden Datenstrukturen und Routinen.

Quelle: [BAA+20, S. 5]

3.2 Portable Extensible Toolkit for Scientific Computing - PETSc

PETSc ist ein Akronym für Portable, Extensible Toolkit for Scientific Computation und besteht aus Datenstrukturen und Routinen, die vom Argonne National Laboratory für das skalierbare und parallele Lösen von partiellen Differenzialgleichungen entwickelt wurden. Für die Kommunikation wird auf das MPI zurückgegriffen. Eine hybride Parallelisierung, beispielsweise mittels OpenMP[1], wird nicht unterstützt.

PETSc kann in Anwendungscodes, die in C, C++ , Fortran oder Python geschrieben sind, verwendet werden. Es bietet viele der Mechanismen, die in parallelen Anwendungscodes benötigt werden, angefangen von parallelen Datenstrukturen, über einfache parallele Matrix- und Vektor-Assembly-Routinen, bis hin zu kompletten Zeitschrittverfahren, welche die Kommunikation und Berechnung abstrahieren (siehe Abbildung 3.2).
Diese Abstraktion ermöglicht mit PETSc schnelles designen von Algorithmen und erste Skalierungsversuche beim Prototyping neuer Ideen. Andererseits wird das manuelle Optimieren durch die versteckte Kommunikation/Verteilung erschwert.

[1] https://www.openmp.org

3.3 Szenario

Für die Performanceanalyse in dieser Arbeit, beschränken sie sich auf das folgende Szenario: Simuliert wird die Antarktis in einem rechteckigen 6080 km auf 6080 km großen Gebiet. Dies entspricht dem Gebiet, ausgehend von dem geografischen Südpol (90°), bis maximal 51,8° südlicher Breite. Wie in Abbildung 3.3a (zugrundeliegende Topologie) zu sehen ist, werden dafür jedoch nur die Daten bis 60° südlicher Breite genutzt und auf ein rechteckiges Gebiet aufgefüllt. Das Ausgangsszenario geht zu Beginn von einer maximalen Eisdicke von 4580,48 m aus, wobei die Oberflächentemperatur zwischen 214,64 K und 279,35 K beträgt. Zu Beginn liegt das Grundgestein bis zu 6325,87 m unter Normalnull und bis zu 3166,09 m über Normalnull. Abbildung 3.3b visualisiert das Ausgangsszenario mit 16 km Auflösung. Dabei symbolisiert die blaue Ebene die aktuelle Meereshöhe, die durchgehende Fläche, die Oberflächenstruktur des arktischen Kontinentes und das Gitter das Eisschild, das über allem liegt.

Das Rechengebiet wird in ein gleichmäßiges reguläres Gitter unterteilt. Dabei ist die Anzahl an Gitterpunkten abhängig, von der angestrebten Auflösung des Rechengebietes. Tabelle 3.1 gibt die Anzahl der Gitterzellen in der x- und y-Achse sowie die daraus resultierende Anzahl an Gitterpunkten insgesamt an, die für eine gewünschte Auflösung berechnet werden müssen.

Ausgang für alle untersuchten Simulationen ist die Antarktis in 16 km Auflösung. Alle weiteren Auflösungen wurden mittels PISM `bootstrap` auf die jeweilige Auflösung interpoliert. `z` und `zb` wurden bei allen Messungen fix auf `z = 81` bzw. `zb = 21` gesetzt.

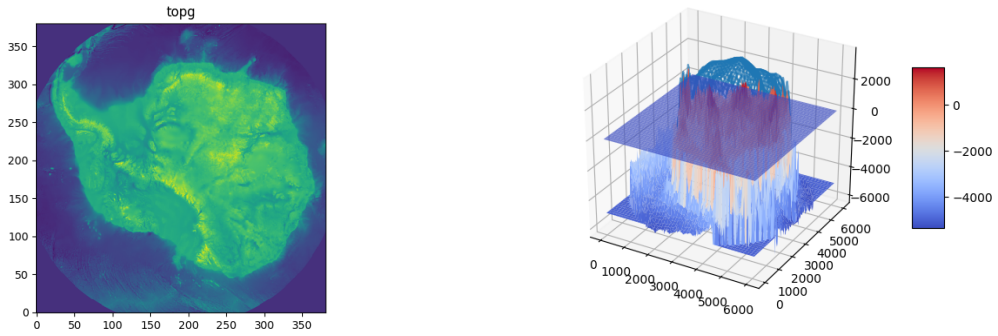
3.3.1 Bedeutung des antarktischen Eisschildes

Der antarktische Eisschild (auch antarktisches Inlandeis) ist die südliche der beiden polaren Eiskappen. Die Fläche des Eisschildes beträgt $12{,}3 \times 10^6\,\mathrm{km}^2$ (zzgl.$1{,}36 \times 10^6\,\mathrm{km}^2$ Schelfeis) und das Volumen beträgt $26{,}5 \times 10^6\,\mathrm{km}^3$ (zzgl. $0{,}4 \times 10^6\,\mathrm{km}^3$Schelfeis) [FPV+13]. Damit ist es die größte einzelne Eismasse der Erde und bedeckt den antarktischen Kontinent zu etwa 98 %. Im antarktischen Inlandeis und dem von diesem gespeisten Schelfeis sind circa 90 % des weltweiten Eises und ungefähr 70 % des Süßwassers der Erde gebunden.[Heg12]

Das Gewicht eines Kubikkilometer Eises lässt sich mit einer Gigatonne abschätzen. Das bedeutet, dass der Eisschild 26 500 000 Gigatonnen wiegt. Es wird damit gerechnet, dass ein vollständiges Abschmelzen dieses Eises ein Meeresspiegelanstieg um ungefähr 60 m bedeuten würde [FPV+13; RMS+19]. Betrug der Eisverlust zwischen 1979 und 1990 noch ungefähr $40\,\mathrm{km}^3$ pro Jahr, stieg er im Zeitraum 2009 bis 2017 auf jährlich $252\,\mathrm{km}^3$ [RMS+19; TO09]. Der weitere Verlauf dieser Entwicklung ist zukunftsweisend für das Aussehen der Erde und für seine Bewohner. Aus diesem Grund, wird aktuell mit Simulationen daran geforscht, welche Umwelteinflüsse wie mit dem Eisverlust interagieren und ab wann das Schmelzen nicht mehr zu stoppen ist.

3.3.2 Einschränkung

PISM und PETSc enthalten zusammen über 400 Kommandozeilenparameter, die unter anderem die Wahl des zugrundeliegenden Simulationsmodells sowie die dafür genutzten Datenstrukturen, Löser und Vorkonditionierer beeinflussen. Damit ist es im Umfang dieser Ausarbeitung nicht möglich, eine



(a) Topologie des Gebietes der Antarktis (> 60° südlicher Breite) aufgefüllt auf ein 6080 km auf 6080 km großes, quadratisches Gebiet.

(b) 3D Oberflächechenstruktur der Antarktis, Normalnull und das Eisschild.

Abbildung 3.3: Visualisierung des Ausgangsszenarios: Antarktis im Bereich > 60° südlicher Breite aufgefüllt auf ein 6080 km auf 6080 km großes quadratisches Gebiet.

Gittergrößen	x	y	Anzahl Gitterpunkte
1 km	6081	6081	36 978 561
2 km	3041	3041	9 247 681
4 km	1521	1521	2 313 441
8 km	761	761	579 121
16 km	381	381	145 161
32 km	191	191	36 481
64 km	96	96	9216

Tabelle 3.1: Gitter Dimensionen und Anzahl an Datenpunkten bei unterschiedlicher Auflösung.

allgemeingültige Aussage über das Performanceverhalten von PISM zu treffen, da die Anzahl an Variationen zu groß ist. Stattdessen wurde die Performanceanalyse auf ein spezifisches Szenario in verschiedenen Auflösungen beschränkt. Das Testszenario wurde von den PISM-Entwicklern für die Performance Analyse zur Verfügung gestellt und basiert auf dem Datensatz „Bedmap2: improved ice bed, surface and thickness datasets for Antarctica“ [FPV+13]. Die Einschränkung der Parameter auf ein repräsentatives Szenario aus der aktuellen Anwendung ermöglicht eine spezifischere tiefere Analyse und damit mehr Erkenntnisse zur Optimierung für die eigentliche Anwendung.

3.3.3 Startskript und Parameter

Listing 3.1 zeigt in einem Ausschnitt des Startskriptes alle gesetzten Parameter. Ausgegangen wird von einer initialen Lösung `pism_equi_input.nc`. Die Variable `$PISM_DO` enthällt den PISM Binary Aufruf zusammen mit dem jeweiligen MPI Aufrufkonstrukt (`mpirun`, `srun` etc.). `$input_dir` ist der Pfad zu dem Ausgangsszenario (`bedmap2_albmap_bheatflx_16km.nc`, `forcing_dgmt.nc`, `forcing_dsl.nc`, `forcing_dto.nc`, `pism_config_override_pism1.2.cdl`,

Gittergrößen	Simulationszeitraum	Zeitreihe
1 km	6 Monate	monthly
2 km	6 Monate	monthly
4 km	1 Jahr	monthly
8 km	10 Jahre	yearly
16 km	100 Jahre	yearly
32 km	1000 Jahre	yearly

Tabelle 3.2: Simulationszeitraum sowie `ts_times` Zeitschritt in Abhängigkeit der Auflösung.

`pism_config_override_pism1.2.nc`, `pism_equi_input.nc`, `schmidtko14_16km.nc`). Mit `$result_dir` wird der Pfad gesetzt, unter dem die Ausgabedateien erstellt werden sollen. Der Pfad muss existieren. `$log` und `$NN` haben keinen Einfluss. Sie wurden gesetzt, um verschiedene Läufe und Ausgaben voneinander zu unterscheiden. Im Skript enthält `$log` Informationen zu dem Szenario und `$NN` die Gitterauflösung.

In Zeile 1 wird das PISM Binary zusammen mit dem jeweiligen MPI Konstrukt (`mpirun`, `srun` etc.) aufgerufen. Zeile 2 lädt das Ausgangsszenario mit 16 km Auflösung. Zeile 3 und 4 passen die Gitterauflösung entsprechend der gewünschten Auflösung an. Tabelle 3.1 zeigt die jeweilige Gitterauflösung, für eine gewünschte Auflösung. In den Zeilen 5 bis 18 werden weitere Modellparameter gesetzt, die bei allen Messungen gleich geblieben sind. Mit Zeile 19 wird der Simulationszeitraum gesetzt. Da PISM die Zeitschrittweite dynamisch anpasst und bei feinerer Auflösung die Zeitschritte kleiner werden, wurde für feinere Auflösungen der Simulationszeitraum, wie in Tabelle 3.2 dargestellt, angepasst. In den Zeilen 20 bis 25 werden die Ausgabepfade gesetzt. Hier wurde die Abspeicherung der Zeitreihe abhängig von der Auflösung gewählt (siehe Tabelle 3.2), sowie auf den Simulationszeitraum angepasst. Ansonsten ist die Ausgabe unabhängig von der Auflösung. Zeile 26 ist ein PETSc Parameter, der eine Ausgabe mit Performancewerten im Pythonformat zur weiteren Analyse erstellt.

3.4 Hardware Übersicht

Die Performanceanalyse wurde auf vier verschiedenen Plattformen durchgeführt. Zusätzlich zu den drei in Tabelle 3.3 vorgestellten Clustern, wurde eine einzelne Linux Maschine mit einer vier-Kern Intel i5-4670k CPU und 16 GB DDR3 Hauptspeicher mit 1333 MHz für die auf Vtune und Amplex basierenden Analysen genutzt. Auf dieser Maschine wurde die Frequenz fix auf 4 GHz gesetzt. Die 4 GHz wurden auch bei voller AVX Auslastung gehalten. Die Charakteristiken der genutzten Prozessoren wird in Tabelle 3.4 zusammengefasst. Sowohl der Intel Xeon Gold 6230 als auch der Intel Xeon E5-2667 v3 wurden bei ihrer Markteinführung als Hochleistungshardware betrachtet. Der Intel Xeon E3–1585 v5 und der Intel i5–4670k hingegen stammen eher aus dem gehobenen Verbraucherbereich. Aus diesem Grund, und da es eine Auswahl unterschiedlicher Generationen ist, welche auf unterschiedlichen Mikroarchitekturen basieren, unterscheiden sich sowohl die Gesamtleistung, als auch einzelne Performance Metriken. Diese Diversifikation soll dazu dienen, den Einfluss unterschiedlicher Hardware zu untersuchen. Außerdem verfügen die

Listing 3.1 PISM Aufruf und alle gesetzten Parameter

```
"$PISM_DO \  @\label{myline}@
-i ${input_dir}/pism_equi_input.nc  \
-bootstrap -Mx $Mx -My $My -Lz 6000 -Lbz 2000 -Mz 81 -Mbz 21 -grid.registration corner \
-regrid_file ${input_dir}/pism_equi_input.nc -regrid_vars tillwat,enthalpy,litho_temp,temp,tillphi \
-pik -sia_e 2.0 -sia_flow_law gpbld -limit_sia_diffusivity \
-stress_balance ssa+sia -ssa_method fd -ssa_flow_law gpbld -ssa_e 0.6 \
-yield_stress mohr_coulomb -pseudo_plastic -pseudo_plastic_q 0.75 -pseudo_plastic_uthreshold 100.0 \
-calving eigen_calving,thickness_calving \
-front_retreat_file ${input_dir}/bedmap2_albmap_bheatflx_16km.nc \
-eigen_calving_K 1.0e17 -thickness_calving_threshold 75.0 \
-bed_def none -hydrology null -tauc_slippery_grounding_lines \
-topg_to_phi 1,45,-900,1500 \
-ocean pico -ocean_pico_file ${input_dir}/forcing_dto.nc \
-sea_level constant,delta_sl -ocean_delta_SL_file ${input_dir}/forcing_dsl.nc \
-gamma_T 1.0e-5 -overturning_coeff 0.8e6 -exclude_icerises -continental_shelf_depth -2500 \
-atmosphere given,delta_T -atmosphere_given_file ${input_dir}/bedmap2_albmap_bheatflx_16km.nc \
-atmosphere_delta_T_file ${input_dir}/forcing_dgmt.nc -surface simple \
-options_left -verbose 2 -o_size none -backup_interval 3.0 \
-config_override ${input_dir}/pism_config_override_pism1.2.nc \
-ys $start -ye $endyr \
-save_file ${result_dir}/snap_test_${log} -save_times $start:1:$endyr -save_split -save_size none \
-ts_file ${result_dir}/ts_test_${log}.nc -ts_times $start:$save:$endyr \
-extra_file ${result_dir}/extra_test_${log}.nc -extra_times $start:monthly:$endyr \
-extra_vars thk,topg,usurf,mask,velsurf_mag,dHdt,tillwat,ice_surface_temp,\
climatic_mass_balance,shelfbmassflux,shelfbtemp,rank \
-o ${result_dir}/equi_test_${log}.nc -o_size none \
-profile ${result_dir}/profile_test_${log}_${NN}.py -log_view"
```

Name	sgscl1	bwUniCluster HTC+HPC	PIK
CPU	Intel Xeon E3-1585 v5	Intel Xeon Gold 6230	Intel Xeon E5-2667 v3
Knoten	16	460	312
Kerne pro Knoten	4	2×20	2×8
Hauptspeicher pro Knoten	32 GB DDR4	96 GB DDR4	64 GB DDR4
Hauptspeichertackt	2133 MHz		2133 MHz
Netzwerk	10 Gbit Ethernet	IB HDR100	Mellanox Connect-IB FDR
Netzwerk-Bandbreite	10 Gbit/s	100 Gbit/s	56 Gbit/s

Tabelle 3.3: Eine Zusammenfassung der Hardware aller in dieser Arbeit verwendeten Computer-Cluster.

Computer-Cluster entweder über 10 Gbit Ethernet, IB HDR100 oder Mellanox Connect-IB FDR. Dies ermöglicht den Einfluss unterschiedlicher Latenzen und Bandbreiten in der Kommunikation zu untersuchen.

Die Ausgabedateien von PISM wurden bei allen Läufen auf den Computer-Clustern auf ein geteiltes Filesystem über das Netzwerk geschrieben. Auf der Linux Maschine auf eine NVME SSD.

CPU Mikroarchitektur	Intel Xeon E3-1585 v5 Skylake	Intel Xeon Gold 6230 Cascadelake	Intel Xeon E5-2667 v3 Haswell	Intel i5-4670k Haswell
Kerne	4	20	8	4
Frequenz (Boost) AVX	3,5 GHz(3,9 GHz)	2,1 GHz(3,9 GHz)	3,2 GHz(3,6 GHz)	4 GHz(4 GHz)
L1 Bandbreite	4(32 KiB + 32 KiB) = 256 KiB 4 · 3,9 GHz · 96 B ≈ 1,3 TB/s	20(32 KiB + 32 KiB) = 1,25 MiB 20 · 3,9 GHz · 192 B ≈ 15 TB/s	8(32 KiB + 32 KiB = 512 KiB 8 · 3,6 GHz · 96 B ≈ 2,8 TB/s	4(32 KiB + 32 KiB) = 256 KiB 4 · 4 GHz · 96 B ≈ 1,5 TB/s
L2 Bandbreite	4 · 256 KiB = 1 MiB 4 · 3,9 GHz · 64 B ≈ 452 GB/s	20 · 1 MiB = 20 MiB 20 · 3,9 GHz · 64 B ≈ 5 TB/s	8 · 256 KiB 8 · 3,6 GHz · 64 B ≈ 1,8 TB/s	4 · 256 KiB = 1 MiB 4 · 4 GHz · 64 B ≈ 1 TB/s
L3 Bandbreite	8 MiB 3,9 GHz · 32 B ≈ 125 GB/s	27,5 MiB 3,9 GHz · 32 B ≈ 125 GB/s	20 MiB 3,9 GHz · 32 B ≈ 115 GB/s	6 MiB 4 GHz · 32 B ≈ 128 GB/s
L4	128 MiB	-	-	-
FMA	FMA3	FMA3	FMA3	FMA3
AVX	2· AVX2(32 B)	2· AVX-512(64 B)	2· AVX2(32 B)	2· AVX2(32 B)
Peak FLOPs SP/DP	0,5 TF / 0,25 TF	5 TF / 2,5 TF	0,9 TF / 0,46 TF	0,5 TF / 0,25 TF
Speicherbandbreite	34 GiB/s	131,13 GiB/s	68 GiB/s	25,6 GiB/s
Maschinengleichgewicht turbo	12 F/B / 6 F/B 14 F/B / 7 F/B	19 F/B / 10 F/B 35 F/B / 18 F/B	11 F/B / 6 F/B 13 F/B / 6 F/B	19 F/B / 9 F/B 19 F/B / 9 F/B

Tabelle 3.4: Eine Zusammenfassung der Leistungsmerkmale aller in dieser Arbeit verwendeten Hardware-Plattformen auf Knotenebene [201a; 201b; 201c; 201d; 202; Fog20]. Zu beachten ist, dass alle Plattformen ihre Frequenz je nach Leistungsaufnahme, Temperatur und Arbeitslast variieren. Das Maschinengleichgewicht wurde mangels genaueren Frequenztabellen einmal mit dem Turbo- und dem Basistakt zur genaueren Einschätzung angegeben. Die Bandbreitenberechnung basiert auf den theoretisch möglichen maximalen Bandbreiten der zugrundeliegenden Mikroarchitekturen und kann abweichen.

3.5 Versionen

Die Performance Analyse basiert auf unterschiedlichen Versionen des Master Zweiges von PISM. Alle Messungen auf der Linux Maschine, so wie auf sgscl1 wurden mit der PISM Version `stable v1.2.1`[2] durchgeführt. Die Messungen auf dem PIK und dem bwUniCluster Computer-Cluster wurden mit der Version `stable v1.2.2` [3] durchgeführt. Die Versionen sind im eigentlichen Programmcode bis auf kleine Fehlerbehebungen identisch, sodass auch die einzelnen Ergebnisse untereinander vergleichbar sind. Das Änderungsprotokoll von Version 1.2.1 zu 1.2.2 ist in dem PISM GitHub Repository unter `CHANGES.rst`[4] aufgeführt. Die Änderungen im Detail, lassen sich über ein GitHub Vergleich[5] betrachten. Auf jeder Hardware wurde gegen PETSc in Version 3.13 gelinkt. Außer auf dem PIK Computer-Cluster wurde immer gegen OpenMPI in Version 3.1.6 gelinkt. Auf dem PIK Computer-Cluster diente die Intel MPI Bibliothek in Version 2018 Update 1 Build 20171011 als Parallelisierungsgrundlage. Als Cluster-Management- und Job-Scheduling-System läuft auf den Clustern Slurm (pik:17.11.7, sgscl1: slurm-wlm 19.05.5, bwUniCluster: 20.02.3).

Alle Untersuchungen, bei denen der Programmcode verändert wurde, wurden auf jeder Hardware auf der Basis von PISM in Version 1.2.1 getätigt. Die jeweiligen Codeänderungen sind über die jeweiligen Zweige unter `https://github.com/Loewe2/pism` dokumentiert.

3.6 Performanceanalyseprogramme

Alle Analysen in dieser Arbeit, die sich auf die Laufzeit, die Anzahl der Aufrufe von Programmabschnitten, MPI Nachrichten innerhalb von Programmabschnitten, so wie die Anzahl an Fließkommaberechnungen in einem Abschnitt beziehen, basieren direkt auf Daten die PETSc als Laufzeit-Daten bereitstellt. Zur Einschätzung der Korrektheit der Anzahl an Fließkommaoperationen wurden einzelne Analysen mit Perf in Version 5.4.65 validiert. Für weitere Knotenanalysen wurde auf den Intel VTune Profiler (build 610396) sowie den Intel Advisor (October 2018) zurückgegriffen. Die Speicheranalyse wurde mittels Slurm, der Werkzeugsammlung Valgrind (3.15.0) sowie Heaptrack (1.1.80) realisiert.

PETSc PETSc bietet direkt Methoden zum Profilen von Anwendungen, die die PETSc-Bibliotheken nutzen. Dafür protokolliert PETSc automatisch die Objekterstellung, Zeiten und Fließkommazahlen für die Bibliotheksroutinen. Durch Funktionen kann dies auch auf weitere Programmfragmente außerhalb von PETSc angewandt werden. Mit der Hilfsklasse Profiling (implementiert in `src/util/Profiling.cc`) stellt PISM eine darauf basierende Hilfsklasse für das Profilen zur Verfügung.

[2] `https://github.com/pism/pism/tree/7d46367`
[3] `https://github.com/pism/pism/tree/5e1debde`
[4] `https://github.com/pism/pism/blob/v1.2.2/CHANGES.rst`
[5] `https://github.com/pism/pism/compare/5e1debde...7d46367`

Perf Perf ist ein Profiling-Tool, welches im Linux Kernel unter tools/perf[6] enthalten ist. Perf basiert auf der perf_events-Schnittstelle, eine Zusammenfassung der Hardware- (z. B. Taktzyklen, Instruktionen, Cache-Referenzen und Cache-Misses) und Software-Performance-Counter (z.B: task-clock-msecs, kontext-wechsel, CPU-migrations und page-faults) sowie weitere Events der Prozessor spezifischen Performance Monitoring Unit (PMU).

Intel VTune Profiler Der Intel VTune Profiler[7] ist eine kommerzielle Anwendung von Intel zur softwareseitigen Performanceanalyse von x86 Anwendungen. Der VTune Profiler unterstützt verschiedene Arten von Code-Profiling, einschließlich Stack-Sampling, Thread-Profiling und Hardware-Event-Sampling. Die Analyse besteht, unter anderem, aus der in jeder Unterroutine/ Instruktion verbrachten Zeit. Das Tool kann auch zur Analyse der Thread- und Speicherleistung verwendet werden.

Intel Advisor Intel Advisor[8] (auch bekannt als Advisor XE) ist ein Tool zur Optimierung der SIMD-Vektorisierung und zur Unterstützung des Threading im gemeinsamen Speicher für C, C++ , C# und Fortran. Dabei misst es die genaue Anzahl an Anruf- und Iterationszähler für alle Schleifen, analysiert Speicherzugriffsmuster und erstellt Software gestützte Roofline Modelle.

Valgrind Valgrind[9] ist Open Source und ist frei verfügbar unter der GNU General Public License Version 2. Valgrind ist ein Instrumentierungs-Framework für den Aufbau dynamischer Analysewerkzeuge und darauf basierende Analysewerkzeuge [NS03]. Das zu debuggende oder zu analysierende Programm läuft in Valgrind nicht direkt auf der Host-CPU [CLCV08; NS07] . Stattdessen übersetzt Valgrind das Programm in einen temporären Bytecode (Vex IR). Dieser wird von den verschiedenen Valgrind-Tools transformiert und annotiert, bevor Valgrind den neuen Code zurück in Maschinencode übersetzt und ausführt. Dieses Verfahren hat allerdings den Nachteil, dass es die Laufzeit meist deutlich negativ beeinflusst.

Heaptrack Heaptrack[10] ist ein Memory Profiler. Ähnlich wie das verbreiterte Massif Tool aus der Valgrind Bibliothek analysiert Heaptrack Programme, indem es `malloc` und `free` trackt, ist dabei aber leichtgewichtiger und hat weniger Overhead [Wol14].

3.7 Skalierung

In dieser Arbeit werden verschiedene Skalierungsbegriffe verwendet. Als Grundlage, um über die Skalierung zu diskutieren, dient der Begriff des Skalierungsfaktors. Mit diesem wird das starke sowie das schwache Skalierungsverhalten im Folgenden definiert. Außerdem werden im weiteren Verlauf des Kapitels die Begriffe des geographischen sowie zeitlichen Skalierungsverhaltens definiert.

[6] https://github.com/torvalds/linux/tree/master/tools/perf
[7] https://software.intel.com/content/www/us/en/develop/tools/vtune-profiler.html
[8] https://software.intel.com/content/www/us/en/develop/tools/advisor.html
[9] https://www.valgrind.org/
[10] https://github.com/KDE/heaptrack

Skalierungsfaktor Den Faktor, um den die Leistung zunimmt bei genau einer zusätzlichen Ressourcen-Einheit, nennt man SpeedUp oder Skalierungsfaktor. Der Skalierungsfaktor ist allerdings im Allgemeinen nicht unabhängig von der Anzahl der schon eingesetzten Hardware. Wenn der Skalierungsfaktor beim Hinzufügen von Ressourcen größer wird, nennt man dies eine super-lineare Skalierung. Lineare Skalierbarkeit bedeutet, dass der Skalierungsfaktor unabhängig von der Anzahl an Ressourceneinheiten ist. Nimmt der Skalierungsfaktor beim Hinzufügen von weiteren Ressourcen hingegen ab, ist dies eine sub-lineare Skalierung. Von negativer Skalierbarkeit spricht man, wenn der Skalierungsfaktor negativ ist, also wenn sich die Leistung durch das Hinzufügen von weiteren Ressourcen verschlechtert.

Starke Skalierung Unter dem Begriff der „starken Skalierung“ versteht man das Verhalten der Laufzeit bei gleichbleibender Gesamtproblemgröße und variierender Anzahl der Prozessoren, also die Problemgröße auf den Prozessoren variiert. Das Amdahlsche Gesetz [Amd67]:

$$S_p = \frac{1}{f + \frac{1-f}{p}} \tag{3.1}$$

kann dabei als Grundlage dienen, den jeweiligen Skalierungsfaktor abzuschätzen. S_p steht dabei für den Speedup wenn p Ressourcen-Einheit eingesetzt werden. Zur Berechnung wird außerdem der serielle Codeanteil $f \in [0{,}0, 1{,}0]$ benötigt.

An dem Gesetz ist abzulesen, dass der Skalierungsfaktor bei steigender Anzahl an Ressourcen gegen $S_p = \lim_{p \to \inf} \frac{1}{f}$ läuft. Bedingt dadurch, dass Programme im Allgemeinen immer einen sequentiellen Anteil haben, kann also ein Problem nicht beliebig schnell berechnet werden, indem immer mehr Hardware darauf angesetzt wird.

Schwache Skalierung Unter dem Begriff der „schwachen Skalierung“ versteht man das Verhalten der Laufzeit bei einer konstanten Problemgröße für jeden Prozessor und variierender Anzahl der Prozessoren. Gustafson hat im Jahr 1988 auf der Basis von Amdahls Gesetz ein Gesetz veröffentlicht [Gus88], welches wie in Amdahls Gesetz den Speedup S_p in Abhängigkeit von dem sequentiellen Programmanteil f sowie der Anzahl an Ressourcen-Einheiten p beschreibt:

$$S_p = f + p(1 - f) \tag{3.2}$$

Das bedeutet, dass im Allgemeinen, um größere Probleme zu lösen, mit Sicherheit mehr Hardware eingesetzt werden kann.

Die Gesetze von Amdahl und Gustafson sind stark vereinfacht, dienen aber als Grundlage für weitere Gesetze, die zum Beispiel Speicherlimitierungen miteinbeziehen [FGEXDS20].

Geographische Skalierung Die grafische Skalierung beschreibt, wie sich die Laufzeit verhält, wenn die geografische Problemgröße verändert wird, also wie sich diese einerseits verhält bei Veränderung der Gebietsgröße, das heißt wenn zum Beispiel von Grönland auf die Antarktis gewechselt wird, als auch wenn die Diskretisierung verändert wird. Für die Analysen in dieser Arbeit wurde die Gebietsgröße konstant gehalten und das Skalierungsverhalten bei unterschiedlicher Gebietsauflösung untersucht.

Zeitliche Skalierung Die zeitliche Skalierung beschreibt, wie sich die Laufzeit unter verschiedenen Simulationszeiträumen verhält. Auch mit der zeitlichen Skalierung kann die Problemgröße variiert werden. PISM parallelisiert nur räumlich und nicht in der Zeit. Da nach dem Spin-Up Lauf und der Initialisierungsphase die einzelnen Zeitschritte sich sehr ähnlich verhalten, wird in dieser Arbeit die zeitliche Skalierung dazu genutzt, der Laufzeitentwicklung der geographischen Skalierung entgegen zu wirken. Läufe auf einer groben Gebietsunterteilung, werden über längere Zeiträume simuliert, um den Anteil der Initialisierungsphase gering zu halten. Läufe auf einer feinen Gebietsunterteilung, werden über kürzere Zeiträume simuliert, um die benötigte Rechenzeit für die Analysen zu begrenzen. Durch das adaptive Zeitschrittverfahren von PISM, werden bei feinerer Auflösung für den gleichen zeitlichen Simulationszeitraum mehr Zeitschritte benötigt. Außerdem fällt die Initialisierungsphase deutlich weniger ins Gewicht, weil die eigentlichen Zeitschritte im Verhältnis deutlich mehr Rechenzeit benötigen.

4 Performanceübersicht

Dieses Kapitel beschäftigt sich damit, eine allgemeine Performanceübersicht über PISM zu geben. In Abschnitt 4.1 werden Performanceeigenschaften von PISM im Ganzen anhand einer Intel Advisor Analyse betrachtet. Für weitere Analysen wird der Programmcode in einzelne Abschnitte unterteilt. In Abschnitt 4.2 werden die Bezeichnungen dieser Abschnitte erläutert sowie auf Details in der folgenden Analyse eingegangen. Abschnitt 4.3 analysiert das Skalierungsverhalten der einzelner großer Abschnitte und arbeitet die Abschnitte heraus, die in dem analysierten Szenario die Hotspots darstellen. Abschnitt 4.4 geht auf den Hauptspeicherverbrauch von PISM ein und Abschnitt 4.5 behandelt die Belastung durch die Menge an Daten, die auf sekundären Speicher geschrieben wird.

4.1 Erster Einblick

Eine erste Performanceeinschätzung lässt sich mit dem Intel Advisor auf einem Central Processing Unit (engl. Prozessor) (CPU) Kern vornehmen. Wie in Tabelle 4.1 auf den ersten Blick zu sehen ist, gibt es bei der Performance Verbesserungspotential. Im Vergleich zu der sehr niedrigen Auslastung der restlichen CPU war die Hauptspeicherauslastung vergleichsweise hoch. Zudem lag der ungefähre Anteil, der in vektorisierten Schleifen verbrachten Zeit, bei 3,2 % von der insgesamt in Schleifen verbrachten Zeit. Eine mögliche Erklärung für diese beiden Ergebnisse wäre, dass aufgrund der fehlenden Vektorisierung die Cache-Blöcke (Cache-Lines) nicht ausgenutzt werden und so die Hauptspeicherbandbreite selbst auf einem Kern der limitierende Faktor ist.

Ein weiterer Faktor, der nicht vernachlässigt werden sollte, ist außerdem, dass bei einer derart kurzen Simulationszeit eventuell das Lesen des Szenarios und das Schreiben der Ergebnisse mehr ins Gewicht fallen.

		Auslastung	Hardware Peak
GFLOPS	1,237	2,02 %	63,763 (DP) FLOPS
GINTOPS	1,860	5,83 %	31,913 (Int64) INTOPS
		2,92 %	63,791 (Int32) INTOPS
CPU <-> Memory / L1	29,909 GB/s	7,32 %	381,431 GB/s
L2 Bandbreite	4,573 GB/s	4,24 %	107,883 GB/s
L3 Bandbreite	4,097 GB/s	8,04 %	50,982 GB/s
DRAM Bandbreite	3,610 GB/s	29,43 %	12,264 GB/s

Tabelle 4.1: Intel Advisor Auslastungsanalyse auf einem Kern der Linux Maschine mit 10 Jahre Simulationszeit und 16 km Auflösung.

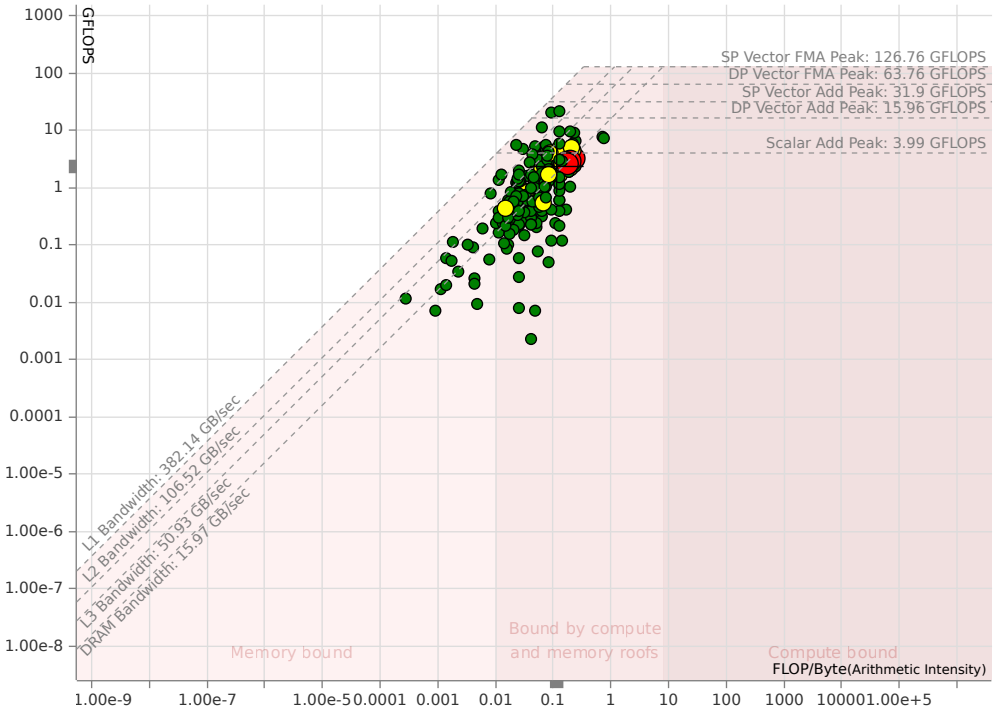


Abbildung 4.1: Intel Advisor generiertes Roofline Modell für einen Lauf auf einem Kern der Linux Maschine mit 10 Jahre Simulationszeit und 16 km Auflösung.

Die Hardware Peakperformance wurde bei dieser Analyse nicht (wie in Tabelle 3.4) berechnet, sondern von dem Intel Advisor automatisch über die Laufzeit optimaler Codes bestimmt. Dies bedeutet, dass die CPU tatsächlich, unter optimalem Code, sehr nahe an die theoretische Peakperformance für einen einzelnen Kern kommt. Die theoretische Peakperformance lässt sich berechnen, indem die Taktfrequenz mit den pro Takt möglichen Fließkommaoperationen multipliziert wird. Der Intel i5 4670k kann pro Takt jeweils vier Fused Multiply Add (FMA) Operationen auf zwei Vektoreinheiten durchführen:

$$4{,}0 \times 10^9 \cdot 4 \cdot 2 \cdot 2 = 64 \times 10^9 \tag{4.1}$$

Auch die L1 Bandbreite stimmt mit der theoretischen Berechnung überein. Allerdings sind die restlichen, automatisch ermittelten Werte teilweise weit entfernt von den theoretisch möglichen. Bei mehreren Ausführungen der Analyse hat sich gezeigt, dass diese Werte stark schwanken und teilweise einzeln nahe an die theoretische Peakperformace reichen. Es scheint, als ob die kurzen Intel Advisor Benchmarks die Taktrate der CPU nicht schnell genug anheben können. Bei anderen Hardware Benchmarking Tools wurde über längere Zeit eine konstante Performance nahe der Peakperformance gemessen. Damit sind die niedrigen, prozentualen Auslastungen in der Realität noch etwas geringer.

Abbildung 4.1 zeigt ein mit dem Intel Advisor erstelltes erweitertes Roofline Modell. Der Intel Advisor bezieht dabei seine Daten aus einem analysierten Programmdurchlauf, und dem, mit debugge Symbolen gebauten, Programmcode. Analysiert werden dabei große Schleifen sowie Funktionen.

Auf der X-Achse wird bei einem Roofline Modell die arithmetische Intensität, also die durchschnittliche Anzahl an Operationen, die pro benötigten Datum ausgeführt werden können, aufgetragen. Auf der Y-Achse werden die bei der Ausführung gemessenen Anzahlen an Fließkommaoperationen aufgetragen. Des Weiteren werden die Hardwareeigenschaften, im Speziellen die Speicherbandbreite und Peakperformance, in den Graphen eingezeichnet.

Werden jetzt einzelne Schleifen, Abschnitte, Funktionen oder das ganze Programm in einen solchen Graphen eingetragen, lassen sich verschiedene Eigenschaften direkt ablesen. Es kann direkt abgelesen werden, ob der analysierte Abschnitt durch die Speicherbandbreite limitiert ist oder, ob die Rechenleistung der zugrundeliegenden Hardware der limitierende Faktor ist. Von einer Bandbreitenlimitierung spricht man in dem Fall, wenn der eingetragene Punkt auf dem X-Achsenabschnitt liegt, auf dem die unter dieser arithmetischen Intensität maximal erreichbaren Bandbreite kleiner ist, als die Peakperformance. Liegt der Punkt außerhalb dieses Bereiches, spricht man von einer Berechnungslimitierung. Im Idealfall liegen die Performancemessungen sehr nahe an den Limitierungen. Ist dies der Fall, ist der Abschnitt effizient implementiert und mehr Performance kann nur durch eine algorithmische Änderung erreicht werden. Liegen die Messungen allerdings nicht an einer Limitierung, wurde entweder die arithmetische Intensität falsch bestimmt, eine Limitierung (wie zum Beispiel Kommunikationsbarrieren, Festplattenbandbreite, etc.) nicht beachtet, oder der Programmcode ist ineffizient. Ist die arithmetische Intensität korrekt, und der größte Limitierungsfaktor bestimmt, beschreibt der vertikale Abstand zur nächsten/größten Limitierung die mögliche Performancesteigerung, die alleine durch eine effizientere Implementierung erreicht werden kann. Automatisch generierte Roofline Modelle haben dabei gegenüber den theoretisch generierten einige Limitierungen. Die arithmetische Intensität wird gemessen, oder aus dem Programmcode extrahiert. Damit ist sie implementierungsabhängig und spiegelt nicht zwingend den zugrundeliegenden Algorithmus wieder.

In Abbildung 4.1 ist ein erweitertes Roofline Modell dargestellt. Hier wird nicht nur die maximale Peakperformance und die minimale Bandbreitenlimitierung betrachtet, sondern es werden mit den unterschiedlichen Cache Bandbreiten, sowie der Peakperformance ohne Vektorisierung, reine Vektorisierung und der Peakperformance von Vektorisierung inklusive der Ausnutzung von FMA, spezifizierter Limitierungsfaktoren betrachtet.

In den meisten Fällen, kann ein Teil der Cache-Architektur genutzt werden, sodass speicherbandbreitenlimitierte Codes zwischen der L1 Bandbreite und der Hauptspeicherbandbreite gemessen werden sollten. Sind Codes berechnungslimitiert, kann evaluiert werden, wie viel Performance ohne Vektorisierung sowie mit Vektorisierung, maximal möglich ist.

Abbildung 4.1 zeigt, dass ein großer Teil von PISM speicherbandbreitenlimitiert ist und Teile der analysierten Funktionen ein deutliches Verbesserungspotenzial bieten. Der Intel Advisor kodiert über die Größe und Farbe der Datenpunkte, den prozentualen Anteil der Gesamtlaufzeit der entsprechenden Funktionen bzw. Schleifen. Da diese Abschnitte über der Linie liegen, die die Hauptspeicherbandbreite auf der zugrundeliegenden Hardware darstellt, sind sie vermutlich durch die Hauptspeicherbandbreite limitiert und bieten nicht mehr viel Verbesserungspotenzial. Da der Simulations-Code PISM sehr umfangreich ist, kann im Nachfolgenden allerdings PISM nicht im

Kompletten analysiert werden. Stattdessen muss die Analyse auf Ausschnitte von PISM begrenzt werden. Da laufzeitintensivsten Codeabschnitte, die sogenannten Hotspots, den größten Anteil an der Laufzeit, und damit das größte potenzielle Verbesserungspotential, haben, wird sich im weiteren auf diese fokussiert. Allerdings ist die Speicherbandbreite in Tabelle 4.1 insgesamt nur zu 29,43 % ausgelastet ist. Das bedeutet in diesem Fall, dass nicht nur der laufzeitintensive Code performancekritisch ist, sondern dass sich viele kurze, nicht optimierte Abschnitte hier aufaddieren und so PISM im Gesamten wesentlich verlangsamen. Auch hat der Intel Advisor in Abbildung 4.1 die Speicherbandbreiten und CPU-Performance teilweise erneut zu niedrig ermittelt, sodass dieser Effekt noch stärker ist, als es in Abbildung 4.1 scheint.

4.2 Analyse Details

In diesem Abschnitt geht es darum, die Rahmenbedingungen für die Analysen aufzuzeigen. Es geht um die Bedeutung der Namen, wie die Ergebnisse aufbereitet wurden und wie verlässlich die mit PETSc ermittelten Daten sind.

4.2.1 Codeabschnitt Bezeichung

PISM ist mit PETSc-Performance-Profiling-Annotationen versehen. Sofern nicht genauer deklariert, entsprechen die Namen in der Analyse (wie zum Beispiel `stress_balance`, `surface`, `ocean`), den entsprechenden Codefragmenten in PISM, die sich innerhalb der gleich bezeichneten Umgebung im Programmcode befinden. Zur genaueren Analyse werden im Verlauf der Arbeit weitere Ausschnitte eingeführt, die die bestehenden Abschnitte weiter unterteilen und so eine spezifischere Analyse ermöglichen, die beispielsweise ermöglichen, zu analysieren in welchem Unterabschnitt der Hotspot genau liegt. Der hierzu abgewandelte Programmcode ist in einem PISM Fork[1] zu dieser Arbeit enthalten.

In den geschichteten Laufzeitgraphen (Graphen wie z. B. Abbildung 4.2), beschreibt die farbige Fläche die Summe der Laufzeiten bzw. die normierte Laufzeit die jeder Thread in einem bestimmten Programmabschnitt verbracht hat. Das bedeutet die Höhe, auf der beispielsweise in Abbildung 4.2 der Abschnitt `stress_balance` aufhört, beschreibt nicht die Laufzeit von `stress_balance` an diesem Punkt, sondern die Summe alle Laufzeiten von `stress_balance` und den darunter abgebildeten Graphen.

Wie zu sehen ist, decken diese selektierten, nicht überlappenden Codeausschnitte nahezu die gesamte Laufzeit ab, die als Summe aller Laufzeiten mit einer schwarzen Linie in der Grafik eingezeichnet ist. Dies bedeutet, dass die Summe der Analysen dieser Abschnitte repräsentativ für ganz PISM ist.

[1] https://github.com/Loewe2/pism

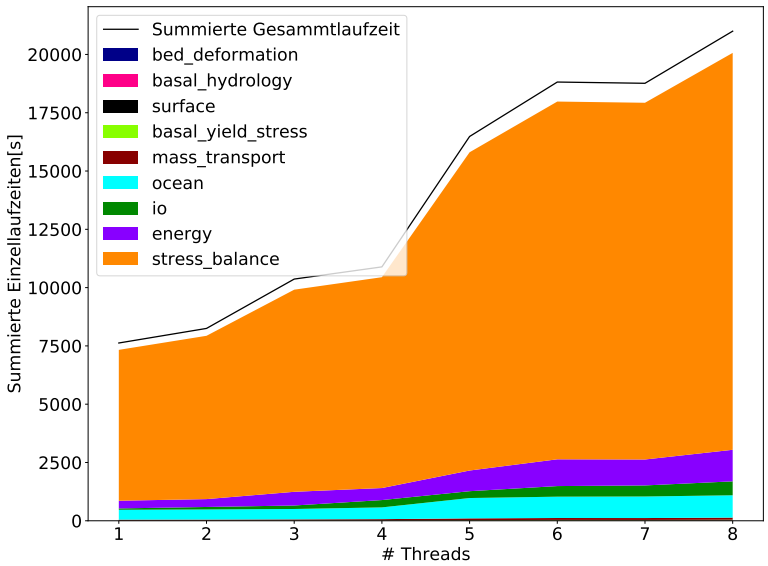


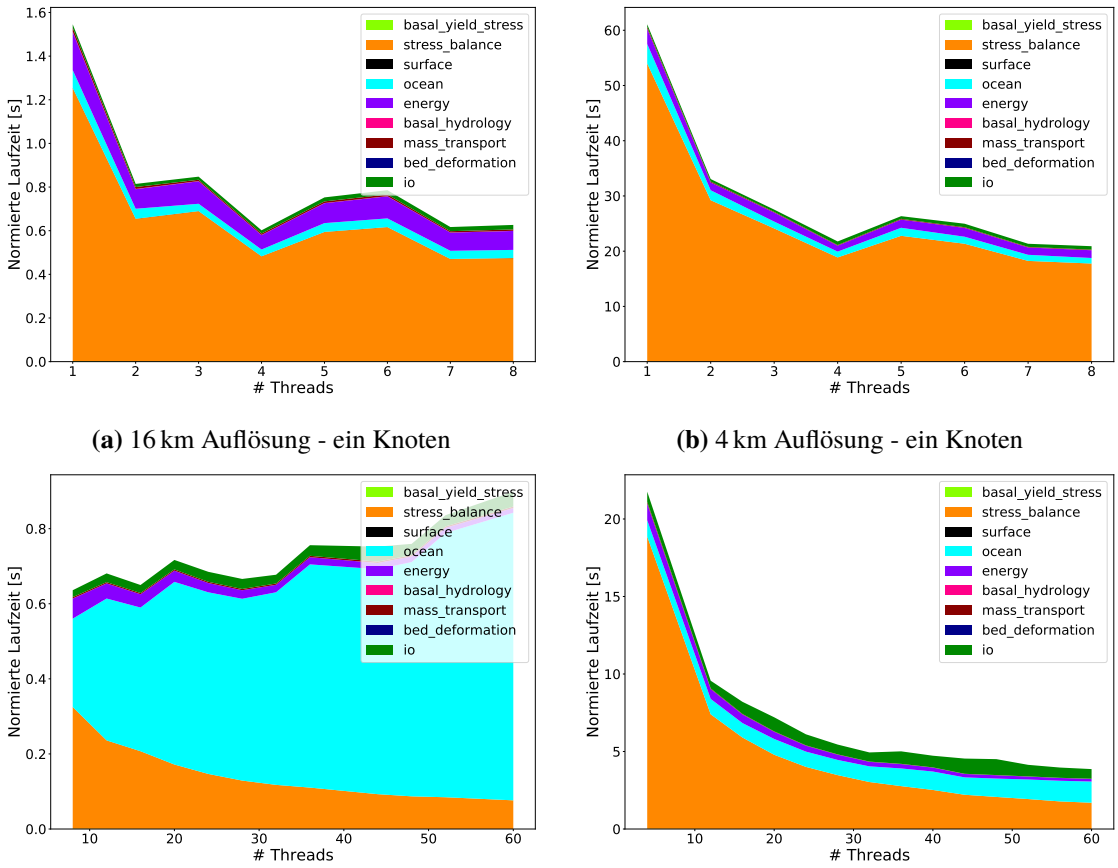
Abbildung 4.2: PISM Gesamtlaufzeit, so wie die Laufzeiten einzelner Codeabschnitte aufsummiert über alle Threads. Gemessen bei 4 km Auflösung auf sgscl1.

4.2.2 Normierung

PISM nutzt ein adaptives Zeitschrittverfahren. Eine feinere Auflösung führt also dazu, dass die Zeitschrittweite verkleinert wird. Eine kleinere Zeitschrittweite hat allerdings zur Folge, dass für den gleichen Simulationszeitraum unterschiedlich viele Zeitschritte berechnet werden müssen. Damit bestimmte Vergleiche, wie z. B. die Analyse der starken Skalierung, möglich sind, müssen die Messungen deshalb normiert werden. Indem die gesamte Laufzeit der zu evaluierenden Funktion durch die Anzahl der Aufrufe der jeweiligen Funktion geteilt wird, ist es möglich jeweils einen durchschnittlichen Aufruf miteinander zu vergleichen. Dies ist dabei nicht mit der Zeit pro Zeitschritt zu verwechseln, da teilweise Funktionen mehrfach pro Zeitschritt genutzt werden.

4.2.3 PETSc Perf Vergleich

Zur Einschätzung der Genauigkeit der PETSc Analyse wurde sie in einem kleinen Beispiel mit einer Performanceanalyse verglichen. Dabei ist zu beobachten, dass für die komplette Ausführung die Ergebnisse signifikant unterschiedlich sind. Dies liegt daran, dass PETSc standardmäßig nur die Fließkommaoperationen in den internen Funktionen zählt. Damit die Fließkommaoperationen in den eigenen Analysebereichen (der PISM interne Code) zusätzlich gezählt werden, müsste der PISM Code mit `PetscLogFlops()` angepasst werden [BAA+20, S. 199f]. Da allerdings die laufzeitintensiven Abschnitte alle über PETSc realisiert wurden, schränkt dies die Verwendung der PETSc-Analyse nicht ein und eine Code Anpassung war nicht notwendig.



(a) 16 km Auflösung - ein Knoten

(b) 4 km Auflösung - ein Knoten

(c) 16 km Auflösung - multiple Knoten

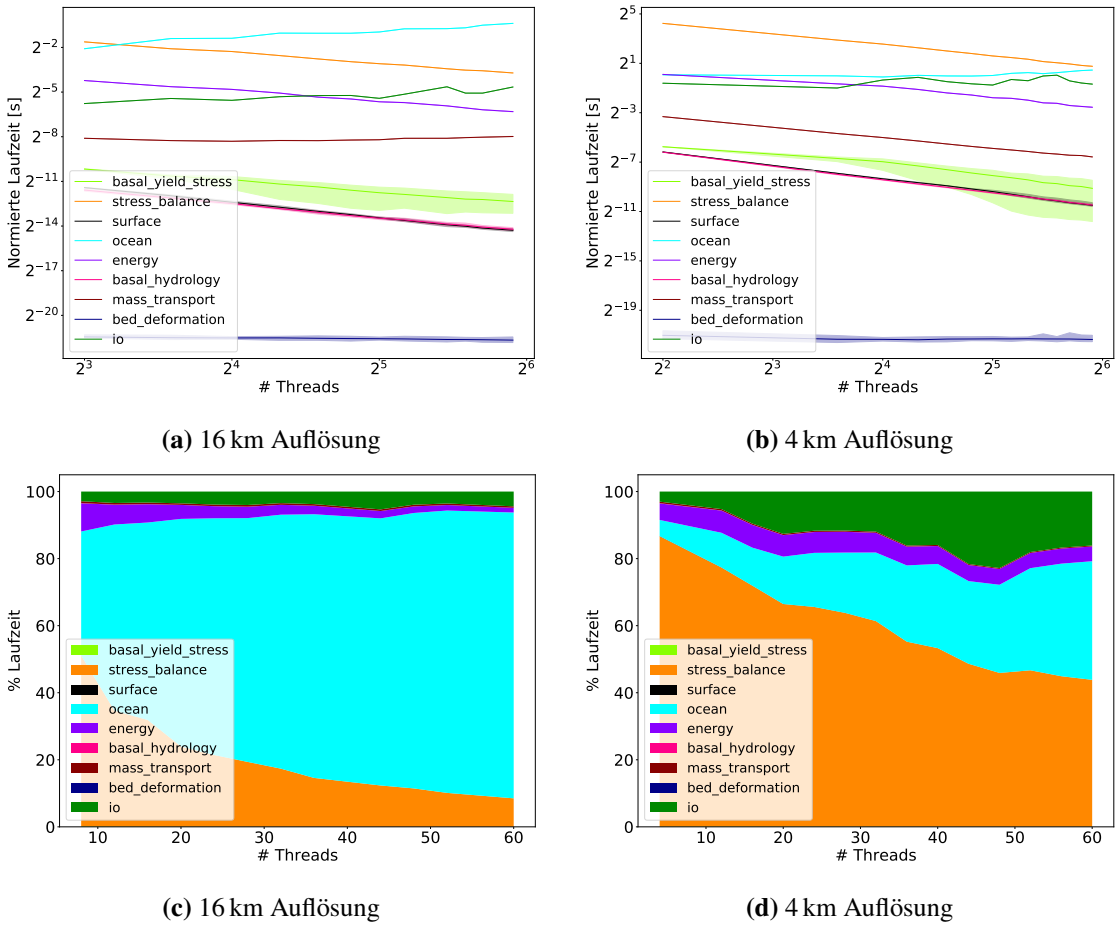
(d) 4 km Auflösung - multiple Knoten

Abbildung 4.3: Normierter Vergleich von auf sgscl1 gemessener Laufzeiten.

4.3 Laufzeitanalyse

Abbildung 4.3 zeigt die durchschnittliche Zeit, die die Threads im Durchschnitt jeweils mit einer einzigen Ausführung des Programmabschnittes auf sgscl1 verbracht hat. Das entspricht der durchschnittlich gemessenen, parallelen Laufzeit. Abbildung 4.3a und Abbildung 4.3b repräsentieren dabei die Laufzeiten, die auf einem Knoten gemessen wurden. Ein Knoten von sgscl1 hat vier physische Kerne, die mittels Hyper-Threading-Technologie [MBH+02] jeweils zwei also insgesamt acht simultane Threads unterstützen. Wie zu erkennen ist, skaliert das Szenario mit 4 km Auflösung (Abbildung 4.3b) auf einem sgscl1 Knoten bis zu einer Verwendung von vier Threads. Daraus lässt sich ableiten, dass PISM von Simultaneous Multithreading (SMT) nicht profitiert.

Das Verhalten von der Simulation mit 16 km Auflösung (Abbildung 4.3a) ist ähnlich. Hier ist allerdings zu beobachten, dass die Skalierung schon ab zwei Threads deutlich schlechter ist. Dies deutet auf eine Limitierung durch die Speicherbandbreite hin, da die Kommunikation zwischen Threads auf einem Knoten mit gemeinsamen Speicher im allgemeinen nicht über das Netzwerk, sondern über den gemeinsamen Speicher abläuft. Deutlich wird die Limitierung in Abbildung 4.3d und vor allem in Abbildung 4.3a. Hier wird die Laufzeit mit steigender Parallelität nicht verkürzt, sondern, bedingt durch das Skalierungsverhalten von `ocean`, verlängert sich die Laufzeit mit



(a) 16 km Auflösung

(b) 4 km Auflösung

(c) 16 km Auflösung

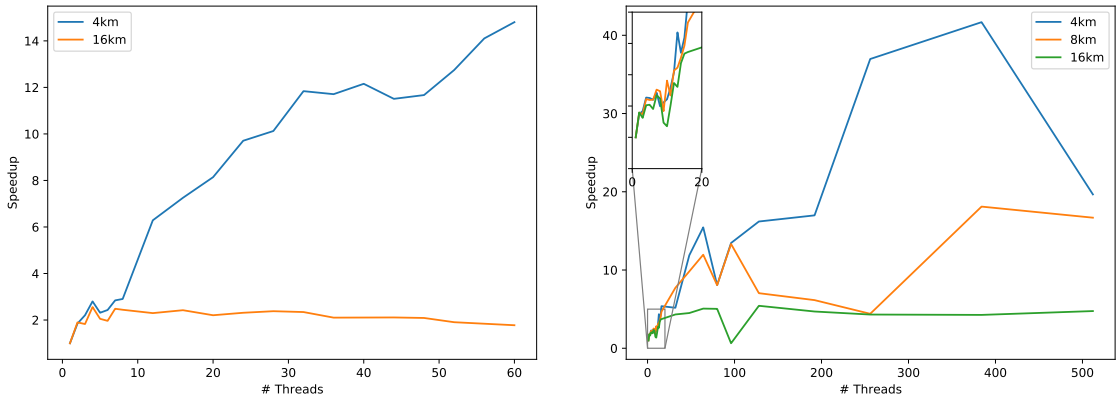
(d) 4 km Auflösung

Abbildung 4.4: Skalierungsverhalten der einzelnen Codeabschnitte gemessen auf sgscl1.

zunehmendem Grad an Parallelität. Bei der Messung auf mehreren Knoten, werden auf den Knoten nur die jeweiligen Hardware Threads genutzt, da Abbildung 4.3a und Abbildung 4.3b zeigen, dass der Einsatz von SMT keinen positiven Effekt hat. Weiter lässt sich in Abbildung 4.3 deutlich erkennen, dass ein Hotspot in `stress_balance` liegt. Abbildung 4.3 lässt des Weiteren vermuten, dass `ocean` und `energy` nicht gut skalieren. Generell fallen nur bei `ocean` und `stress_balance` starke Reaktionen auf die Granularität der Parallelität auf.

4.3.1 Starke Skalierung

Die bisher in diesem Kapitel betrachteten Laufzeitverhalten in Abbildung 4.3, kann man als Analyse der starken Skalierung einordnen. Die Problemgröße bleibt konstant, während die Laufzeit auf ein Verhalten in Abhängigkeit der Ressourcen untersucht wird. Stellt man die Laufzeiten wie in Abbildung 4.4 dar, kann das Skalierungsverhalten abgelesen werden. Abbildung 4.4a und Abbildung 4.4b stellen dabei die gleichen Daten dar, wie Abbildung 4.3c und Abbildung 4.3d, nur sind hier beide Achsen logarithmisch skaliert. Dabei bilden die Linien die Mittelwerte aller Threads ab und die eingefärbten Bereiche die Schwankungen zwischen dem schnellsten Thread und dem langsamsten Thread. Beim Vergleich der beiden Graphen ist zu sehen, dass, bis auf bei `mass_transport`, das



(a) Speedup auf sgscl1

(b) Speedup auf dem PIK-Cluster

Abbildung 4.5: Speedup Vergleich der starken Skalierung auf unterschiedlicher Hardware.

Skalierungsverhalten in diesem Ausschnitt unabhängig von der Auflösung ist. `mass_transport` skaliert mit zusätzlichem Hardwareeinsatz bei 4 km Auflösung. Bei 16 km Auflösung hingegen nicht mehr. Hier wird durch `mass_transport` mit steigendem Parallelisierungsgrad die Laufzeit negativ beeinflusst. `stress_balance`, `energy`, `basal_yield_stress`, `surface`, `basal_hydrology` und `mass_transport` skalieren mit steigender Parallelität um einen sehr ähnlichen Faktor. Die Laufzeit von `io` ist relativ unabhängig von dem Grad der Parallelität, bedingt dadurch wird es ab einem gewissen Parallelitätsgrad zusammen mit `ocean` zu einem dominierenden Faktor. `ocean` wird stark negativ von zusätzlicher Hardware beeinflusst. Ab einem gewissen Punkt der Parallelität dominiert `ocean` die Laufzeit, wie in Abbildung 4.3c dargestellt, deutlich. Auffällig ist zudem auch, dass die Kurven im doppelt logarithmischen Plot weit weg von einer im 45° Winkel nach rechts unten fallenden Geraden sind, welche eine optimale Skalierung beschreiben würde.

In Abbildung 4.4c und Abbildung 4.4c wurden die Anteile, den die jeweiligen Abschnitte an der Gesamtlaufzeit haben, visualisiert. Auch dies verdeutlicht, dass der Anteil der skalierenden Abschnitte an der Gesamtlaufzeit mit steigender Parallelität immer geringer wird, während der Anteil der nicht skalierenden Abschnitte wächst.

Bei der Speedup Analyse der starken Skalierung fällt auf, dass der Speedup nachlässt, wenn die Anzahl der Threads die Kapazität eines Knotens übersteigt, also die Kommunikation zwischen den einzelnen Knoten hinzukommt. Deutlich zu sehen ist dies in Abbildung 4.5a. Das bedeutet, dass die Laufzeit der Kommunikation nicht komplett durch die Laufzeit der Berechnungen verborgen werden kann und damit die Gesamtlaufzeit negativ beeinflusst. Interessant ist auch, dass bei gleicher Auflösung auf den unterschiedlichen Clustern unterschiedliche Speedups erreicht werden. Dies ist beim Vergleich von Abbildung 4.5a und Abbildung 4.5b zu sehen. Bei Betrachtung der Graphen mit niedriger Auflösung ist zu sehen, dass Amdahls Gesetz für die starke Skalierung greift, da offensichtlich kein weiterer Speedup mehr erreicht wird, wenn weitere Hardware hinzugenommen wird.

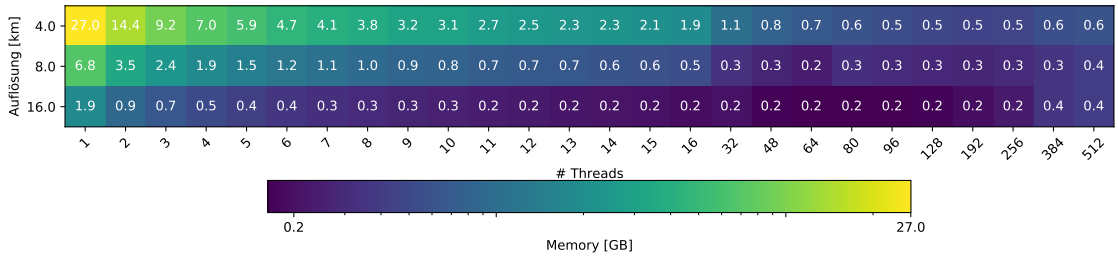


Abbildung 4.6: Haupspeicherverbrauch in Abhängigkeit von Auflösung und Threadanzahl, gemessen auf dem PIK-Cluster.

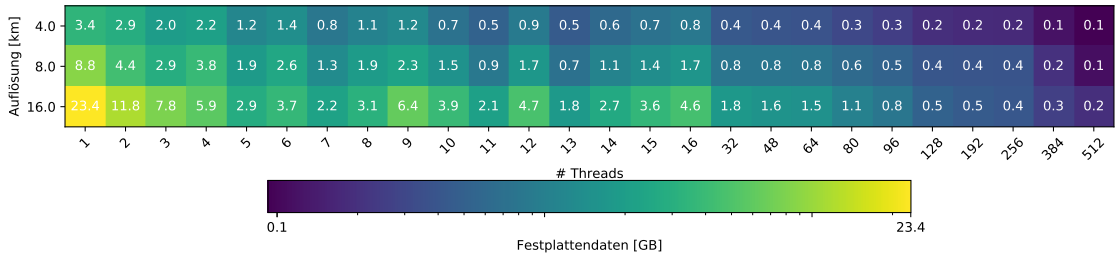
Der Ursprung der weiteren Sprünge in Abbildung 4.5 lässt sich möglicherweise auf die Netzwerkauslastung zurückführen, da das Netzwerk auf dem PIK-Cluster mit anderen Jobs geteilt wird. Hier haben sich starke zeitlich abhängige Schwankungen gezeigt. Für die Messungen auf sgscl1 kann dies ausgeschlossen werden, da hier während der Analyse keine weiteren Programme ausgeführt wurden.

4.4 Speicherbetrachtung

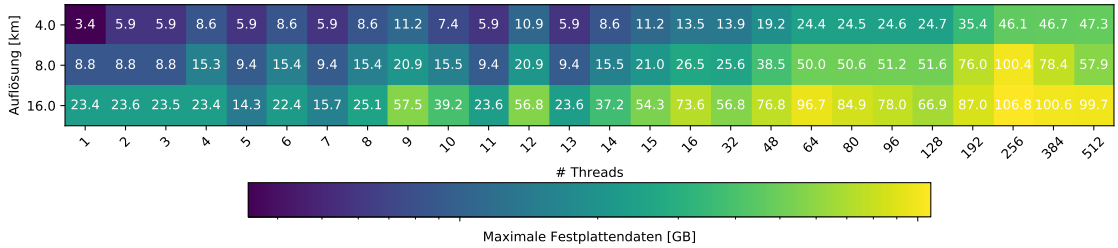
Abbildung 4.6 visualisiert den Hauptspeicherbedarf pro Knoten in Abhängigkeit der Anzahl an Threads sowie der Auflösung. Auffällig ist, dass mit steigender Anzahl an Threads ein Sweetspot kommt, nach welchem mit steigendem Parallelisierungsgrad, die Hauptspeicherauslastung wieder ansteigt. Die Ursachen für dieses Verhalten können Daten sein, die redundant in mehreren Threads gespeichert werden und sich so aufsummieren. Eine weitere Möglichkeit wäre, dass ein Thread zu allen weiteren große Kommunikationsbuffer benötigt. Unabhängig der Ursache, führt dieses Verhalten unweigerlich dazu, dass nicht beliebig gut skaliert werden kann. Interessanterweise scheint auch der Sweetspot bei 16 km sowie 4 km bei einer höheren Threadanzahl zu sein, als bei 8 km. Damit lässt sich ohne Vergleichsläufe in unterschiedlicher Auflösung nicht abschätzen, wie viele Threads gestartet werden müssen, damit der Hauptspeicherbedarf minimiert wird. Da mit feinerer Auflösung der Hauptspeicherbedarf dafür entscheidend ist, ob die Simulation durchgeführt werden kann oder nicht, wäre dies eine wünschenswerte Eigenschaft gewesen.

4.5 Festplattenauslastung

Während eines Simulationsdurchlaufs fallen Zwischenergebnisse an, die in PISM direkt in Dateien gespeichert werden. Abbildung 4.7a beschreibt die dabei durchschnittlich generierte Datenmenge. Die Auflösungen sind dabei untereinander nur bedingt vergleichbar, da, wie in Tabelle 3.2 dargestellt, die zeitliche Auflösung sowie der Simulationszeitraum unterschiedlich sind. Beim Vergleich von z. B. 16 km und 8 km muss man also berücksichtigen, dass bei 16 km die Zeitreihe 100 Einträge hat, während sie bei 8 km nur 10 beinhaltet. Das Verhältnis der Datenmengen auf einem Thread zwischen 16 km und 8 km beträgt allerdings nicht $\frac{1}{10}$ sondern ungefähr $\frac{44}{117}$. Daraus lässt sich abschätzen, dass bei einer Halbierung der Auflösung ungefähr das 2,6-fache an Daten geschrieben wird.



(a) Durchschnittliche generierte Datenmenge in Abhängigkeit von Auflösung und Threadanzahl mit logarithmisch skaliertem Farbschema, gemessen auf dem PIK-Cluster.



(b) Maximale generierte Datenmenge in Abhängigkeit von Auflösung und Threadanzahl mit logarithmisch skaliertem Farbschema, gemessen auf dem PIK-Cluster.

Abbildung 4.7: Maximale und durchschnittliche Datenrate in Abhängigkeit von Auflösung und Threadanzahl. Gemessen auf dem PIK-Cluster.

Kontraintuitiv wird das Verhalten, wenn die Anzahl der Threads gesteigert wird. Hier verändert sich das Verhältnis zwischen den Auflösungen deutlich. Teilweise ändert es sich sogar so stark, dass 16 km und 8 km insgesamt ungefähr gleich viele Daten generieren. Noch deutlicher wird das Verhalten, bei der Betrachtung von Abbildung 4.7b. Hier wird nicht die durchschnittlich generierte Datenmenge visualisiert, sondern die maximale Datenmenge, die durch einen Thread generiert wurde. Dabei wird auch deutlich, dass die Ergebniserstellung nur durch einen Thread behandelt wird und hier sehr große Datenmengen zustande kommen können. Im Allgemeinen nimmt die generierte Datenmenge mit steigendem Parallelisierungsgrad bei gleichem Szenario deutlich zu, obwohl die Ergebnisse die Gleichen sein sollten, somit auch gleich viel Speicher benötigen.

Die Datenmengen sind mit bis zu 107 GB so groß, dass sie erheblichen Einfluss auf die Gesamtlaufzeit haben können. Wenn man bedenkt, dass ein Festplattenlaufwerk (HDD) ungefähr eine Schreibrate von 100 MB/s hat, und damit lediglich das Schreiben der Ergebnisse knapp 18 min dauern würde, wird deutlich, dass PISM auf ein performantes, paralleles Dateisystem angewiesen ist. Da die Daten beim Schreiben allerdings Netzwerkressourcen belegen, beeinträchtigt dies auch die Performance negativ. Angenommen, das Dateisystem würde die vollen 56 Gbit/s, der auf dem PIK-Cluster zur Verfügung stehenden Netzwerkbandbreite, an Schreibrate unterstützen, würde dies, bei einer Datenmenge von 107 GB, 15,3 s beanspruchen. Da alle Daten allerdings erst in einem Thread gesammelt werden, und dabei die gleiche Datenmenge anfällt, sind mindestens 30 s der Simulationszeit Ergebnisdatengenerierung. Da während dieser Zeit das Netzwerk komplett ausgelastet ist, kann ansonsten nicht zwischen den Threads kommuniziert werden, sodass die Simulation auch nicht

asynchron weitergeführt werden kann. Damit ist dies, vor allem bei langsamen Dateisystemen oder geringer Netzwerkbandbreite, ein eindeutiger Flaschenhals. Bei 10 Gbit Ethernet würde das Ergebnisse schreiben keine 30 s, sondern 28 min benötigen.

5 Hotspot Analysen

In diesem Kapitel werden die laufzeitintensiven Programmabschnitte näher beleuchtet. Wie im vorherigen Kapitel 4 herausgefunden, sind die kritischen Abschnitte `io`, `ocean` und `stress_balance`. Das Kapitel beginnt mit einer Übersicht über das in `io` implementierte Ein-und Ausgabeverhalten von PISM, gefolgt von einer Erklärung für das Laufzeitverhalten des `ocean`-Abschnittes. Der Fokus in diesem Kapitel liegt auf `stress_balance`. Nach einer detaillierten Codebetrachtung, bis hin zum eigentlichen Hotspot, dem Lösen eines Gleichungssystems, folgt eine detaillierte Performanceanalyse des, für das Lösen genutzten, GMRES Lösers.

5.1 Abschnitt: io

Wie beispielsweise in Abbildung 4.4 zu sehen ist, hat die Ein-/Ausgabe ein schlechteres Skalierungsverhalten, als die meisten anderen betrachteten Abschnitte. Wie in Abschnitt 4.5 gezeigt, ist die Zeit, bedingt durch die große Datenmenge bei feiner Gitterauflösung, die PISM mit dem Schreiben von Ausgabedateien benötigt, erheblich. Damit hat die Anzahl und Art der Ausgaben eine signifikante Auswirkung auf die gesamte Performance. Je größer die Daten werden, desto mehr führt die Einschränkung der Ausgabe auf das Wesentliche zu einer Verkürzung der Gesamtlaufzeit. Außerdem ist zu beobachten, dass, sowohl bei den Eingabe- als auch bei den Ausgabedateien, für eine möglichst geringe Laufzeit wichtig ist, dass die Reihenfolge der Dimensionen der NetCDF-Variablen mit der vom PISM im Speicher verwendeten Reihenfolge (`time,y,x,z`) übereinstimmt.

PISM unterstützt paralleles Einlesen und Schreiben unter Verwendung von parallel NetCDF, PnetCDF oder ParallelIO. Über das Kommandozeilenargument `-o_format` kann für die Ausgabe zwischen verschiedenen Möglichkeiten gewählt werden, sofern PISM damit konfiguriert und gebaut wurde.

netcdf3 Standardkonfiguration, Serielle E/A von Rank 0 in NetCDF-3 Dateien.

pio_netcdf serielles Lesen und Schreiben, wobei die Datenaggregation in ParallelIO stattfindet.

pio_netcdf4c Serielles Lesen und Schreiben wie mit pio_netcdf, aber in auf HDF5 basierende NetCDF-4 Dateien.

netcdf4_parallel Paralleles E/A in auf HDF5 basierende NetCDF-4 Dateien.

pnetcdf Paralleles E/A mittels PnetCDF in CDF5 Dateien.

pio_pnetcdf Wie pnetcdf, nur basierend auf ParallelIO und nicht auf PnetCDF.

pio_netcdf4p pio_netcdf4c in parallel.

Hier empfiehlt es sich auf jedem neuen System die auf dem jeweiligen System möglichen Parameter in kurzen Läufen zu testen. Bei den Tests hat sich ergeben, dass für diese Läufe die Anzahl an Dimensionen keinen Einfluss hat. Sie sollte allerdings groß genug sein, dass signifikante Laufzeitunterschiede entstehen können und möglichst mit der gleichen Anzahl an Threads wie die geplanten Läufe sein. Da die Ein- und Ausgabeperformance stark von dem zugrundeliegenden Filesystem abhängig ist, wurden sie in der weiteren Analyse nicht weiter aufgegriffen. Die in dieser Analyse gemessenen Laufzeiten auf den Rechenclustern wurden alle mit den jeweiligen Netzwerkspeichersystemen gemessen. Damit sind sie weder untereinander vergleichbar, noch für andere Umgebungen repräsentativ [Pa15, S. 83ff].

5.2 Abschnitt: ocean

Die `ocean` Komponenten des PISM Modells liefern die Schelfeistemperatur sowie den Schelfeismassenfluss [Pa15, S. 229f].

Die Schelfeistemperatur wird als Dirichlet-Randbedingung in dem Energieerhaltungscode verwendet. Der Schelfeismassenfluss wird als Quelle in der Massenkontinuitäts-(/Transport-)Gleichung verwendet. Ein positiver Fluss entspricht einem Eisverlust, damit ist dieser Schelfeismassenfluss eine Schmelzrate [WMH+11].

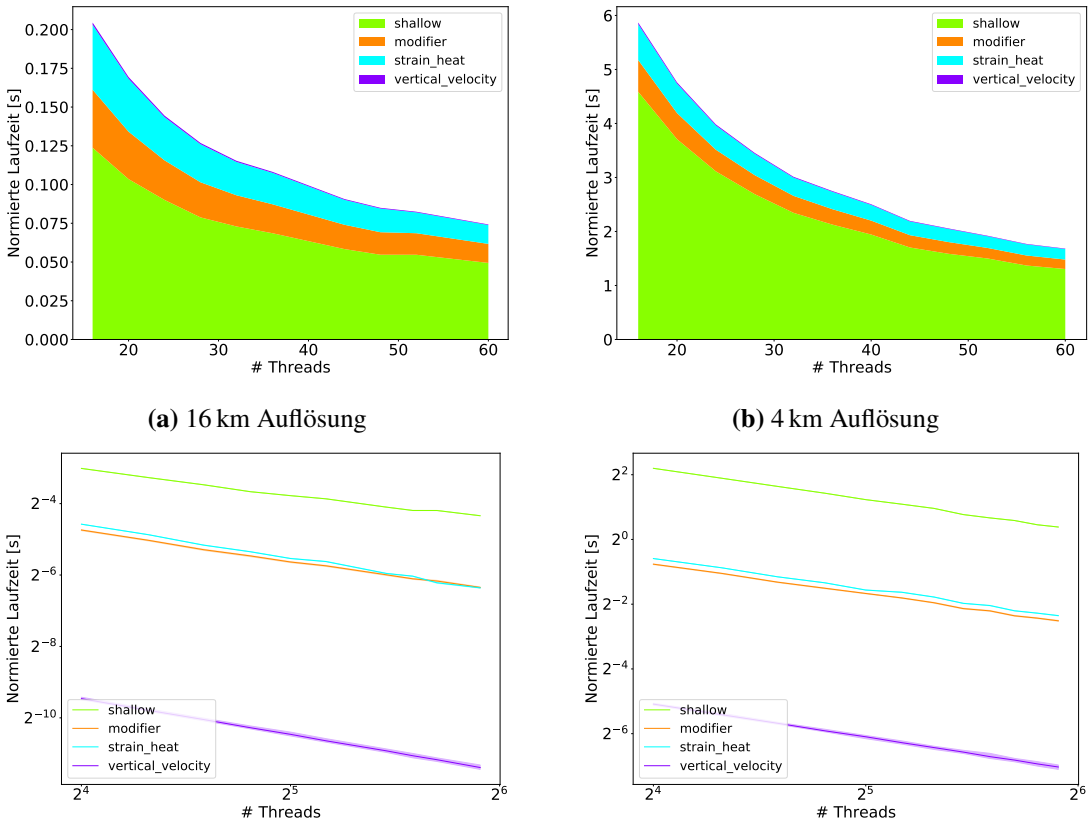
Wie in den Laufzeitgraphen zu sehen ist, ist `ocean` aktuell rein sequentiell implementiert. Daher profitiert dieses Modul nicht von der Parallelisierung über mehrere Threads. Die benötigten Daten müssen von immer mehr Knoten zusammen kopiert und die Ergebnisse schließlich wieder verteilt werden. Dies erklärt das negative Skalierungsverhalten, welches z. B. in Abbildung 4.4 zu erkennen ist.

5.3 Abschnitt: stress_balance

Um die einzelnen Abschnitte von `stress_balance` analysieren zu können, wurde der Programmcode für die weitere Analyse mit weiteren PETSc-Profiling-Anotationen versehen. Bei der genaueren Laufzeitbetrachtung der Abschnitte innerhalb von Stress Balance (vergleiche Abbildung 5.1), fällt auf, dass der laufzeitintensive Anteil eindeutig `shallow` ist. Wie aus den doppelt logarithmischen Abbildungen 5.1c und Abbildung 5.1d abzulesen ist, haben die vier Komponenten ungefähr das gleiche Skalierungsverhalten bis hoch zu 60 Threads. Dies deutet darauf, dass auch für weitere Parallelisierung `shallow` der Hotspot bleibt und weiter analysiert werden sollte.

5.3.1 shallow

In `shallow` ist die SSA implementiert, welche die komplette Laufzeit von `shallow` darstellt. Abbildung 5.2 visualisiert die Laufzeiten der einzelnen Programmcodeabschnitte der SSA Implementierung sowohl in 4 km als auch in 16 km Auflösung. Unabhängig der Auflösung ist der `solve` Abschnitt eindeutig der laufzeitdominierende. In den doppelt logarithmischen Plots in



(a) 16 km Auflösung

(b) 4 km Auflösung

(c) 16 km Auflösung doppelt logarithmisch

(d) 4 km Auflösung doppelt logarithmisch

Abbildung 5.1: Laufzeitanalyse unterschiedlicher Codeabschnitte in `stress_balance` in Abhängigkeit der Anzahl an genutzter Threads. Die Laufzeit wird normiert und gemittelt dargestellt. In den logarithmischen Plots gibt der schraffierte Bereich den Bereich der Laufzeiten der unterschiedlichen Threads an. Alle Daten wurden auf sgscl1 gemessen.

Abbildung 5.2c und Abbildung 5.2d sieht man, dass die anderen Komponenten ein ähnliches Skalierungsverhalten wie `solve` haben, oder eine vernachlässigbare Laufzeit haben, die nicht negativ skaliert. Somit kann davon ausgegangen werden, dass der `solve` Abschnitt, bei jeder Thread-Anzahl der dominierende Anteil von `stress_balance` ist.

5.3.2 solve

Der Codeausschnitt in 5.1 zeigt, wie das Lösen der Stressgleichung (siehe Unterabschnitt 3.1.4) in PISM mit der Finiten-Differenzen-Option implementiert ist. Erst wird versucht, das Gleichungssystem über Picarditerationen zu lösen. Wird hierbei nach einer begrenzten Anzahl an Iterationen keine Konvergenz erreicht, wird eine Unter-Relaxation versucht. Divergiert auch diese, wird versucht, ob eine Über-Relaxation zu einer Konvergenz führt.

Listing 5.1 `SSAFD::solve` Ausschnitt ohne Profiling-Anotationen und Logging.

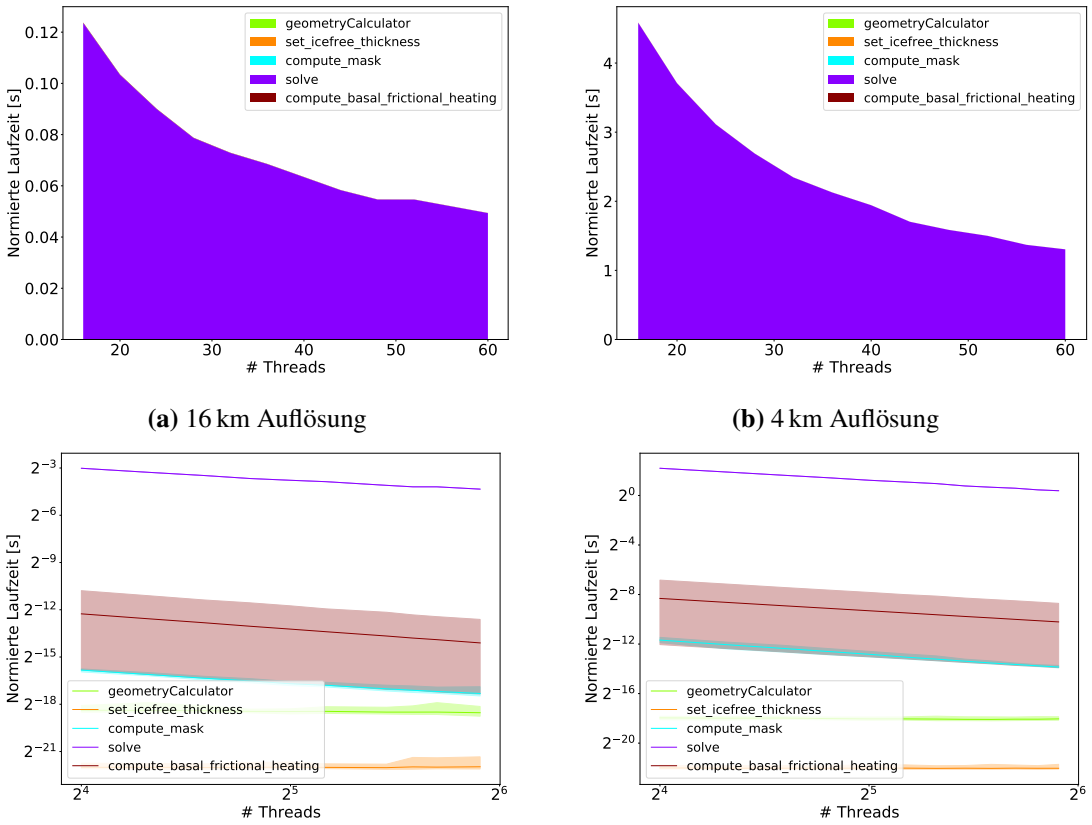
```
void SSAFD::solve(const Inputs &inputs) {
  // Store away old SSA velocity (it might be needed in case a solver fails).
  m_velocity_old.copy_from(m_velocity);
  assemble_rhs(inputs);
  compute_hardav_staggered(inputs);
  for (unsigned int k = 0; k < 3; ++k) {
    try {
      if (k == 0) { // default strategy
        picard_iteration(inputs, m_config->get_number("stress_balance.ssa.epsilon"), 1.0);
        break;
      } else if (k == 1) { // try underrelaxing the iteration
        const double underrelax =
        ↪  m_config->get_number("stress_balance.ssa.fd.nuH_iter_failure_underrelaxation");
        picard_iteration(inputs, m_config->get_number("stress_balance.ssa.epsilon"), underrelax);
        break;
      } else if (k == 2) { // try over-regularization
        picard_strategy_regularization(inputs);
        break;
      } else {  // if we reached this, then all strategies above failed
        write_system_petsc("all_strategies_failed");
        throw RuntimeError(PISM_ERROR_LOCATION, "all SSAFD strategies failed");
      }
    } catch (PicardFailure &f) {} // proceed to the next strategy
  }

  // Post-process velocities if the user asked for it
  ...
}
```

Listing 5.2 `SSAFD::picard_iteration` Ausschnitt ohne Profiling-Anotationen und Logging.

```
void SSAFD::picard_iteration(const Inputs &inputs, double nuH_regularization, double
↪  nuH_iter_failure_underrelax) {

  if (m_default_pc_failure_count < m_default_pc_failure_max_count) { // Give BJACOBI another shot if
  ↪  we haven't tried it enough yet
    try {
      pc_setup_bjacobi();
      picard_manager(inputs, nuH_regularization, nuH_iter_failure_underrelax);
    } catch (KSPFailure &f) {
      m_default_pc_failure_count += 1;
      pc_setup_asm();
      m_velocity.copy_from(m_velocity_old);
      picard_manager(inputs, nuH_regularization, nuH_iter_failure_underrelax);
    }
  } else { // otherwise use ASM
    pc_setup_asm();
    picard_manager(inputs, nuH_regularization, nuH_iter_failure_underrelax);
  }
}
```



(a) 16 km Auflösung

(b) 4 km Auflösung

(c) 16 km Auflösung doppelt logarithmisch

(d) 4 km Auflösung doppelt logarithmisch

Abbildung 5.2: Laufzeitanalyse unterschiedlicher Codeabschnitte in `SSAFD` in Abhängigkeit der Anzahl an genutzter Threads. Die Laufzeit wird normiert und gemittelt dargestellt. In den logarythmischen Plots gibt der schraffierte Bereich den Bereich jener Laufzeiten der unterschiedlichen Threads an.

Dabei wird, wie in dem Programmcodeausschnitt für die normale Picarditeration in 5.2 zu sehen ist, bei jedem Versuch überprüft, ob der Block-Jacobi-Vorkonditionierer Ansatz schon mehr als fünf mal (`m_default_pc_failure_max_count`) nicht weit genug konvergiert oder divergiert ist. Konvergiert der Löser dabei nicht innerhalb von 300 Schritten, gibt der `picard_manager` (5.3) einen Fehler zurück. Diese 300 Schritte, sowie der Block-Jakobi, sind die Standardwerte und können über Kommandozeilenparameter bei Bedarf mit `-ssafd_pc_type` bzw. `-ssafd_picard_maxi` überschrieben werden. Die Konvergenztoleranz kann mit `-ssafd_ksp_rtol` angepasst werden.

Konvergiert der Löser mit dem Block-Jacobi-Vorkonditionierer nicht unter die Toleranzgrenze, wird ein erneuter Lösungsversuch mit einem Additive-Schwarz-Vorkonditionierer [SBG04; WD87] gestartet. Konvergiert der Löser mit dem Block-Jacobi-Vorkonditionierer fünf Mal nicht unter die Toleranzgrenze, wird in den weiteren Aufrufen direkt auf den Additive-Schwarz-Vorkonditionierer zurückgegriffen.

Listing 5.3 `SSAFD::picard_manager` Ausschnitt ohne Profiling-Anotationen und Logging.

```
void SSAFD::picard_manager(...) {
  ... //initialisation etc
  for (unsigned int k = 0; k < max_iterations; ++k) {
    // in preparation of measuring change of effective viscosity:
    m_nuH_old.copy_from(m_nuH);
    assemble_matrix(inputs, true, m_A);
    // Call PETSc to solve linear system by iterative method; "inner iteration":
    ierr = KSPSetOperators(m_KSP, m_A, m_A);
    PISM_CHK(ierr, "KSPSetOperator");
    ierr = KSPSolve(m_KSP, m_b.vec(), m_velocity_global.vec());
    PISM_CHK(ierr, "KSPSolve");
    ierr = KSPGetConvergedReason(m_KSP, &reason);
    PISM_CHK(ierr, "KSPGetConvergedReason");

    if (reason < 0)  throw KSPFailure(KSPConvergedReasons[reason]);
    ierr = KSPGetIterationNumber(m_KSP, &ksp_iterations);
    PISM_CHK(ierr, "KSPGetIterationNumber");
    ksp_iterations_total += ksp_iterations;
    ...// limit ice speed Communicate, update viscosity, check for viscosity convergence
    outer_iterations = k + 1;
    if (nuH_norm == 0  nuH_norm_change / nuH_norm < ssa_relative_tolerance)  goto done;
  } // outer loop (k)
  throw PicardFailure(...);

 done: ...
}
```

In dem Analyseszenario auf welchem diese Arbeit basiert, ist die Standard-Picarditeration mit dem Block-Jacobi Vorkonditionierer immer konvergiert, sodass deren Laufzeiten hier als eine Art „untere Grenze“ genommen werden kann. Triviale Szenarien benötigen natürlich weniger Iterationen, bestimmte Szenarien hingegen benötigen die Über- bzw. Unterrelaxation und haben dadurch bedingt schlechtere Laufzeiten.

Der `picard_manager` ist dann dafür verantwortlich, das Gleichungssystem aufzusetzen, und mittels PETSc Funktionen zu lösen. Der Standardlöser, der mit `KSPSolve` aufgerufen wird, ist ein GMRES(k).

5.3.3 GMRES

GMRES [SS86] benötigt pro Iteration eine Matrix-Vektor-Multiplikation und eine Reihe von Skalarprodukten, deren Anzahl pro Iterationsschritt um eins steigt, ebenso wie die Anzahl der vollbesetzten zu speichernden Basisvektoren, da in jedem GMRES-Iterationschritt auf alle Basisvektoren zugegriffen wird. Da damit der Aufwand und der benötigte Speicher linear mit der Iterationsanzahl ansteigt, ist es üblich, nach k Schritten die berechnete Basis zu verwerfen und die Iteration mit der aktuellen Näherungslösung neu zu starten. Dies ist dann der sogenannte GMRES(k)-Algorithmus. In PISM ist der Standartwert für $k = 300$. Die PETSc-GMRES(k)-Implementierung basiert auf dem von Ghysels et al. vorgestellten Algorithmus [GAMV13]. Untersucht wird der Algorithmus wie er von Ghysels et al. vorgestellt wurde, und in 5.1 dargestellt ist. Der Algorithmus wurde also nicht aus der PETSc-Implementierung extrahiert.

Algorithmus 5.1 GMRES(k) mit Givens Rotation

1: **if** $||\vec{r_0}||_2 \leq TOL$ **then**
2: **return**
3: **end if**

$(n-1)\cdot fma + mult + sqrt + leq \Rightarrow 2n+1,\ n\cdot load,\ 1\cdot store$

4: $\vec{v_1} \leftarrow \frac{\vec{r_0}}{||\vec{r_0}||_2}$
5: $\gamma_1 \leftarrow ||\vec{r_0}||_2$

$n\cdot div \Rightarrow n,\ (n+1)\cdot load,\ (n+1)\cdot store$

6: **for** $j = 1, \ldots, k$ **do**
7: $\vec{q} \leftarrow A \cdot \vec{v_j}$ $\quad (n^2-n)\cdot fma + n\cdot mult \Rightarrow (2n^2-n), (n^2+n)\cdot load, n\cdot store$
8: **for** $i = 1, \ldots, j$ **do**
9: $h_{i,j} \leftarrow \vec{v_i}^T \vec{q}$
10: **end for**

$j\cdot((n-1)\cdot fma + mult) \Rightarrow j\cdot(2n-1),\ (j\cdot n+n)\cdot load,\ j\cdot store$

11: $\vec{w_j} \leftarrow \vec{q} - \sum_{i=1}^{j} h_{i,j}\cdot\vec{v_i}$ $\quad j\cdot n\cdot mult + n\cdot sub + j\cdot(n-1)\cdot add \Rightarrow (j+2)n-j,\ (j\cdot n+n+j)\cdot load, n\cdot store$

12: $h_{j+1,j} \leftarrow ||\vec{w_j}||_2$ $\quad (n-1)\cdot fma + mult + sqrt \Rightarrow 2n,\ n\cdot load,\ 1\cdot store$

13: **for** $i = 1, \ldots, j-1$ **do**
14: $\begin{pmatrix} h_{i,j} \\ h_{i+1,j} \end{pmatrix} \leftarrow \begin{pmatrix} c_{i+1} & s_{i+1} \\ -s_{i+1} & c_{i+1} \end{pmatrix} \cdot \begin{pmatrix} h_{i,j} \\ h_{i+1,j} \end{pmatrix}$
15: **end for**

$(j-1)\cdot(2\cdot(mult+fma)+neg) \Rightarrow 7j-7,\ (j-1)\cdot 2\cdot 4\cdot load,\ (j-1)\cdot 2\cdot store$

16: $\beta \leftarrow \sqrt{h_{j,j}^2 + h_{j+1,j}^2}$ $\quad mult + fma + sqrt \Rightarrow 4, 2\cdot load, 1\cdot store$

17: $s_{j+1} \leftarrow \frac{h_{j+1,j}}{\beta}$
18: $c_{j+1} \leftarrow \frac{h_{j,j}}{\beta}$
19: $h_{j,j} \leftarrow \beta$
20: $\gamma_{j+1} \leftarrow -s_{j+1}\cdot\gamma_j$
21: $\gamma_j \leftarrow c_{j+1}\cdot\gamma_j$

$neg + 2\cdot div + 2\cdot mult \Rightarrow 5,\ 4\cdot load,\ 5\cdot store$

22: **if** $|\gamma_{j+1}| \geq TOL$ **then**
23: $\vec{v_{j+1}} \leftarrow \frac{\vec{w_j}}{h_{j+1,j}}$ $\quad n\cdot div \Rightarrow n, (n+1)\cdot load, n\cdot store$
24: **else**
25: **for** $i = j, \ldots, 1$ **do**
26: $y_i \leftarrow \frac{1}{h_{i,i}}\left(\gamma_i - \sum_{l=i+1}^{j} h_{i,l}\cdot y_l\right)$
27: **end for**

$j(div+mult) + (j-1)\cdot(mult+sub) + (\frac{j^2}{2} + \frac{3j}{2} + 1)\cdot fma \Rightarrow j^2+7j,\ (3j+\frac{j^2-j}{2})\cdot load,\ j\cdot store$

28: $\vec{x} \leftarrow \vec{x_0} + \sum_{i=1}^{j} y_j\cdot\vec{v_i}$ $\quad j\cdot n\cdot fma \Rightarrow 2jn, (jn+n+1)\cdot load, n\cdot store$
29: **return**
30: **end if**
31: **end for**

5.3.3.1 Analyse

Da PISM standardmäßig mit doppelter Genauigkeit rechnet, wird für die Analyse mit acht Byte für jeden *load* / *store* gerechnet. Für die Konsistenz zur Berechnung der Leistungsmerkmale aus Tabelle 3.4 werden FMA-Operationen als zwei Fließkommaoperationen gerechnet. Das Gleichungssystem, das in `stress_balance` gelöst wird, hat doppelt so viele Zeilen und doppelt so viele Spalten wie die Anzahl an Gitterpunkten (siehe Tabelle 3.1), also $n = \#Datenpunkte \cdot 2$.

Um die erwartbare Performance einzuschätzen, wird 5.1 auf seine arithmetische Intensität untersucht. Dafür wird die Anzahl an Fließkommaoperationen und Speicheroperationen benötigt. In 5.1 ist die Anzahl an Fließkommaoperationen in rot, und die Anzahl an Speicheroperationen in blau und orange angegeben. Das `if` in Zeile 1 muss in jedem Fall ausgewertet werden. Dafür müssen $2 \cdot n + 1$ Fließkommaoperationen durchgeführt, und $n \cdot 8$ B geladen werden. Für den Fall, dass die Startschätzung genau genug ist, bricht der Algorithmus an dieser Stelle bereits ab. Ansonsten läuft der Algorithmus weiter. Unter der Annahme, dass für die Zeilen 4 und 5 die euklidische Norm des Residuums nur einmal berechnet und danach in einem Register gehalten wird, werden vor der Schleife beginnend in Zeile 6 weitere n Fließkommaoperationen und $(2n + 1) \cdot 8$ B Daten geladen beziehungsweise geschrieben.

Die Matrix-Vektor-Multiplikation in Zeile 7 benötigt ohne weiteres Vorwissen über die Matrix $2 \cdot n^2 - n$ Fließkommaoperationen und $(n+2) \cdot n \cdot 8$ B. Unter Berücksichtigung, dass A eine dünnbesetzte Matrix mit maximal 14 Einträgen pro Zeile ist, lässt es sich auf $27 \cdot n$ Fließkommaoperationen ($14 \cdot mult + 13 \cdot add$ in jeder Zeile) und $(14+2) \cdot n \cdot 8$ B für jeden Schleifendurchlauf einschränken.

Das modifiziert Gram-Schmidttsche-Orthogonalisierungsverfahren (Zeile 8-12) benötigt $(3j + 4) \cdot n - 2j$ Operationen und $(2j + (2j + 4)n + 1) \cdot 8$ B `load` / `store`, wenn in Zeile 9 darauf geachtet wird, dass jeder Eintrag von q nur einmal geladen wird.

Die Givens-Rotation in Zeile 13-15 benötigt $7j - 7$ und die Zeilen 16 bis 21 benötigen neun Fließkommaoperationen pro Iteration der Schleife in Zeile 6. Hier werden $(8j - 4) \cdot 8$ B geladen, und $(2j + 3) \cdot 8$ B geschrieben.

Das `if` in Zeile 22 überprüft, ob das Residuum noch größer oder gleich der Toleranzschranke ist und beendet den Algorithmus, sobald das Residuum kleiner der Toleranz ist. Dies bedeutet, der `else`-Zweig wird maximal einmal ausgeführt. Er benötigt $j^2 + 7j + 2jn$ Fließkommaoperationen und $\left(\frac{1}{2}j^2 + \frac{7}{2}j + 2n + jn + 1\right) \cdot 8$ B `load` / `store`. Der `if`-Zweig benötigt für die restlichen Iterationen jeweils n Divisionen, sowie $(2n + 1) \cdot 8$ B `load` / `store`.

Damit lassen sich die Fließkommaoperationen für die Schleife beginnend in Zeile 6 zusammenfassen mit:

$$3jn + 5n + 5j + 2 + \begin{cases} n^2 - n & \text{volle Matrix} \\ 27n & \text{dünnbesetzte Matrix} \end{cases} \tag{5.1}$$

für jede Iteration die im `if`-Zweig von Zeile 22 endet, und

$$j^2 + 5jn + 4n + 12j + 2 + \begin{cases} n^2 - n & \text{volle Matrix} \\ 27n & \text{dünnbesetzte Matrix} \end{cases} \tag{5.2}$$

für die Iteration die in dem `else`-Zweig endet.
Die Speicheroperationen für die Schleife beginnend in Zeile 6 lassen sich zusammenfassen mit:

$$2jn + 6n + 12j + 4 + \begin{cases} n^2 + 2n & \text{volle Matrix} \\ 16n & \text{dünnbesetzte Matrix} \end{cases}, \tag{5.3}$$

für jede Iteration die im `if`-Zweig von Zeile 22 endet, und mit

$$3jn + \frac{j^2}{2} + 6n + \frac{31}{2}j + 4 + \begin{cases} n^2 + 2n & \text{volle Matrix} \\ 16n & \text{dünnbesetzte Matrix} \end{cases}, \tag{5.4}$$

für die Iteration die in dem `else`-Zweig endet.

Wird die Abbruchtoleranz nicht unterschritten, wird Gleichung 5.1 k mal benötigt, damit gilt $\forall j : 1 \leq j \leq k$, um die Anzahl der minimal benötigten Fließkommaoperationen zu bestimmen. Da in jedem Schleifendurchlauf der `if`-Zweig genommen wird, gilt:

$$\begin{aligned} & \sum_{j=1}^{k} \left(3jn + 5n + 5j + 2 + \begin{cases} n^2 - n & \text{volle Matrix} \\ 27n & \text{dünnbesetzte Matrix} \end{cases} \right) \\ = \quad & (3n+5) \cdot \sum_{j=1}^{k} (j) + k \cdot \left(5n + 2 + \begin{cases} n^2 - n \\ 27n \end{cases} \right) \\ = \quad & (3n+5) \cdot \tfrac{k^2+k}{2} + 5kn + 2k + \begin{cases} kn^2 - kn \\ 27kn \end{cases} \\ = \quad & \tfrac{3nk^2+13nk+5k^2+9k}{2} + \begin{cases} kn^2 - kn & \text{volle Matrix} \\ 27kn & \text{dünnbesetzte Matrix} \end{cases}. \end{aligned}$$

Zusätzlich, mit $3n + 1$ Fließkommaoperationen vor dem Schleifenaufruf, hat eine vollständige GMRES(k) Ausführung insgesamt mindestens:

$$\frac{3nk^2 + 13nk + 5k^2 + 9k}{2} + 3n + 1 + \begin{cases} kn^2 - kn & \text{volle Matrix} \\ 27kn & \text{dünnbesetzte Matrix,} \end{cases} \tag{5.5}$$

Fließkommaoperationen.
Bei den Speicheroperationen, wird Gleichung 5.3 k mal benötigt, damit gilt $\forall j : 1 \leq j \leq k$. Daraus folgt:

$$\begin{aligned} & \sum_{j=1}^{k} \left(2jn + 6n + 12j + 4 + \begin{cases} n^2 + 2n & \text{volle Matrix} \\ 16n & \text{dünnbesetzte Matrix} \end{cases} \right) \\ = \quad & (2n+12)\tfrac{k^2+k}{2} + k \cdot \left(6n + 4 + \begin{cases} n^2 + 2n \\ 16n \end{cases} \right) \\ = \quad & nk^2 + 7nk + 6k^2 + 10k + \begin{cases} kn^2 + 2kn & \text{volle Matrix} \\ 16kn & \text{dünnbesetzte Matrix} \end{cases}, \end{aligned}$$

für die k Schleifenausführungen abschätzen.
Mit den $3n + 3$ Speicheroperation vor dem Schleifenaufruf, hat eine vollständige GMRES(k)-Ausführung insgesamt mindestens:

$$nk^2 + 7nk + 6k^2 + 10k + \begin{cases} kn^2 + 2kn & \text{volle Matrix} \\ 16kn + 3n + 3 & \text{dünnbesetzte Matrix} \end{cases}, \tag{5.6}$$

Speicheroperationen, beziehungsweise

$$8nk^2 + 56nk + 48k^2 + 80k + \begin{cases} 8kn^2 + 16kn & \text{volle Matrix} \\ 128kn + 24n + 24 & \text{dünnbesetzte Matrix} \end{cases} \tag{5.7}$$

Byte die gelesen oder geschrieben werden.

Wird die Abbruch Toleranz in der m-ten Iteration unterschritten, wird bei $m-1$ Schleifendurchläufen der `if`-Zweig genommen, also wird Gleichung 5.1 $m-1$ mal mit $\forall j : 1 \leq j < m$ benötigt und Gleichung 5.2 einmal mit $j = m$ benötigt, da der `else`-Zweig in der m-ten Schleifeniteration genommen wird. Daher gilt:

$$\begin{aligned}
& \textstyle\sum_{j=1}^{m-1} \left(3jn + 5n + 5j + 2 + \left\{ \begin{array}{l} n^2 - n \\ 27n \end{array} \right\}\right) + m^2 + 5mn + 4n + 12m + 2 + \left\{ \begin{array}{ll} n^2 - n & \text{volle M.} \\ 27n & \text{dünnbes. M.} \end{array} \right. \\
= \quad & \textstyle\sum_{j=1}^{m-1} \left(j(3n+5)\right) + (m-1) \cdot \left(5n + 2 + \left\{ \begin{array}{l} n^2 - n \\ 27n \end{array} \right\}\right) + m^2 + 5mn + 4n + 12m + 2 + \left\{ \begin{array}{l} n^2 - n \\ 27n \end{array} \right. \\
= \quad & (3n+5)\tfrac{m^2-m}{2} + 10mn - n + m^2 + 14m + \left\{ \begin{array}{l} mn^2 - mn \\ 27mn \end{array} \right. \\
= \quad & \tfrac{3nm^2+7m^2+17mn+23m}{2} - n + \left\{ \begin{array}{ll} mn^2 - mn & \text{volle M.} \\ 27mn & \text{dünnbesetzte M.} \end{array} \right. .
\end{aligned}$$

Zusätzlich mit $3n + 1$ Fließkommaoperationen vor dem Schleifenaufruf, hat so eine vollständige GMRES Ausführung insgesamt mindestens:

$$\frac{3nm^2 + 7m^2 + 17mn + 23m}{2} + 2n + 1 + \begin{cases} mn^2 - mn & \text{volle M.} \\ 27mn & \text{dünnbesetzte M.} \end{cases} \tag{5.8}$$

Fließkommaoperationen.
Auch um die Speicheroperationen abzuschätzen gilt, dass Gleichung 5.3 $m-1$ mal ausgeführt $\forall j : 1 \leq j < m$ wird, und Gleichung 5.4 ein mal mit $j = m$:

$$\begin{aligned}
& \textstyle\sum_{j=1}^{m-1} \left(2jn + 6n + 12j + 4 + \left\{ \begin{array}{l} n^2 + 2n \\ 16n \end{array} \right\}\right) + 3mn + \frac{m^2}{2} + 6n + \frac{31}{2}m + 4 + \left\{ \begin{array}{ll} n^2 + 2n & \text{volle M.} \\ 16n & \text{dünnbes. M.} \end{array} \right. \\
= \quad & (2n+12)\tfrac{m^2-m}{2} + (m-1)\left(6n + 4 + \left\{ \begin{array}{l} n^2 + 2n \\ 16n \end{array} \right\}\right) + 3mn + \tfrac{m^2}{2} + 6n + \tfrac{31}{2}m + 4 + \left\{ \begin{array}{l} n^2 + 2n \\ 16n \end{array} \right. \\
= \quad & nm^2 + 8nm + \tfrac{13m^2}{2} + \tfrac{27}{2}m + \left\{ \begin{array}{ll} mn^2 + 2mn & \text{volle M.} \\ 16nm & \text{dünnbesetzte M.} \end{array} \right.
\end{aligned}$$

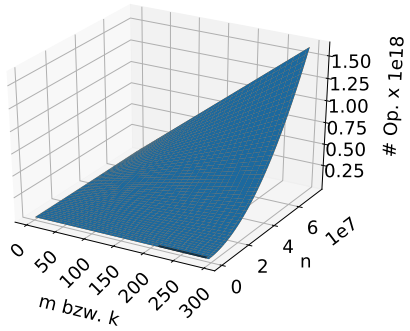
Mit den $3n+3$ Speicheroperation vor dem Schleifenaufruf, hat eine vollständige GMRES Ausführung insgesamt mindestens:

$$nm^2 + 8nm + \frac{13m^2}{2} + \frac{27}{2}m + 3n + 3 + \begin{cases} mn^2 + 2mn & \text{volle M.} \\ 16nm & \text{dünnbesetzte M.} \end{cases} \tag{5.9}$$

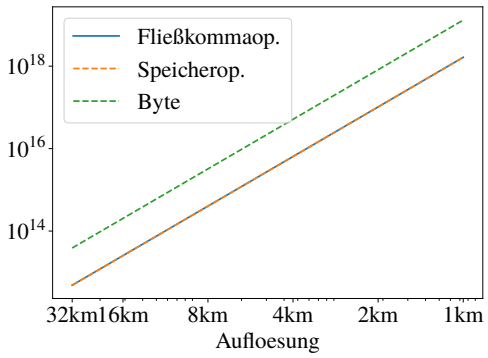
Speicheroperationen, beziehungsweise

$$8nm^2 + 64nm + 52m^2 + 108m + 24n + 24 + \begin{cases} 8mn^2 + 16mn & \text{volle M.} \\ 128nm & \text{dünnbesetzte M.} \end{cases} \tag{5.10}$$

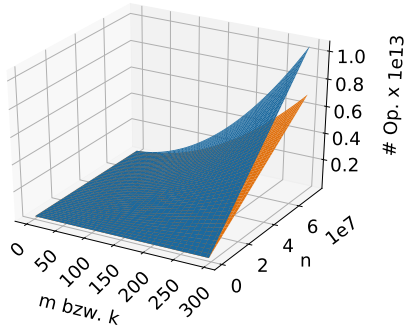
Byte die gelesen oder geschrieben werden.



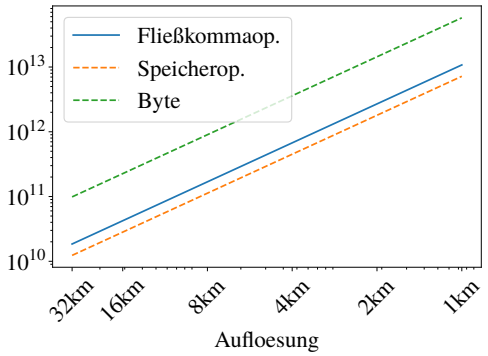
(a) Anzahl an Fließkommaoperationen (blau dargestellt) und verdeckt die Speicheroperationen (orange dargestellt) beim Lösen des Gleichungssystems mit einer voll besetzten Matrix (Gleichung 5.8, 5.9 volle M.).



(b) Anzahl an Speicheroperationen, Anzahl an Fließkommaoperationen und damit verbundene Datenmenge in Byte in GMRES bei einer Unterschreitung der Toleranz nach 300 Iterationen und einer voll besetzten Matrix.



(c) Anzahl an Fließkommaoperationen (blau dargestellt) und Speicheroperationen (orange dargestellt) beim Lösen des Gleichungssystems mit 14 Matrixeinträge in jeder Zeile (Gleichung 5.8, 5.9 dünnbesetzte M.).



(d) Anzahl an Speicheroperationen, Anzahl an Fließkommaoperationen und damit verbundene Datenmenge in Byte in GMRES(k) bei einer Unterschreitung der Toleranz nach 300 Iterationen und 14 Matrixeinträge in jeder Zeile des Gleichungssystems auf doppelt logarithmischen Achsen.

Abbildung 5.3: Vergleich der Anzahl an Speicheroperationen (blau dargestellt), und der Anzahl an Fließkommaoperationen (orange dargestellt), in Bezug auf der Anzahl an GMRES Iterationen und Datenpunkten n (vergleiche Tabelle 3.1). Dabei zu berücksichtigen ist, dass größere Matrizen und Matrizen mit mehr Einträgen eine größere Anzahl an Iterationen bis zur Konvergenz benötigen (vergleiche Abbildung 5.8b).

5.3.3.2 Interpretation

Betrachtet man die Bytes, die gelesen oder geschrieben werden müssen (Gleichung 5.7 und Gleichung 5.10), ist festzustellen, dass das grundlegende Verhalten gleich sein muss. Die beiden Gleichungen, die die Anzahl an Speicheroperationen angeben (5.7 und 5.10), sind quadratisch

in m bzw. k und bei einer voll besetzten Matrix auch quadratisch in n. Wenn die Matrixstruktur berücksichtigt wird, also nur die nötigen Einträge der Matrix, die in die Berechnung mit einfließen, sind beide quadratisch in m bzw. k und linear in n.

Für die Gleichungen, die die Anzahl an Fließkommaoperationen angeben (Gleichung 5.8 und Gleichung 5.5), gilt dies ebenso. Sie sind quadratisch in m bzw. k und n und quadratisch in m bzw. k und linear in n, für den Fall, dass die Matrixstruktur berücksichtigt wird.

Dies bedeutet, dass über die Komplexität nicht abgeleitet werden kann, ob die Performance im Allgemeinen durch die Speichergeschwindigkeit oder die CPU-Performance limitiert wird. Dazu müssen die Gleichungen genauer verglichen werden. In der Visualisierung von den Gleichungen 5.5 bis 5.10 in Abbildung 5.3 lässt sich allerdings ablesen, dass, sowohl wenn die Matrixstruktur berücksichtigt wird, als auch bei einer vollen Matrix, mehr Byte gelesen bzw. geschrieben werden müssen, als Fließkommaoperationen durchgeführt werden müssen. Bei einer Rechnung mit der voll besetzten Matrix, sind die Anzahl an Fließkommaoperationen und Speicheroperationen quasi identisch, wie in Abbildung 5.3a und Abbildung 5.3b zu sehen ist. Bei der Betrachtung von Abbildung 5.3c könnte man zum Schluss kommen, dass bei größerem m bzw. k der Algorithmus nicht mehr speicherbandbreitenlimitiert ist, sondern durch die Leistung der CPU limitiert wird, da die Anzahl an Fließkommaoperationen mit steigendem m bzw. k schneller ansteigt, als die Anzahl an Speicheroperationen. Bei $n = 73957122$, was einer Auflösung von einem Kilometer entspricht, und einem k von 300 beträgt der Unterschied immerhin circa $0{,}4 \times 10^{13}$ Operationen. Da eine Fließkommaoperation allerdings 8 B entspricht, müssen, wie in Abbildung 5.3c zu sehen ist, bei einem k von 300 immer noch deutlich mehr Byte geladen oder gespeichert werden, als Fließkommaoperationen durchgeführt werden.

Für eine genauere Einschätzung werden die Anzahl an Fließkommaoperationen durch die Anzahl an Byte geteilt. Für den:

GMRES / frühzeitiger Abbruch bedeutet dies, dass Gleichung 5.5 durch Gleichung 5.7 geteilt wird. Also für die dünnbesetzte Matrix:

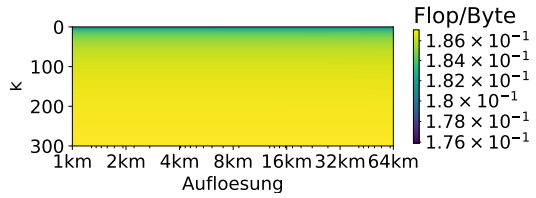
$$\begin{aligned} &\frac{\frac{3nm^2+7m^2+17mn+23m}{2} + 2n + 1 + 27mn}{8nm^2 + 64nm + 52m^2 + 108m + 24n + 24 + 128nm} \\ &= \frac{3nm^2 + 7m^2 + 71mn + 23m + 4n + 2}{16nm^2 + 104m^2 + 384mn + 216m + 48n + 48} \end{aligned} \tag{5.11}$$

und für die komplette Matrix :

$$\begin{aligned} &\frac{\frac{3nm^2+7m^2+17mn+23m}{2} + 2n + 1 + mn^2 - mn}{8nm^2 + 64nm + 52m^2 + 108m + 24n + 24 + 8mn^2 + 16mn} \\ &= \frac{3nm^2 + 7m^2 + 15mn + 23m + 4n + 2 + 2mn^2}{16nm^2 + 160nm + 104m^2 + 216m + 48n + 48 + 16mn^2} \end{aligned} \tag{5.12}$$

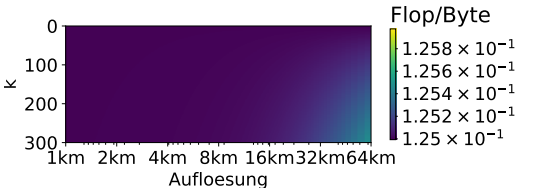
GMRES (k)/ ohne frühzeitigen Abbruch bedeutet dies, dass Gleichung 5.8 durch Gleichung 5.10 geteilt wird. Also für die dünnbesetzte Matrix:

$$\frac{\frac{3nk^2+13nk+5k^2+9k}{2} + 3n + 1 + 27kn}{8nk^2 + 56nk + 48k^2 + 80k + 128kn + 24n + 24} \frac{\text{Flop}}{\text{Byte}}$$

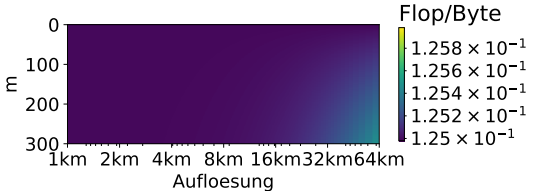


(a) Dünnbesetzte Matrix GMRES(k) ohne Abbruch. Visualisierung von Gleichung 5.13

(b) Dünnbesetzte Matrix GMRES mit Abbruch in der m. Iteration. Visualisierung von Gleichung 5.11



(c) Voll besetzte Matrix GMRES(k) ohne Abbruch. Visualisierung von Gleichung 5.14



(d) Voll besetzte Matrix GMRES mit Abbruch in der m. Iteration. Visualisierung von Gleichung 5.12

Abbildung 5.4: Vergleich der Fließkommaberechnungen pro benötigtem Byte von GMRES(k) und GMRES, von einer voll besetzten Matrix, sowie einer Matrix mit vierzehn Einträgen pro Zeile.

$$= \frac{3nk^2 + 67nk + 5k^2 + 9k + 6n + 2}{16nk^2 + 368nk + 96k^2 + 160k + 48n + 48} \frac{\text{Flop}}{\text{Byte}} \tag{5.13}$$

und für die komplette Matrix :

$$\frac{\frac{3nk^2+13nk+5k^2+9k}{2} + 3n + 1 + kn^2 - kn}{8nk^2 + 56nk + 48k^2 + 80k + 8kn^2 + 16kn} \frac{\text{Flop}}{\text{Byte}} =$$

$$\frac{3nk^2 + 11nk + 5k^2 + 9k + 6n + 2 + 2kn^2}{16nk^2 + 144nk + 96k^2 + 160k + 16kn^2} \frac{\text{Flop}}{\text{Byte}} \tag{5.14}$$

In der Visualisierung dieser vier Gleichungen (5.11, 5.12, 5.13, 5.14) in Abbildung 5.4, sieht man, dass die erreichbare Peakperformance nahezu unabhängig von der Auflösung ist. Außerdem liegt dieser Quotient, in dem interessanten Gebiet relativ konstant bei 0,125 für eine voll besetzte Matrix und bei 0,18, wenn die Matrixstruktur berücksichtigt wird. Damit sind sie weit entfernt von dem Maschinengleichgewicht der Hardware (sechs bis achtzehn, siehe Tabelle 3.4), auf welcher die Analyse durchgeführt wurde. Demnach sollte der GMRES(k)-Algorithmus circa $\frac{0,125}{10} = 0,69\,\%$ bis $\frac{0,125}{6} = 2,08\,\%$ der Peakperformance erreichen, wenn der Algorithmus mit einer vollen Matrix rechnet, und $\frac{0,18}{18} = 1\,\%$ bis $\frac{0,18}{6} = 3\,\%$, wenn die Matrixstruktur berücksichtigt wird. Tabelle 5.1 zeigt die prozentual erreichbare Performance aufgeschlüsselt auf die unterschiedliche Hardware.

Die Berechnung, setzt zwar volle SIMD-Vektorisierbarkeit voraus, was nicht für den Kompletten in 5.1 vorgestellten Algorithmus zutrifft. Allerdings bestehen die rechenintensiven Bereiche aus Matrix-Vektor- und Vektor-Vektor-Operationen, welche gut vektorisierbar sind. Die Berechnung geht aber auch davon aus, dass alle Daten immer aus dem Hauptspeicher geladen werden müssen und nichts gechached wird. Da aber nur die Matrix und der Vektor der zu lösenden Gleichung

CPU	Intel Xeon E3-1585 v5		Intel Xeon Gold 6230		Intel Xeon E5-2667 v3	
Speicherbandbreite	34 GiB/s		131,13 GiB/s		68 GiB/s	
Maschinengleichgewicht	6 F/B		10 F/B		6 F/B	
turbo	7 F/B		18 F/B		6 F/B	
ein Kern	1,6 F/B		0,5 F/B		0,7 F/B	
ein Kern turbo	1,7 F/B		0,9 F/B		0,8 F/B	
Matrix	dünn	voll	dünn	voll	dünn	voll
möglich	3 %	2,1 %	1,8 %	1,3 %	3,0 %	2,1 %
turbo	2,6 %	1,8 %	1,0 %	0,7 %	3,0 %	2,1 %
ein Kern	11,3 %	7,4 %	36,0 %	25,0 %	25,7 %	17,9 %
ein Kern turbo	10,6 %	7,8 %	20,0 %	13,9 %	22,5 %	15,6 %

Tabelle 5.1: Wiederholung der Hardwareeigenschaften und die daraus resultierende von GMRES(k) erreichbare Peakperformance.

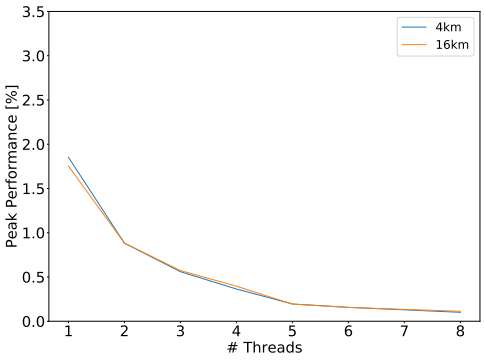
wirkliche Eingangsdaten sind und diese zwingend gelesen werden müssen, kann ein Teil davon gecached werden. Demnach sollte die prozentuale Auslastung gut erreichbar sein und kann als grobe untere Grenze für die maximal mögliche Performance angenommen werden.

Wie in Abbildung 5.5 zu sehen ist, wird die niedrige mögliche Performance nicht erreicht. Das Maschinengleichgewicht aus Tabelle 3.4 ist für die Auslastung aller Kerne der jeweiligen CPU berechnet. Um eine bessere Einschätzung zu haben, was auf einem Kern an Leistung möglich ist, wurde in Tabelle 5.1 aus diesem Grund zusätzlich dazu das Gleichgewicht für nur einen Kern und der darauf maximale erreichbare Performanceanteil berechnet. Da der Algorithmus stark Hauptspeicherbandbreiten limitiert ist, ist der erreichbare Performanceanteil deutlich größer, wenn der einzelne Kern die limitierende Bandbreite nicht teilen muss. Dies ist im Vergleich der beiden Plots in Abbildung 5.5a und Abbildung 5.5c deutlich zu sehen. Auf dem Intel Xeon E5-2667 v3 des PIK-Clusters wird mit einem Thread auf einem Kern ungefähr doppelt so viel prozentuale Performance erreicht wie auf dem Intel Xeon E3-1585 v5 von sgscl1, was die Speicherbandbreitenlimitierung bestärkt und auch darauf zurückzuführen ist. Mit 68 GiB/s ist diese genau doppelt so groß, wie die 32 GiB/s auf einem sgscl1-Knoten.

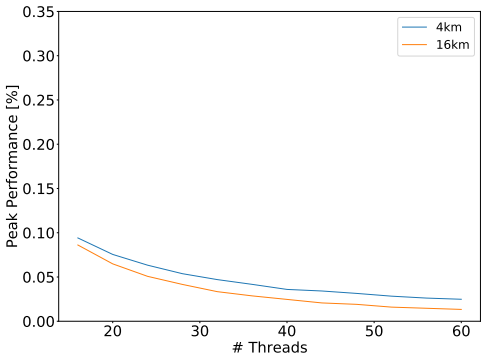
Ausgehend von den bisherigen Erkenntnissen, sollte also mit einer effizienteren Implementierung auf einem Kern circa sechs bis sieben Mal mehr Performance erreicht werden können, also die Laufzeit für das Lösen des Gleichungssystems auf $\frac{1}{6}$ bis $\frac{1}{7}$ der aktuellen reduziert werden.

Daraus, dass der Quotient bei der Auslastung von 8 Threads auf einem PIK-Konten bei $\frac{1}{10}$ ist, lässt sich schlussfolgern, dass mit einer effizienteren Skalierung auf einem Knoten hier schon eine Laufzeitverbesserung erreichbar sein sollte.

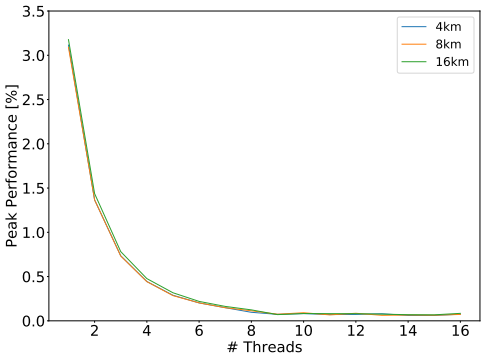
Abbildung 5.5b und Abbildung 5.5d zeigen, dass auch auf mehreren Knoten, wenn also die Hauptspeicherbandbreite erhöht wird, die prozentual errichte Peakperformance immer weiter abnimmt. Werden Abbildung 5.5a und Abbildung 5.5b zusammen genommen und wie in Abbildung 5.5e doppelt logarithmisch skaliert, ist festzustellen, dass beim Übergang von einem Knoten auf mehrere Knoten keine Unterschiede in dem Verhalten erkennbar ist. Dies deutet auf weitere Faktoren, die die Skalierung beeinträchtigen, wie beispielsweise eine Limitierung bedingt durch die Netzwerkbandbreite.



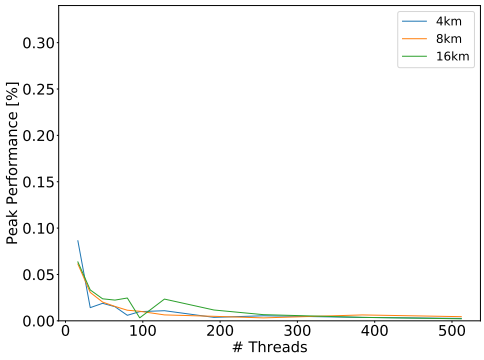
(a) Prozentuale erreichte Peakperformance von KSPSolve auf einem Knoten von sgscl1.



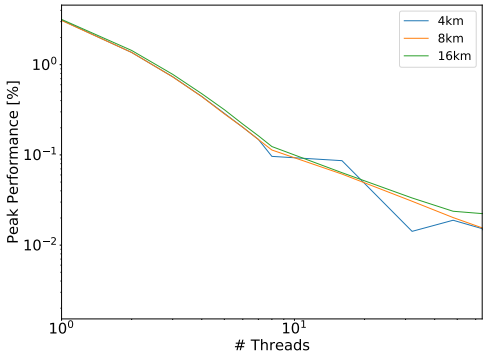
(b) Prozentuale erreichte Peakperformance von KSPSolve auf mehreren Knoten von sgscl1.



(c) Prozentuale erreichte Peakperformance von KSPSolve auf einem Knoten des PIK-Clusters.

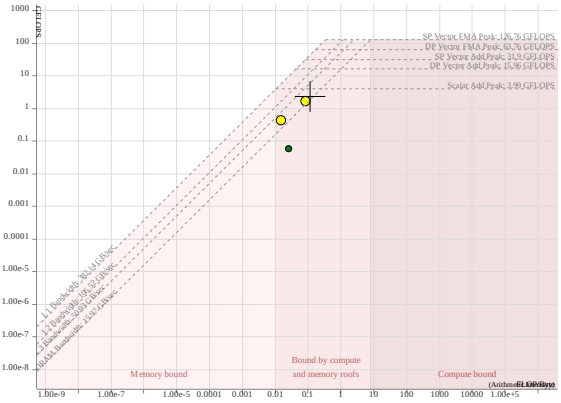


(d) Prozentuale erreichte Peakperformance von KSPSolve auf mehreren Knoten des PIK-Clusters.

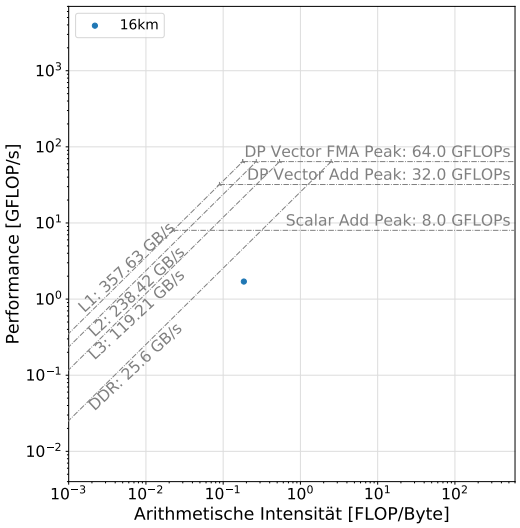


(e) Prozentuale erreichte Peakperformance von KSPSolve auf mehreren Knoten des PIK-Clusters.

Abbildung 5.5: Vergleich der prozentual erreichten Peakperformance von GMRES(k) auf dem sgscl1- sowie PIK-Cluster.



(a) Automatisch generiertes Modell des Intel Advisors.



(b) Modell auf Basis der Theoretischen Vorüberlegungen.

Abbildung 5.6: Roofline Modell Vergleich, auf Basis einer Simulation von 16 km Auflösung über 10 Jahre auf einem Haswell Intel i5-4670k Kern.

5.3.3.3 Roofline Modell

Abbildung 5.6a zeigt eine Selektion der Funktionen, die an KSPSolve involviert sind, aus Abbildung 4.1, also ein vom Intel Advisor automatisch generiertes Roofline-Modell aus einer Simulation mit 16 km über 10 Jahre. Auch hier gilt, dass der Intel Advisor aus unbekannter Ursache die skalaren Fließkommaoperationen, die vektorisierte Operationen ohne FMA sowie die Bandbreiten deutlich schlechter einschätzt als sie sein sollten und ebenso mit anderen Tests validiert wurden (siehe Abschnitt 4.1). Im Vergleich dazu, zeigt Abbildung 5.6b ein Modell, welches sowohl auf der arithmetischen Intensität, berechnet nach Gleichung 5.12, als auch der gemessenen $1{,}7 \times 10^9$ Fließkommaoperationen pro Sekunde sowie den erreichbaren Maximalwerten aus Tabelle 3.4 beruht. Beim Vergleich von Abbildung 4.1 und Abbildung 5.6b ist zu sehen, dass die aus dem Programmcode abgeleitete arithmetische Intensität deutlich niedriger ist, als die theoretisch ermittelte. Also werden in der aktuellen PETSc-Implementierung mehr Daten geladen beziehungsweise geschrieben, als nach der Analyse des GMRES-Algorithmus in Unterunterabschnitt 5.3.3.1 nötig sind, da die Anzahl an Fließkommaoperationen bei gleichem Algorithmus nicht kleiner sein sollte als die in Unterunterabschnitt 5.3.3.1 ermittelte „optimale“ Anzahl. Die gemessene Anzahl an Fließkommaoperationen pro Sekunde ist logischerweise bei beiden Fällen gleich.

Eine weitere Eigenschaft der automatisch generierten Roofline Modelle ist, dass nicht abgelesen werden kann, wie viel langsamer der Code ist, als er optimalerweise sein sollte. Programme laufen immer so schnell wie möglich, sind also zwingend entweder bandbreitenlimitiert oder CPU-limitiert. Damit zeigen automatisch generierten Modelle, wie Abbildung 4.1, für Programme die keine Eingabe oder Ausgabe haben, immer Messungen nahe der Limitierungen. Damit kann nicht abgelesen werden, um wie viel der Algorithmus schneller, unabhängig der Implementierung,

laufen müsste, hierfür wird ein theoretisch erstelltes Modell, wie Abbildung 5.6b benötigt. Der Abstand der Punkte zu dem als Erstes limitierenden Limit, zeigt das mögliche Performancepotential an.

Für die Codeoptimierung hingegen, sind automatisch generierte Roofline-Modelle sehr hilfreich, da direkt ersichtlich ist, worin aktuell der limitierende Faktor liegt. Aus dem Vergleich der beiden Modelle ist dann ersichtlich, wie das Programm angepasst werden sollte, um die theoretisch optimale Performance zu erreichen. Aus dem Vergleich in Abbildung 5.6, lässt sich beispielsweise ablesen, dass pro Fließkommaoperation nur ein Zehntel der Daten gelesen beziehungsweise geschrieben werden sollte, um eine optimale Performance zu erreichen.

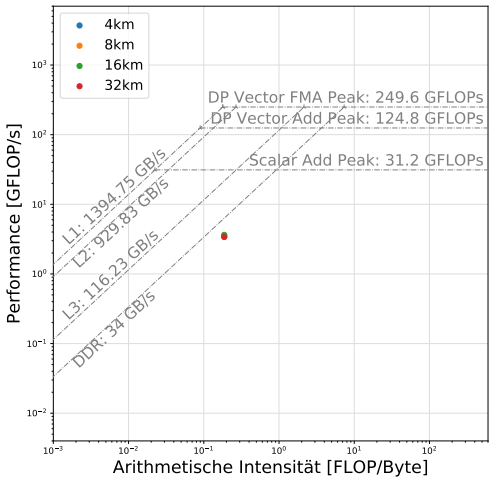
Abbildung 5.7 zeigt Roofline Modelle für die unterschiedliche zur Verfügung stehenden Hardware. Es ist zu sehen, dass es auf jeder Hardware deutliches Verbesserungspotential gibt. Auch ist festzustellen, dass die Voraussetzung der Vektorisierbarkeit für die vorangegangene Berechnung der möglichen Performancesteigerung keine Einschränkung ist. Wie in den Modellen zu sehen ist, benötigt der GMRES Algorithmus frühestens die Performance der Vektorisierung, wenn er nicht mehr Hauptspeicher-, sondern L3 Bandbreitenlimitiert ist. Damit gelten die zuvor aufgestellten unteren Schranken für das Performancepotential.

5.3.3.4 Kommunikation

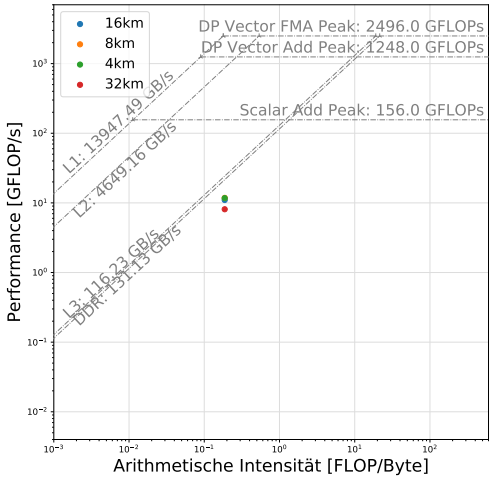
Nachdem, wie in Abbildung 5.5b und Abbildung 5.5d zu sehen ist, die CPU-Auslastung im parallelen Einsatz nahe 0 % ist, deutet dies darauf hin, dass auch die Kommunikation zwischen den einzelnen Knoten ein Flaschenhals sein kann. Hierauf verweist auch PETSc selber :„PETSc can be used with any kind of parallel system that supports MPI BUT for any decent performance one needs a fast, low-latency interconnect; any ethernet, even 10 Gbit ethernet simply cannot provide the needed performance.“ [BAA+], auf deutsch: „PETSc kann mit jeder Art von Parallelsystem verwendet werden, das MPI unterstützt, aber für jede annehmbare Leistung benötigt man eine schnelle Verbindung mit niedriger Latenz; jedes Ethernet, selbst 10 Gbit Ethernet, kann die benötigte Leistung einfach nicht erbringen.“ Dies würde bedeuten, dass sgscl1 bedingt durch sein 10 Gbit Ethernet nicht adäquat für PETSc geeignet ist, und durch das Infinity Band von dem BwUniCluster und dem PIk-Cluster die Parallelisierung deutlich besser erfolgen sollte. Wie im Vergleich von Abbildung 5.5b und Abbildung 5.5d zu sehen ist, kann PETSc nicht ausreichend von dieser theoretischen Verbesserung profitieren, um ansatzweise gute Hardwareauslastung zu erreichen.

Aus der große Anzahl an Nachrichten in Abbildung 5.8a lässt sich schlussfolgern, dass die GMRES(k)-Implementierung von PETSc für die Matrixstruktur von PISM eindeutig netzwerkbandbreitenlimitiert ist. Wie in Abbildung 5.2a zu sehen ist, benötigt der picard-Manager für das Lösen auf 60 Threads circa 50 ms. Wenn man dies in Relation zu 300 Nachrichten (Abbildung 5.8a) setzt, bedeutet dies, dass im Durchschnitt alle 0,17 ms eine Kommunikation stattfindet.

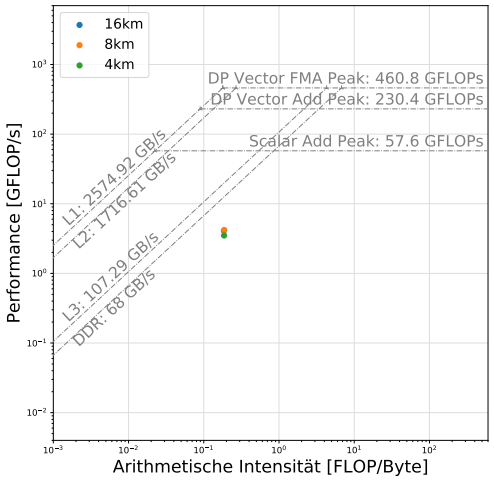
Die parallele GMRES(k)-Implementierung von PETSc basiert auf dem Parallelisierungsansatz von Ghysels et al. [GAMV13]. In diesem Ansatz wird allerdings davon ausgegangen, dass eine all-to-all Kommunikation des Ergebnisvektors hinter der dünnbesetzten Matrix-Vektor-Multiplikation versteckbar ist. Für die PISM-Matrixstruktur bedeutet dies, dass die all-to-all Kommunikation eines Double-Wertes von ungefähr neun FMA-Operationen versteckt werden soll. Dass dies nicht



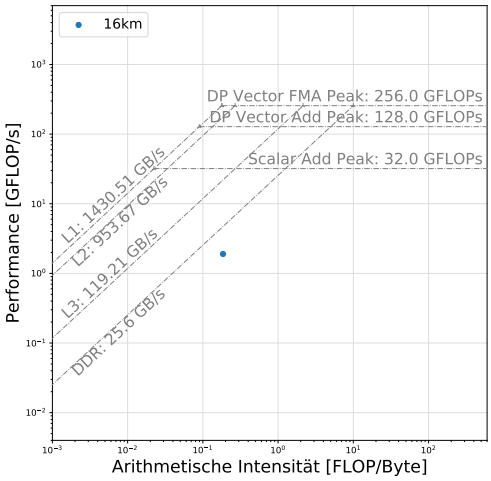
(a) Skylake Intel Xeon E3-1585 v5 (sgscl1)



(b) Cascadelake Intel Xeon Gold 6230 (bwUniCluster)

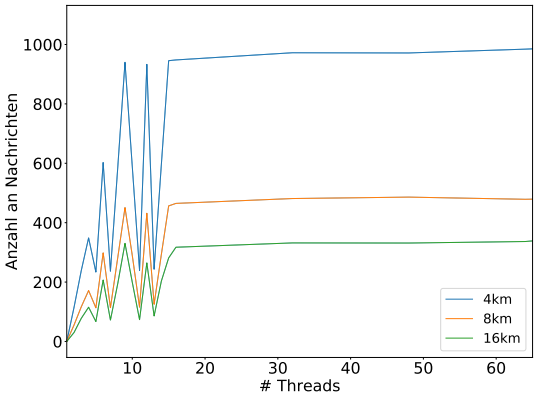


(c) Haswell Intel Xeon E5-2667 v3 (PIK Cluster)

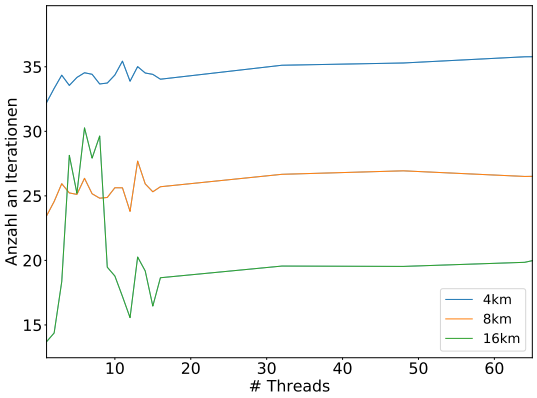


(d) Haswell Intel i5-4670k

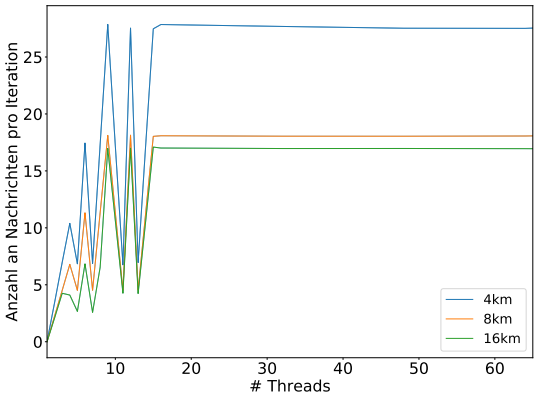
Abbildung 5.7: Roofline Modelle auf Basis der theoretischen Vorüberlegungen für GMRES(k) auf unterschiedlicher Hardware.



(a) Durchschnittliche Anzahl der MPI-Nachrichten pro GMRES(k) Aufruf.



(b) Anzahl der Iterationen, die GMRES(k) zur Konvergenz benötigt.



(c) Anzahl der Kommunikationen innerhalb einer GMRES(k) Iteration.

Abbildung 5.8: Kommunikations- und Iterationsverhalten von GMRES(k) in Abhängigkeit der Threadanzahl.

realistisch ist, fällt spätestens auf, wenn das Maschinengleichgewicht aus Tabelle 3.4 betrachtet wird. Mit 18 Fließkommaoperationen pro vier Byte also $4\frac{\text{Flop}}{\text{Byte}}$, kann dies nicht einmal innerhalb eines Knotens erreicht werden.

5.3.3.5 Partitionierung

In Abbildung 5.8a ist nicht nur zu sehen, dass sehr viele Nachrichten kommuniziert werden, sondern auch, dass bei gewissen Threadanzahlen deutlich weniger kommuniziert wird, als bei anderen. Dies ist darauf zurückzuführen, wie in PISM die Gebietspartitionierung durchgeführt wird, und wie sich diese auf das resultierende Gleichungssystem auswirkt.

Die `stress_balance` Gleichungen werden in PISM in einem zweidimensionalen Gebiet aufgestellt. Die Finiten-Differenzen-Implementierung für die das Gleichungssystem aufgestellt und gelöst wird, arbeitet mit Differenzen-Sternen. Jeder Datenpunkt hat dabei zwei Freiheitsgrade und ist von seinen acht Nachbarn abhängig.

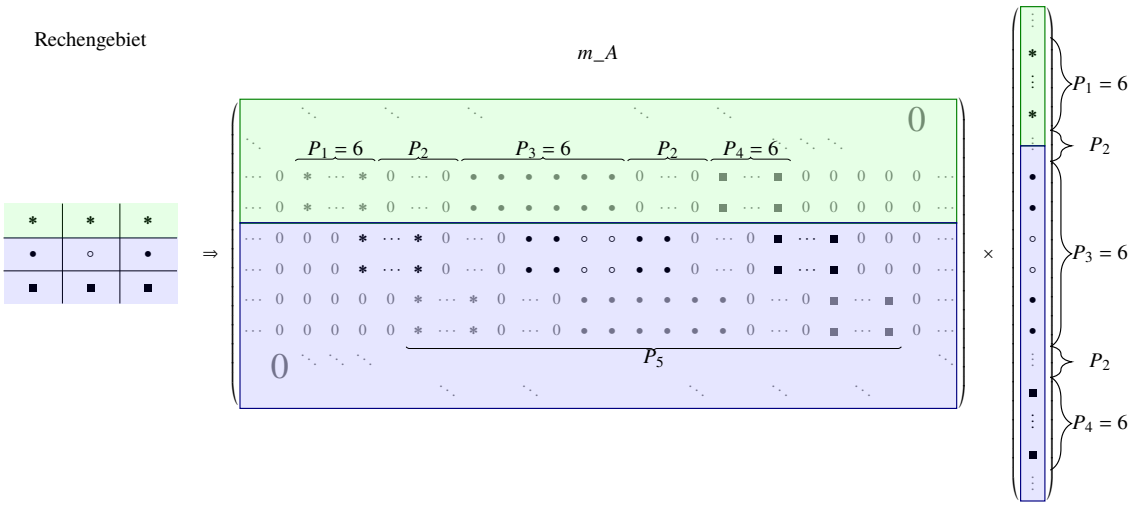


Abbildung 5.9: Visualisierung der Partitionierung der Matrix-Vektor-Multiplikation des GMRES(k)-Algorithmus.

In dem Gleichungssystem welches aufgestellt wird, spiegeln sich diese Abhängigkeiten wieder. Immer zwei Zeilen bzw. zwei Spalten stellen die zwei Freiheitsgrade eines Datenpunktes dar. Die Datenpunkte werden dabei zeilenweise linearisiert. Bestehen hierbei Abhängigkeiten zueinander, ist der Matrixeintrag am Schnittpunkt ungleich Null. Dies führt dazu, dass in Zeile 7 im vorgestellten GMRES(k)-Algorithmus in Algorithmus 5.1 eine Matrix-Vektor-Multiplikation, wie in Abbildung 5.9 dargestellt, durchgeführt wird.

Im nicht parallelisierten Fall werden die Abhängigkeiten der n Datenpunkte in einer Matrix mit $2n$ Zeilen und Spalten dargestellt. Diese Matrix beinhaltet eine über sechs Zellen breite Diagonale, sowie zwei sechs Zellen breite Bänder, die den Abstand P_2 (zweimal Rechengebietsbreite minus vier) zu dem mittleren Band haben. Von diesen achtzehn möglichen Einträgen sind bis zu vierzehn ungleich Null. Außerdem ist jede zweite Zeile um eins nach links versetzt.

Wird nun das Rechengebiet auf zwei Threads parallelisiert, geschieht dies in Zeilenblöcken. Die ersten $\lfloor n/2 \rfloor * 2$ Zeilen liegen im Speicher des einem Threads und die anderen im Speicher des zweiten Threads. Auch der Vektor und der Ergebnisvektor werden auf diese Weise aufgeteilt. Das bedeutet, dass, um die Multiplikation durchführen zu können, Teile des q Vektors kommuniziert werden müssen. Genauer gesagt, ein Thread muss jeweils P_2 plus sechs viele Daten an die Nachbarthreads kommunizieren.

Unter der Annahme, dass ein Thread mindestens eine Zeile hat, würde dies auch für die weitere Parallelisierung gelten, solange das Eingangsgebiet nur zeilenweise aufgeteilt wird. Dies ist jedoch leider nicht der Fall. Das Rechengebiet wird in einem Schachbrettmuster unterteilt. Eine vertikale Unterteilung des Rechengebietes führt zu dem in Abbildung 5.10 dargestellten Effekt.

Bei gleichbleibender Matrix sind die Vektorabschnitte auf den einzelnen Threads nicht mehr zusammenhängend. Die Matrix und die Vektoren sind allerdings nach PETSc-Definition zwingend zeilenweise zusammenhängend aufgeteilt. Dies bedeutet, dass eine komplette Umstrukturierung des Gleichungssystems vorgenommen wird. Die Matrix bleibt dabei zeilenweise aufgeteilt, für die Multiplikation muss nun allerdings nicht mehr mit den beiden Nachbarn kommuniziert werden, sondern

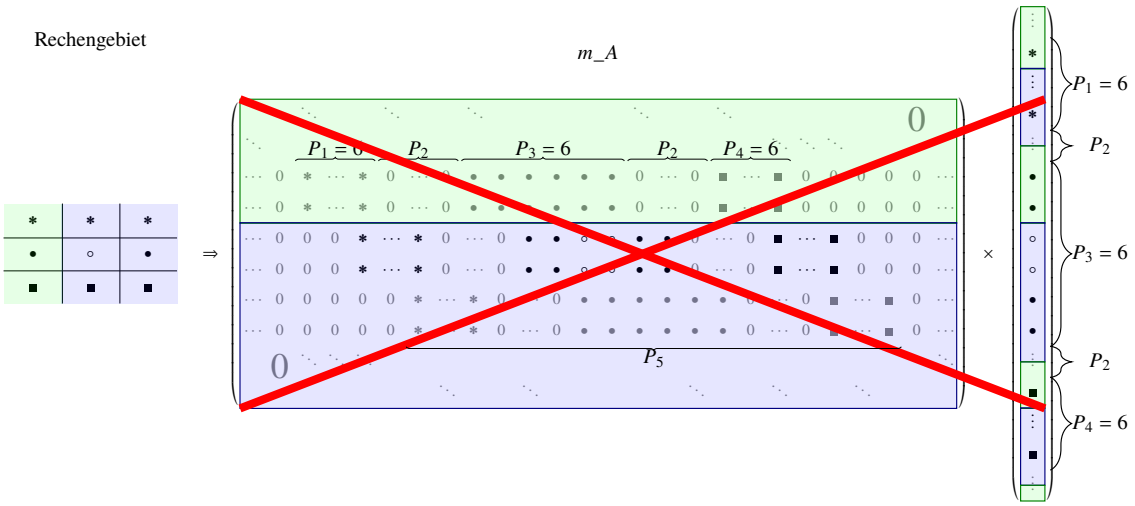


Abbildung 5.10: Visualisierung der Parittionierungsproblematik.

mit allen bis zu acht angrenzenden Nachbarn. Außerdem geht bei einer Rechengebietszerlegung durch vertikale Streifen die gleichmäßige Bandstruktur verloren. Während bei der horizontalen Aufteilung die Matrixstruktur nicht gespeichert werden müsste, ist dies hier notwendig.

Ist die Anzahl an Threads eine Primzahl, teilt PETSc das Gebiet zeilenweise auf. Dies erklärt das Muster in Abbildung 5.8a, da hier die Kommunikation um ein Achtel geringer ist.

6 Verbesserungsvorschläge

Nachdem nun bekannt ist, an welchen Stellen Performanceprobleme von PISM existieren, geht es in diesem Kapitel um Ideen und Ansätze, mit denen diese Probleme angegangen werden könnten. Dazu wird als Erstes Probleme der Bandbreitenlimitierung, sowohl in Bezug auf die Netzwerkbandbreite, als auch in Bezug auf die Haupspeicherbandbreite, thematisiert. Abgeschlossen wird das Kapitel von Verbesserungsvorschlägen allgemeiner Natur.

6.1 Bandbreitenlimitierung

Wie bereits festgestellt, ist die zur Verfügung stehende Bandbreite der limitierende Faktor für die Laufzeitverbesserung. Daher sollte für eine langfristige Performancesteigerung hier angesetzt werden. Unter einer effizienten Auslastung der zur Verfügung stehenden Bandbreite profitieren auch die weiteren Optimierungsansätze. Für die effiziente Nutzung der Hauptspeicherbandbreite werden im Allgemeinen jedoch andere Techniken genutzt, wie zur effizienten Netzwerknutzung.

6.1.1 Hauptspeicherbandbreite

Damit eine optimale Ressourcenauslastung für ein bestimmtes Problem erreicht werden kann, ist es im Allgemeinen nicht möglich auf Standartimplementierungen zurückzugreifen. Je genauer ein Problem eingegrenzt werden kann, desto spezifischer kann ein Code speziell für dieses Problem geschrieben werden. Eine allgemeine dünnbesetzte Matrix-Vektor-Multiplikation muss beispielsweise für jeden Matrixeintrag die Indizes bestimmen und kann dann erst damit rechnen. Für die vorgestellte Matrix-Vektor-Struktur ist, bei einer Parallelisierung des Ausgangsgebietes in Zeilen, die Matrixstruktur fix. Ist es dann möglich die Daten diagonalweise zu speichern, kann die Matrix-Vektor-Multiplikation über vierzehn Vektor-Vektor-Multiplikationen, mit gleichzeitigem Aufaddieren auf einen Ergebnisvektor, realisiert werden. Hierbei müssen keinerlei Indizes abgespeichert, berechnet, oder berücksichtigt werden. Außerdem hat dieser Ansatz den Vorteil, dass alle Daten linear aus dem Speicher gelesen werden können. Das bedeutet, dass man die Hauptspeicherbandbreite optimal ausnutzen kann. Wird hierbei ausgenutzt, dass alle vierzehn Vektoren mit dem gleichen Vektor multipliziert werden müssen und dieser durch Strategien, wie zum Beispiel dem Blocking, effektiv in den Cache-Hierarchien gehalten wird, sollten deutliche Performancesteigerungen möglich sein. Für die vierzehn unterschiedlichen Vektoren lassen sich zusätzlich sogenannte „non-temporal“ Befehle nutzen, mit denen nun die Daten am Cache vorbei geladen werden können.

Eine weitere Möglichkeit die Speicherbandbreiten besser auszunutzen besteht darin, die Daten, die von mehreren Kernen benötigt werden, nicht redundant im Cache zu halten, sondern nur einmal. Hierfür müsste allerdings die Parallelisierung von einer reinen MPI-Parallelisierung auf einen Parallelisierungsansatz erweitert werden, bei welchem effiziente und explizite lokale

Ressourcenteilung möglich ist. Eine hybride MPI-OpenMP-Parallelisierung würde zum Beispiel ermöglichen, dass bei der eben genannten mehrfachen Multiplikation mit dem gleichen Vektor, der Vektor nur einmal im L3 Cache gehalten werden muss. Dadurch ist es möglich einen deutlich größeren Anteil des Vektors zu behalten, weil dieser nicht redundant gespeichert werden muss. Gleichzeitig muss dieser auch nur einmal geladen werden und nicht redundant für jeden Kern einzeln.

6.1.2 Netzwerkbandbreite

Ein hybrider MPI-OpenMP-Ansatz hätte zudem den Vorteil, dass er die Netzwerkbandbreite schonen könnte. Aktuell ist PISM so implementiert, dass für viele Datenstrukturen, an den Kommunikationskanten zu den Nachbar-Threads, sogenannte Halo-Zellen genutzt werden. Fallen die knoteninternen Halo-Zellen weg, bedeutet dies nicht nur, dass deutlich weniger Speicherplatz benötigt wird, weil die internen Halo-Zellen komplett wegfallen, sondern auch, dass die Kommunikationsanzahl sowie die Hauptspeicherbelastung sinkt, weil intern die Halo-Zellen nicht mehr geupdated werden müssen. Jeder Knoten benötigt zwar noch die gleichen Daten der anderen Knoten, allerdings werden dafür nicht mehr viele kleine Kommunikationen, sondern pro Außenkante nur noch eine große Kommunikation benötigt. Wenige große Kommunikationen haben vielen kleinen Kommunikationen gegenüber den Vorteil, dass sie weniger Overhead haben und somit das Netzwerk entlasten.

Eine weitere Möglichkeit, um Last aus dem Netzwerk zu nehmen, wäre, die Zwischenergebnisse zunächst lokal auf den einzelnen Knoten zu speichern und schließlich nach der Simulation zusammenzubauen. Ansonsten kann dies idealerweise auch asynchron zu Simulationsabschnitten, die keine Kommunikation zwischen den Knoten benötigen, realisiert werden. Dies würde während der Simulation die Netzwerkbandbreite für Kommunikation, die für die eigentliche Berechnung notwendig ist, freihalten. Außerdem haben viele lokale Speicher im Allgemeinen in Summe eine deutlich größere Bandbreite, als ein zentraler Netzwerkspeicher.

6.2 Parallelität

PISM setzt für eine gute Skalierbarkeit der Szenariogröße auf die Parallelisierung mit PETSc. Allerdings ist die Problemgröße hart begrenzt, da die Annahmen, die für SIA und SSA vorausgesetzt werden, nur bis zu einer gewissen Auflösung gelten [Ahl16]. Das Ziel sollte also nicht auf einer Verbesserung der schwachen Skalierungseigenschaften liegen, sondern langfristig muss die starke Skalierung verbessert werden. Wie im Amdahlschen-Gesetz (Abschnitt 3.7) zu sehen ist, läuft der Speedup gegen den Kehrwert des sequentiellen Anteils. Aus diesem Grund ist es wichtig, dass alle Komponenten möglichst optimal parallelisiert werden (z. B. `ocean`), da diese die Skalierung verschlechtern, wie man zum Beispiel von `ocean` schon bei einer Auflösung von 16 km sieht (Abbildung 4.3c).

Auch ist der aktuelle Parallelisierungsansatz ungeeignet für feine Auflösungen. Für diese Arbeit, war es beispielsweise nicht möglich Läufe zwischen 1 km und 2 km durchzuführen. Der Grund hierfür lag jedoch nicht in einer extrem langen Laufzeit, sondern darin, dass der Hauptspeicher der Cluster nicht ausreichte. Die hohe Parallelität ist bei feiner Auflösung aktuell vor allem für den Speicherplatz notwendig, das Netzwerk wird dabei zu einem so starker Flaschenhals, dass die Leistung der

CPUs quasi nicht mehr genutzt werden kann. Eine Möglichkeit den Hauptspeicherbedarf zu reduzieren, könnte darin bestehen, herleitbare Daten bei jedem Zugriff explizit zu generieren und nicht zwischendurch zu speichern. Solange dafür nicht zusätzlich kommuniziert werden muss, werden die Prozessoren besser ausgelastet und die Hauptspeicherauslastung geringer.

Dieser Ansatz ist auch für weitere Daten möglich, die normalerweise kommuniziert werden müssen. Bei einfach zu berechnenden Daten, kann es für die Laufzeit von Vorteil sein, wenn sie neu berechnet werden und nicht kommuniziert werden.

Eine weitere Möglichkeit parallele Hardware besser ausnutzen zu können, könnte darin bestehen einzelne Module auf unterschiedlicher Hardware parallel auszuführen. In *PISM: What does it do? And how does it work?* [Khr19] beschreibt Khrulev die Interaktion zwischen den einzelnen Modulen von PISM, wie in Abbildung 6.1 dargestellt. Außerdem beschreibt er, dass innerhalb eines Zeitschrittes folgende sequentielle Updatereihenfolge durchgeführt wird.

1. Basal yield stress
2. Stress balance
3. Zeitschrittweite
4. Age
5. Energy balance
6. Ice geometry (in Abhängigkeit vom Fluss)
7. Sea level
8. Sub-shelf Randwerte
9. Top-surface Randwerte
10. Ice geometry (in Abhängigkeit der Ergebnisse der Oberflächen und Basalen Massengleichungen)
11. Subglacial hydrology
12. Bed deformation

Diese beiden Informationen können nun miteinander kombiniert werden, indem aus Abbildung 6.1 alle Rückwärtskanten entfernt werden. Diese Rückwärtskanten können als Abhängigkeit für den nächsten Zeitschritt interpretiert werden, der auf den Ergebnissen dieses Zeitschritts aufbaut. `Basal yield stress` wird als erstes ausgeführt, demnach fällt die Kante von `stress balance` nach `basal yield stress` weg. `Stress balance` wird als zweites ausgeführt. Da `stress balance` keine eingehende Kante von `basal yield stress` hat, können alle eingehenden Kanten von `stress balance` gelöscht werden. `Energy balance` wird vor `subglacial hydrology` ausgeführt, wodurch auch die Kante zwischen `subglacial hydrology` und `energy balance` wegfällt.

Abbildung 6.2 zeigt den daraus entstehenden Graphen. Es fällt auf, dass der längste Pfad nur die Länge drei hat und theoretisch viele Abschnitte parallel zueinander ausgeführt werden können. Beispielsweise können während der sequentielle `ocean`-Code abgearbeitet wird, parallel dazu `age`, `sea level`, `bed deformation`, `subclacial hydrology` sowie die Oberflächenprozesse auf anderen Kernen ausgeführt werden. Dies würde bei Szenarien mit niedriger Auflösung insgesamt mehr nutzbare parallele Hardware bedeuten (vergleiche Abbildung 4.5).

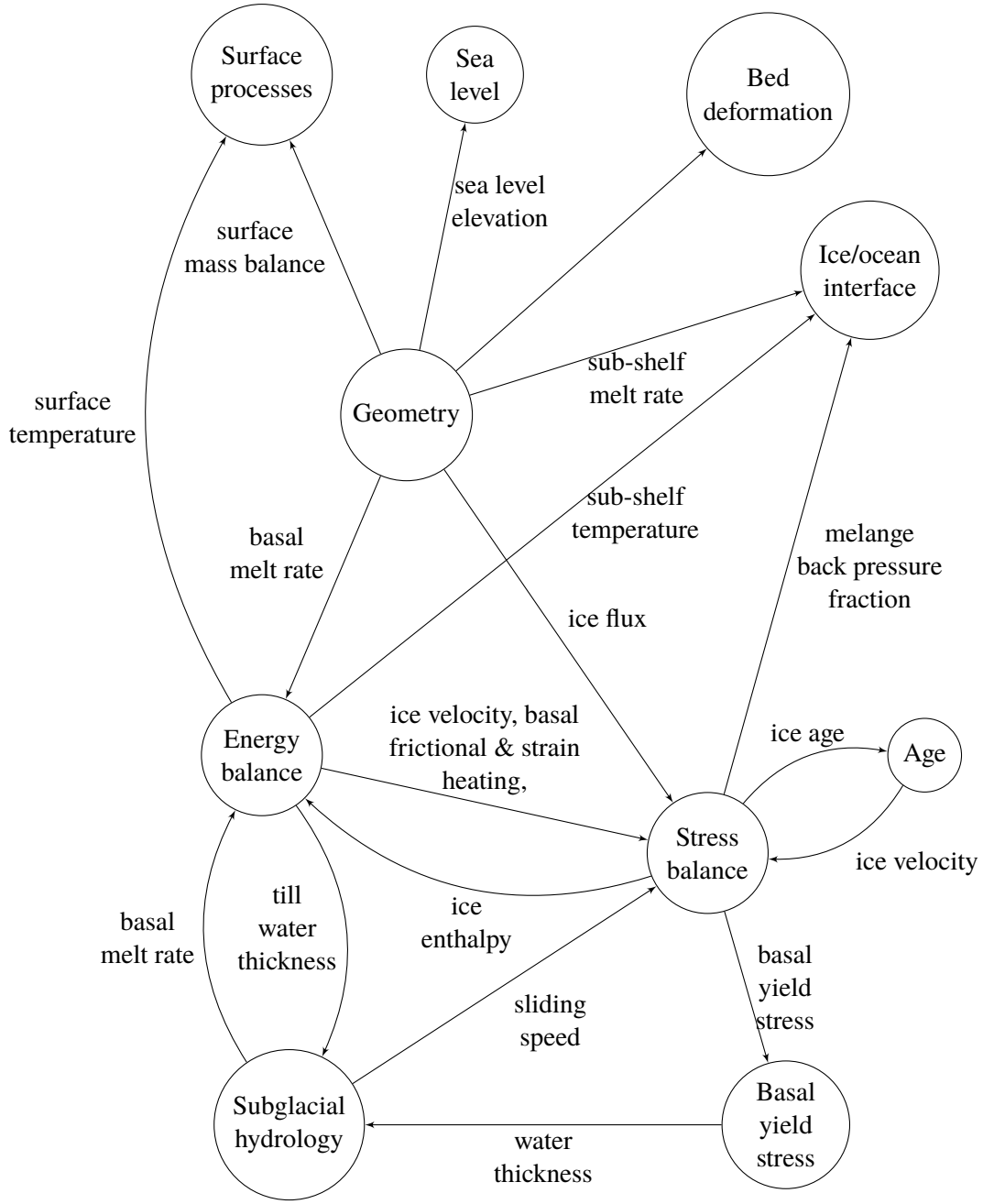


Abbildung 6.1: PISM Interaktionen zwischen den unterschiedlichen Modulen.

Quelle: In Anlehnung an [Khr19, S. 3]

Diese Erkenntnisse basieren dabei rein auf der Interpretation von *PISM: What does it do? And how does it work?* [Khr19] und nicht auf dem Programmcode. Bevor diese Idee allerdings umgesetzt werden kann, müssten also die Abhängigkeiten im Detail bestimmt werden. Zudem müsste man analysieren, wie nebenläufig welche Abschnitte ausgeführt werden können und wie viel Kommunikation zwischen den einzelnen Modulen nötig ist. Für Module, die große Datenmengen untereinander kommunizieren müssen, ist dieser Ansatz allerdings nicht geeignet, da über eine geteilte Hardware die Kommunikation deutlich schneller ist, als über das Netzwerk.

Sind die einzelnen Module voneinander abgegrenzt und mit definierten Schnittstellen versehen, bietet sich hierfür der Einsatz von High-Performance-ParalleX (HPX) [KWS+20] an, da hier die Abhängigkeiten im Hintergrund automatisch ermittelt werden und es dadurch möglich ist eine viel feingranularere Nebenläufigkeit zu erreichen und globale Barrieren komplett vermieden werden können.

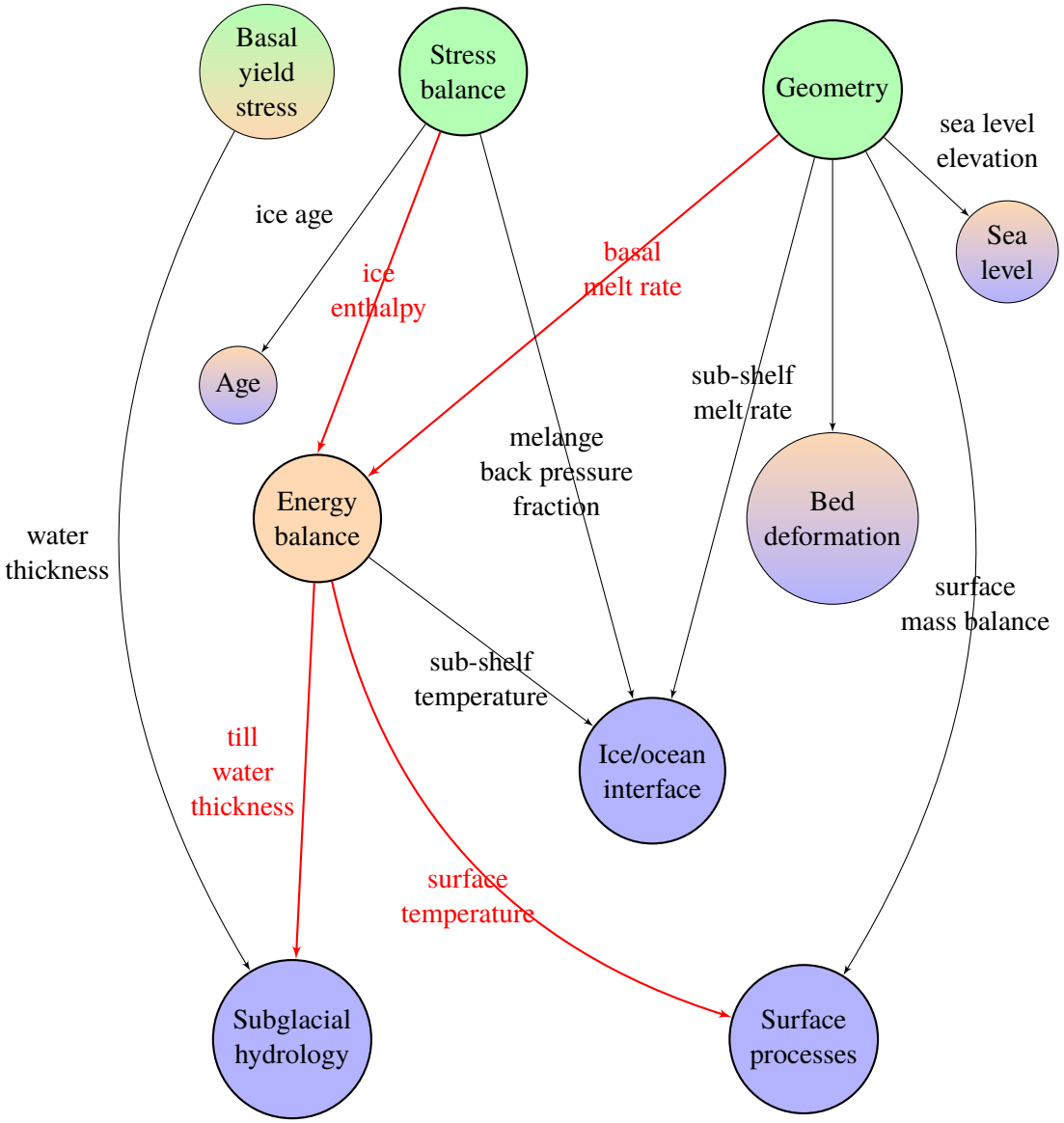


Abbildung 6.2: PISM Interaktionsmuster ohne die Rückwärtskanten innerhalb eines Update Schrittes.

7 Verbesserungsansatz

Zur Überprüfung der Umsetzbarkeit, wurde im Laufe dieser Arbeit begonnen eine matrixfreie Implementierung, für das Lösen der SSAFD Gleichung, zu implementieren.

7.1 Vorgehen

Begonnen wurde dafür in der `pism::stressbalance::SSAFDassemble_matrix()` Routine. Da die Implementierung hier auf Lesbarkeit und physikalische Nachvollziehbarkeit ausgelegt wurde, war zuerst nicht ersichtlich, wie genau die Matrix erstellt wird. Über semantische und boolesche Umformungen, wurde die Funktion so lange angepasst, bis ersichtlich war, von welchen Parametern der Wert einer Zelle genau abhängig ist und wie dieser berechnet wird.

Mit diesem Wissen wurde dann das eigentliche matrixfreie Lösen angegangen. Dies war direkt mit PETSc möglich. Dafür muss mit `MatShellSetOperation()` eine eigene Matrix-Vektor-Multiplikation gesetzt werden, die einen Vektor übergeben bekommen und diesen intern mit dem entsprechenden Matrixäquivalent multipliziert. In 5.1 würde dies dem entsprechen, dass die Zeile sieben selbstständig implementiert wird.

Zu Beginn der „Matrix“-Multiplikation, wird mit den acht möglichen Nachbarn, die im Simulationsgebiet angrenzen, kommuniziert, um die Vektorabschnitte, die in diesen benötigt werden zu kommunizieren. Aktuell ist diese Kommunikation noch sequentiell implementiert. Sie würde sich allerdings mit geringem Aufwand hinter den von diesen Daten unabhängigen Berechnungen verstecken lassen, indem erst die Datenpunkte abgearbeitet werden, die nicht an einer Außenkante des lokalen Gebietes liegen.

Die eigentliche „Matrix“-Vektor-Multiplikation läuft dann so ab, dass datenpunktweise die entsprechenden Matrixeinträge berechnet werden. Diese Matrixeinträge werden bei der Berechnung direkt mit dem entsprechenden Element des Eingangsvektors multipliziert und aufaddiert. Es werden also direkt die zwei Ergebniseinträge berechnet.

7.2 Erkenntnisse

Da die Ergebniseinträge der einzelnen Datenpunkte unabhängig voneinander sind, und die Matrix-Vektor-Multiplikation ein Großteil der GMRES(k)-Laufzeit beansprucht, wäre es interessant, hiermit direkt eine hybride Parallelisierung zu testen. Auf jedem Knoten wird dabei nur ein MPI-Rank gestartet, welcher dann die Matrix-Vektor-Multiplikation über OpenMP parallelisiert. Leider war es auch mithilfe der PISM Entwickler nicht möglich herauszufinden, warum innerhalb

der PETSc Funktionen OpenMP nicht parallelisiert, sodass dies nicht getestet werden konnte. Die PETSc-Dokumentation ist zum Teil sehr wenig aussagekräftig, sodass sie hier auch keine Hilfe war.

Die PISM Dokumentation ist sehr gut und ausführlich für Anwender. Leider wird im Handbuch nicht auf die Implementierung eingegangen und für Leser, die nicht mit der Eisschildphysik bewandert sind, ist die Doxygendokumentation für spezifischere Abschnitte eher dürftig, sodass erst durch diesen Implementierungsversuch klar wurde, dass hier sehr viele Datenabhängigkeiten bestehen.

Bei den Datenabhängigkeiten sorgt PISM im Hintergrund, dass sie geupdated und kommuniziert werden. Die Kommunikation für den Nutzer automatisch abzuhandeln, sorgt dafür, dass bei der Implementierung auf weniger geachtet werden muss. Es sorgt aber auch dafür, dass potenziell sehr teure Zugriffe gleich aussehen wie lokale Zugriffe, was es erschwert Performanceprobleme zu erkennen.

Bei der Kommunikation stellt PETSc Möglichkeiten bereit, die Kommunikation hinter anderen Abschnitten zu verstecken. Diese Möglichkeiten werden allerdings von PISM aktuell nicht genutzt.

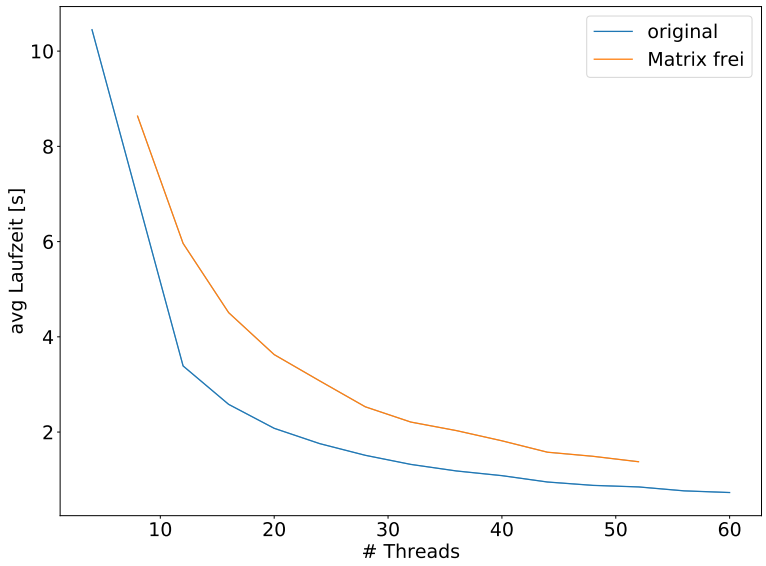


Abbildung 7.1: Laufzeitvergleich von KSPSolve mit Matrix freier Implementierung und im Ursprungszustand.

7.3 Ergebnisse

Die Implementierung hat keinerlei Anspruch effizient zu sein, sondern wurde als Prototyp implementiert, um das Gleichungssystem besser nachvollziehen zu können und die Abhängigkeiten genauer zu analysieren. Die hierbei gewonnenen Erkenntnisse führten unter anderem zu Unterunterabschnitt 5.3.3.4 und Unterunterabschnitt 5.3.3.5.

Wie in Abbildung 7.1 zu sehen ist, ist die nicht optimierte Implementierung, mit deutlich mehr Kommunikation, nicht signifikant langsamer, sodass davon ausgegangen werden kann, dass hier zumindest gleiche Performance erreicht werden kann. Dabei würde allerdings die Notwendigkeit einer Speicherung der Matrix wegfallen. Dies würde bei 4 km Auflösung dazu führen, dass für die 83 283 876 Einträge 666 MB und für die Indices der jeweiligen Einträge nochmal 666 MB weniger Hauptspeicher benötigt wird. Mit der gleichen Rechnung kommt man bei einer Auflösung von 1 km auf eine Reduzierung des benötigten Hauptspeichers von ungefähr 213 GB. Damit kann das Szenario mit weniger Knoten berechnet werden.

8 Fazit

Abschließend lässt sich sagen, dass PISM ein Forschungscode ist, der über die Zeit gewachsen ist, und dabei immer mehr an Funktionalität, Komplexität und Umfang gewonnen hat. Auch wenn, während der Implementierung der einzelnen dazukommenden Funktionalitäten, darauf geachtet wurde, diese effizient zu implementieren, war es vermutlich nie das primäre Ziel, einen hochperformanten Code zu entwickeln. Stattdessen ist ein Forschungscode entstanden, der sehr viele unterschiedliche physikalische Eigenschaften abbildet, dabei leicht zu warten sein muss und es möglichst einfach sein muss weitere Physik hinzuzufügen. Dieses benötigt ein hohes Maß an Flexibilität, um allen vergangenen und zukünftigen Einsatzszenarien bestmöglich gerecht zu werden, was allerdings durch teilweise starke Performance Einbußen erkauft wird.

Mit kleineren Kompromissen an der Codestruktur, sollte es möglich sein deutlich performantere Codealternativen für die Hauptmodule zu implementieren. Dazu bietet es sich an, für die Standardszenarien zusätzliche, speziell auf diese Szenarien beschränkte, hoch optimierte Modulalternativen zu entwickeln. Wie am Beispiel von GMRES gezeigt, kann eine simple Einschränkung der Matrixstruktur, bei einer Auflösung von 1 km, zu einer Reduktion von über 213 GB der benötigten Hauptspeicherkapazität führen.

Bei feiner Auflösung ist der Hauptspeicherbedarf groß, wodurch hier ein hoher Grad an Parallelität benötigt wird. Allerdings kann die aktuelle Implementierung, bei steigendem Grad an Parallelität, die zusätzliche Rechenleistung der Hardware nicht ausgenutzt werden. Einer der erarbeiteten Gründe hierfür ist, dass die PETSc-GMRES-Implementierung ungeeignet für den Einsatz in PISM ist, da die Parallelisierungsannahmen für den parallelen GMRES-Ansatz von PETSc nicht zutreffen.

Allgemein lässt sich zur Performance sagen, dass PISM eine aktuelle Hardware nicht effizient ausnutzen kann, und viel Verbesserungspotential besteht. Dabei existiert kein einzelner großer Hotspot, der einfach optimiert werden kann, sondern es müssen neben den aktuellen Hotspots viele kleine ineffiziente Abschnitte optimiert werden. Als Hotspots wurden das sequentielle `ocean` Modul, sowie das Lösen der SSA-Gleichung, bedingt durch die PETSc-GMRES-Implementierung, und das Schreiben der Ergebnisse identifiziert. Für die PETSc -GMRES-Implementierung wurde gezeigt, dass sie schon auf einem Kern sehr schlechte Laufzeiten aufweist. Auf einem Kern dürfte eine effiziente Implementierung maximal ein Siebtel der aktuellen Laufzeit benötigen.

Für die eingesetzte Hardware hat sich gezeigt, dass PISM von hohen Hauptspeicher- und Netzwerkbandbreiten profitiert, da die Performance, selbst bei einer optimalen Implementierung, durch die zur Verfügung stehende Bandbreite limitiert wird.

Als einen Ansatzpunkt, um die Kommunikation zu reduzieren, wurde die Gebietspartitionierung betrachtet. Hier wurde gezeigt, dass sich die Kommunikation für die Matrixmultiplikation in GMRES pro Thread auf ein Viertel reduzieren lässt.

Um einen höheren Parallelitätsgrad sinnvoll ausnutzen zu können, wurde die Möglichkeit aufgezeigt, die einzelnen Module nicht mehr sequentiell, sondern nebenläufig zueinander, auszuführen. Wie in der Arbeit deutlich wurde, ist dies von Vorteil, da PISM nicht beliebig hoch skaliert. Auch sollte diese Verbesserungsmöglichkeit, bei einer kompletten Modularisierung, relativ einfach umsetzbar sein.

8.1 Ausblick

Von der eigentlichen Verbesserung der Implementierung, lässt sich an dem Performanceverhalten von PISM in zukünftigen Arbeiten noch viel untersuchen. Da die Analyse nur auf einem Testszenario basiert, kann sie keine allgemeingültigen Aussagen über die Performance von PISM treffen. Hierfür wäre es wichtig, noch möglichst viele reale Szenarien zu betrachten. Aus den Unterschieden dieser Szenarien, und dem jeweils entsprechenden Performanceverhalten von PISM, sollten ableitbar sein, welche Parameter auf welche Art und Weise die Performance beeinträchtigen.

Auch bietet diese Arbeit nur für den GMRES(k) Löser der `stress_balance` eine detaillierte Analyse, also nur für einen sehr kleinen Teil von PISM. Je detaillierter die anderen Module in weiteren Arbeiten noch analysiert werden, desto genauer kann abgeschätzt werden welches Performanceverhalten unter den jeweils eingesetzten Algorithmen erreichbar ist.

Während dieser Arbeit wurde außerdem eine schlecht Skalierung im Hauptspeicherverbrauch sowie der Datenmenge der Ergebnisse festgestellt, deren Ursache in folgenden Analysen untersucht werden sollte.

Für weitere Parallelisierungen wäre es außerdem hilfreich, in weiteren Arbeiten die Datenabhängigkeiten der einzelnen Module genauer zu analysieren.

Nachdem festgestellt wurde, dass die GMRES-Implementierung ungeeignet ist, wäre es sinnvoll sich in nachfolgenden Arbeiten mit alternativen Implementierungs- und Parallelisierungsstrategien auseinander zu setzen. Als Ausgang hierfür kann, neben dieser Arbeit, unter anderem [ADEQO18; CD11; GAMV13; ICSB19; SS86; YHLD17] dienen.

Literaturverzeichnis

[201a] *8th Gen (S-platform) Intel® Processor Family Datasheet Vol.1*. Intel Corporation. Okt. 2017. URL: https://www.intel.com/content/dam/www/public/us/en/documents/datasheets/8th-gen-processor-family-s-platform-datasheet-vol-1.pdf (zitiert auf S. 35).

[201b] *Intel 64 and IA-32 Architectures Optimization Reference Manual*. Intel Cooperation. Juni 2016. URL: https://www.intel.com/content/dam/www/public/us/en/documents/manuals/64-ia-32-architectures-optimization-manual.pdf (zitiert auf S. 35).

[201c] *Intel 64 and IA-32 Architectures Optimization Reference Manual*. Intel Cooperation. Apr. 2018. URL: https://kib.kiev.ua/x86docs/Intel/SDMs/252046-058.pdf (zitiert auf S. 35).

[201d] *Intel 64 and IA-32 Architectures Optimization Reference Manual*. Intel Cooperation. Sep. 2019. URL: https://software.intel.com/sites/default/files/managed/9e/bc/64-ia-32-architectures-optimization-manual.pdf (zitiert auf S. 35).

[202] *Intel 64 and IA-32 Architectures Optimization Reference Manual*. Intel Cooperation. Mai 2020. URL: https://software.intel.com/content/dam/develop/public/us/en/documents/64-ia-32-architectures-optimization-manual.pdf (zitiert auf S. 35).

[AB09] A. Aschwanden, H. Blatter. „Mathematical modeling and numerical simulation of polythermal glaciers“. In: *Journal of Geophysical Research* 114.F1 (März 2009). DOI: 10.1029/2008jf001028 (zitiert auf S. 27, 28).

[ABKB12] A. Aschwanden, E. Bueler, C. Khroulev, H. Blatter. „An enthalpy formulation for glaciers and ice sheets“. In: *Journal of Glaciology* 58.209 (2012), S. 441–457. DOI: 10.3189/2012jog11j088 (zitiert auf S. 22, 25, 28, 29).

[ADEQO18] J. I. Aliaga, E. Dufrechou, P. Ezzatti, E. S. Quintana-Ortí. „An efficient GPU version of the preconditioned GMRES method“. In: *The Journal of Supercomputing* 75.3 (Okt. 2018), S. 1455–1469. DOI: 10.1007/s11227-018-2658-1 (zitiert auf S. 86).

[AM12] S. Adhikari, S. J. Marshall. „Parameterization of lateral drag in flowline models of glacier dynamics“. In: *Journal of Glaciology* 58.212 (2012), S. 1119–1132. DOI: 10.3189/2012jog12j018 (zitiert auf S. 22).

[Ahl16] J. Ahlkrona. „Computational Ice Sheet Dynamics: Error control and efficiency“. Diss. Acta Universitatis Upsaliensis, 2016 (zitiert auf S. 76).

[Amd67] G. M. Amdahl. „Validity of the single processor approach to achieving large scale computing capabilities“. In: *Proceedings of the April 18-20, 1967, spring joint computer conference on - AFIPS '67 (Spring)*. ACM Press, 1967. DOI: 10.1145/1465482.1465560 (zitiert auf S. 38).

[BAA+] S. Balay, S. Abhyankar, M. F. Adams, J. Brown, P. Brune, K. Buschelman, L. Dalcin, A. Dener, V. Eijkhout, W. D. Gropp, D. Karpeyev, D. Kaushik, M. G. Knepley, D. A. May, L. C. McInnes, R. T. Mills, T. Munson, K. Rupp, P. Sanan, B. F. Smith, S. Zampini, H. Zhang, H. Zhang. *PETSc FAQ*. https://www.mcs.anl.gov/petsc/documentation/faq.html. URL: https://www.mcs.anl.gov/petsc (zitiert auf S. 69).

[BAA+19] S. Balay, S. Abhyankar, M. F. Adams, J. Brown, P. Brune, K. Buschelman, L. Dalcin, A. Dener, V. Eijkhout, W. D. Gropp, D. Karpeyev, D. Kaushik, M. G. Knepley, D. A. May, L. C. McInnes, R. T. Mills, T. Munson, K. Rupp, P. Sanan, B. F. Smith, S. Zampini, H. Zhang, H. Zhang. *PETSc Web page*. https://www.mcs.anl.gov/petsc. 2019. URL: https://www.mcs.anl.gov/petsc (zitiert auf S. 25).

[BAA+20] S. Balay, S. Abhyankar, M. F. Adams, J. Brown, P. Brune, K. Buschelman, L. Dalcin, A. Dener, V. Eijkhout, W. D. Gropp, D. Karpeyev, D. Kaushik, M. G. Knepley, D. A. May, L. C. McInnes, R. T. Mills, T. Munson, K. Rupp, P. Sanan, B. F. Smith, S. Zampini, H. Zhang, H. Zhang. *PETSc Users Manual*. Techn. Ber. ANL-95/11 - Revision 3.14. Argonne National Laboratory, 2020. URL: https://www.mcs.anl.gov/petsc/petsc-current/docs/manual.pdf (zitiert auf S. 30, 45).

[BB09a] E. Bueler, J. Brown. „Shallow shelf approximation as a ßliding lawïn a thermodynamically coupled ice sheet model“. In: *J. Geophys. Res.* 114 (2009). DOI: 10.1029/2008JF001179. URL: http://www.agu.org/pubs/crossref/2009/2008JF001179.shtml (zitiert auf S. 23, 29).

[BB09b] E. Bueler, J. Brown. „Shallow shelf approximation as a “sliding law” in a thermomechanically coupled ice sheet model“. In: *Journal of Geophysical Research* 114.F3 (Juli 2009). DOI: 10.1029/2008jf001179 (zitiert auf S. 22, 23, 25).

[BBL07] E. Bueler, J. Brown, C. Lingle. „Exact solutions to the thermomechanically coupled shallow-ice approximation: effective tools for verification“. In: *Journal of Glaciology* 53.182 (2007), S. 499–516. DOI: 10.3189/002214307783258396 (zitiert auf S. 29).

[BDB+15] B. de Boer, A. M. Dolan, J. Bernales, E. Gasson, H. Goelzer, N. R. Golledge, J. Sutter, P. Huybrechts, G. Lohmann, I. Rogozhina, A. Abe-Ouchi, F. Saito, R. S. W. van de Wal. „Simulating the Antarctic ice sheet in the late-Pliocene warm period: PLISMIP-ANT, an ice-sheet model intercomparison project“. In: *The Cryosphere* 9.3 (Mai 2015), S. 881–903. DOI: 10.5194/tc-9-881-2015 (zitiert auf S. 21).

[BDE14] C. F. Brædstrup, A. Damsgaard, D. L. Egholm. „Ice-sheet modelling accelerated by graphics cards“. In: *Computers & Geosciences* 72 (Nov. 2014), S. 210–220. DOI: 10.1016/j.cageo.2014.07.019 (zitiert auf S. 23).

[BE12] C. Brædstrup, D. Egholm. „CUDA GPU based full-Stokes finite difference modelling of glaciers“. In: *EGUGA* (2012), S. 2932 (zitiert auf S. 23).

[BGAO10] H. Blatter, R. Greve, A. Abe-Ouchi. „A short history of the thermomechanical theory and modeling of glaciers and ice sheets“. In: *Journal of Glaciology* 56.200 (2010), S. 1087–1094. DOI: 10.3189/002214311796406059 (zitiert auf S. 22).

[BGMS97] S. Balay, W. D. Gropp, L. C. McInnes, B. F. Smith. „Efficient Management of Parallelism in Object Oriented Numerical Software Libraries“. In: *Modern Software Tools in Scientific Computing*. Hrsg. von E. Arge, A. M. Bruaset, H. P. Langtangen. Birkhäuser Press, 1997, S. 163–202. DOI: 10.1007/978-1-4612-1986-6_8 (zitiert auf S. 25).

[BJ13] D. J. Brinkerhoff, J. V. Johnson. „Data assimilation and prognostic whole ice sheet modelling with the variationally derived, higher order, open source, and fully parallel ice sheet model VarGlaS“. In: *The Cryosphere* 7.4 (Juli 2013), S. 1161–1184. DOI: 10.5194/tc-7-1161-2013 (zitiert auf S. 22).

[BKA+09] E. Bueler, C. Khroulev, A. Aschwanden, I. Joughin, B. E. Smith. „Modeled and observed fast flow in the Greenland ice sheet“. 2009. URL: http://pism.github.io/uaf-iceflow/BKAJS_submit2_twocolumn.pdf (zitiert auf S. 23, 28).

[BLB07] E. Bueler, C. S. Lingle, J. Brown. „Fast computation of a viscoelastic deformable Earth model for ice-sheet simulations“. In: *Annals of Glaciology* 46 (2007), S. 97–105. DOI: 10.3189/172756407782871567 (zitiert auf S. 27).

[BNAO+13] R. A. Bindschadler, S. Nowicki, A. Abe-Ouchi, A. Aschwanden, H. Choi, J. Fastook, G. Granzow, R. Greve, G. Gutowski, U. Herzfeld, C. Jackson, J. Johnson, C. Khroulev, A. Levermann, W. H. Lipscomb, M. A. Martin, M. Morlighem, B. R. Parizek, D. Pollard, S. F. Price, D. Ren, F. Saito, T. Sato, H. Seddik, H. Seroussi, K. Takahashi, R. Walker, W. L. Wang. „Ice-sheet model sensitivities to environmental forcing and their use in projecting future sea level (the SeaRISE project)“. In: *Journal of Glaciology* 59.214 (2013), S. 195–224. DOI: 10.3189/2013jog12j125 (zitiert auf S. 23).

[Bas10] J. Bassis. „Hamilton-type principles applied to ice-sheet dynamics: new approximations for large-scale ice-sheet flow“. In: *Journal of Glaciology* 56.197 (2010), S. 497–513. DOI: 10.3189/002214310792447761 (zitiert auf S. 22).

[CCLD98] S. D. L. Chapelle, O. Castelnau, V. Lipenkov, P. Duval. „Dynamic recrystallization and texture development in ice as revealed by the study of deep ice cores in Antarctica and Greenland“. In: *Journal of Geophysical Research: Solid Earth* 103.B3 (März 1998), S. 5091–5105. DOI: 10.1029/97jb02621 (zitiert auf S. 27).

[CD11] R. Couturier, S. Domas. „Sparse systems solving on GPUs with GMRES“. In: *The Journal of Supercomputing* 59.3 (Feb. 2011), S. 1504–1516. DOI: 10.1007/s11227-011-0562-z (zitiert auf S. 86).

[CLCV08] F. Cabecinhas, N. Lopes, R Crisotomo, L. Veiga. *Optimizing Binary Code Produced by Valgrind*. Techn. Ber. Citeseer, 2008. URL: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.144.6601&rep=rep1&type=pdf (zitiert auf S. 37).

[CLS20] G. Cheng, P. Lötstedt, L. von Sydow. „A full Stokes subgrid scheme in two dimensions for simulation of grounding line migration in ice sheets using Elmer/ICE (v8.3)“. In: *Geoscientific Model Development* 13.5 (Mai 2020), S. 2245–2258. DOI: 10.5194/gmd-13-2245-2020 (zitiert auf S. 23).

[CRR+18] L. Caesar, S. Rahmstorf, A. Robinson, G. Feulner, V. Saba. „Observed fingerprint of a weakening Atlantic Ocean overturning circulation“. In: *Nature* 556.7700 (Apr. 2018), S. 191–196. DOI: 10.1038/s41586-018-0006-5 (zitiert auf S. 19).

[Cuf10] K. M. Cuffey. *The physics of glaciers*. Burlington, MA: Butterworth-Heinemann/Elsevier, 2010. ISBN: 9780123694614 (zitiert auf S. 28).

[DG11] D. N. Della-Giustina. „Regional Modeling of Greenland's Outlet Glaciers with the Parallel Ice Sheet Model". Diss. University of Alaska Fairbanks, 2011 (zitiert auf S. 25).

[EKCL11] D. L. Egholm, M. F. Knudsen, C. D. Clark, J. E. Lesemann. „Modeling the flow of glaciers in steep terrains: The integrated second-order shallow ice approximation (iSOSIA)". In: *Journal of Geophysical Research: Earth Surface* 116.F2 (Mai 2011). DOI: 10.1029/2010jf001900 (zitiert auf S. 22).

[ESW+12] K. J. Evans, A. G. Salinger, P. H. Worley, S. F. Price, W. H. Lipscomb, J. A. Nichols, J. B. White, M. Perego, M. Vertenstein, J. Edwards, J.-F. Lemieux. „A modern solver interface to manage solution algorithms in the Community Earth System Model". In: *The International Journal of High Performance Computing Applications* 26.1 (Feb. 2012), S. 54–62. DOI: 10.1177/1094342011435159 (zitiert auf S. 22).

[FFF97] A. C. Fowler, A. C. Fowler, A. Fowler. *Mathematical models in the applied sciences*. Bd. 17. Cambridge University Press, 1997 (zitiert auf S. 28).

[FGEXDS20] A. F. A. Furtunato, K. Georgiou, K. Eder, S. Xavier-De-Souza. „When Parallel Speedups Hit the Memory Wall". In: *IEEE Access* 8 (2020), S. 79225–79238. DOI: 10.1109/access.2020.2990418 (zitiert auf S. 38).

[FPV+13] P. Fretwell, H. D. Pritchard, D. G. Vaughan, J. L. Bamber, N. E. Barrand, R. Bell, C. Bianchi, R. G. Bingham, D. D. Blankenship, G. Casassa, G. Catania, D. Callens, H. Conway, A. J. Cook, H. F. J. Corr, D. Damaske, V. Damm, F. Ferraccioli, R. Forsberg, S. Fujita, Y. Gim, P. Gogineni, J. A. Griggs, R. C. A. Hindmarsh, P. Holmlund, J. W. Holt, R. W. Jacobel, A. Jenkins, W. Jokat, T. Jordan, E. C. King, J. Kohler, W. Krabill, M. Riger-Kusk, K. A. Langley, G. Leitchenkov, C. Leuschen, B. P. Luyendyk, K. Matsuoka, J. Mouginot, F. O. Nitsche, Y. Nogi, O. A. Nost, S. V. Popov, E. Rignot, D. M. Rippin, A. Rivera, J. Roberts, N. Ross, M. J. Siegert, A. M. Smith, D. Steinhage, M. Studinger, B. Sun, B. K. Tinto, B. C. Welch, D. Wilson, D. A. Young, C. Xiangbin, A. Zirizzotti. „Bedmap2: improved ice bed, surface and thickness datasets for Antarctica". In: *The Cryosphere* 7.1 (Feb. 2013), S. 375–393. DOI: 10.5194/tc-7-375-2013 (zitiert auf S. 19, 31, 32).

[FW98] A. G. Fountain, J. S. Walder. „Water flow through temperate glaciers". In: *Reviews of Geophysics* 36.3 (Aug. 1998), S. 299–328. DOI: 10.1029/97rg03579 (zitiert auf S. 28).

[FWP+11] J. G. Fyke, A. J. Weaver, D. Pollard, M. Eby, L. Carter, A. Mackintosh. „A new coupled ice sheet/climate model: description and sensitivity to model physics under Eemian, Last Glacial Maximum, late Holocene and modern climate conditions". In: *Geoscientific Model Development* 4.1 (März 2011), S. 117–136. DOI: 10.5194/gmd-4-117-2011 (zitiert auf S. 21).

[Fas94] J. Fastook. „Modeling the ice age: the finite-element method in glaciology". In: *IEEE Computational Science and Engineering* 1.1 (1994), S. 55–67. DOI: 10.1109/99.295374 (zitiert auf S. 21).

[Fog20] A. Fog. *The microarchitecture of Intel, AMD and VIA CPUs: An optimization guide for assembly programmers and compiler makers*. 2020-09-02. Technical University of Denmark. 2020. URL: https://www.agner.org/optimize/microarchitecture.pdf (zitiert auf S. 35).

[For15] M. P. I. Forum. *MPI: A Message-passing Interface Standard, Version 3.1 ; June 4, 2015*. High-Performance Computing Center Stuttgart, University of Stuttgart, 2015. URL: https://books.google.de/books?id=Fbv7jwEACAAJ (zitiert auf S. 17, 19).

[Fou11] A. Fountain. *Encyclopedia of Snow, Ice and Glaciers*. Springer-Verlag GmbH, 1. Juli 2011. ISBN: 904812641X. URL: https://www.ebook.de/de/product/9573573/encyclopedia_of_snow_ice_and_glaciers.html (zitiert auf S. 28).

[GAMV13] P. Ghysels, T. J. Ashby, K. Meerbergen, W. Vanroose. „Hiding Global Communication Latency in the GMRES Algorithm on Massively Parallel Machines“. In: *SIAM Journal on Scientific Computing* 35.1 (Jan. 2013), S. C48–C71. DOI: 10.1137/12086563x (zitiert auf S. 58, 69, 86).

[GNE+18] H. Goelzer, S. Nowicki, T. Edwards, M. Beckley, A. Abe-Ouchi, A. Aschwanden, R. Calov, O. Gagliardini, F. Gillet-Chaulet, N. R. Golledge, J. Gregory, R. Greve, A. Humbert, P. Huybrechts, J. H. Kennedy, E. Larour, W. H. Lipscomb, S. L. clec'h, V. Lee, M. Morlighem, F. Pattyn, A. J. Payne, C. Rodehacke, M. Rückamp, F. Saito, N. Schlegel, H. Seroussi, A. Shepherd, S. Sun, R. van de Wal, F. A. Ziemen. „Design and results of the ice sheet model initialisation experiments initMIP-Greenland: an ISMIP6 intercomparison“. In: *The Cryosphere* 12.4 (Apr. 2018), S. 1433–1460. DOI: 10.5194/tc-12-1433-2018 (zitiert auf S. 23).

[GSAO11] R. Greve, F. Saito, A. Abe-Ouchi. „Initial results of the SeaRISE numerical experiments with the models SICOPOLIS and IcIES for the Greenland ice sheet“. In: *Annals of Glaciology* 52.58 (2011), S. 23–30. DOI: 10.3189/172756411797252068 (zitiert auf S. 21).

[GZ08] O. Gagliardini, T. Zwinger. „The ISMIP-HOM benchmark experiments performed using the Finite-Element code Elmer“. In: *The Cryosphere* 2.1 (Juni 2008), S. 67–76. DOI: 10.5194/tc-2-67-2008 (zitiert auf S. 22).

[GZGC+13] O. Gagliardini, T. Zwinger, F. Gillet-Chaulet, G. Durand, L. Favier, B. de Fleurian, R. Greve, M. Malinen, C. Martín, P. Råback, J. Ruokolainen, M. Sacchettini, M. Schäfer, H. Seddik, J. Thies. „Capabilities and performance of Elmer/Ice, a new-generation ice sheet model“. In: *Geoscientific Model Development* 6.4 (Aug. 2013), S. 1299–1318. DOI: 10.5194/gmd-6-1299-2013 (zitiert auf S. 22, 23).

[Gol11] D. N. Goldberg. „A variationally derived, depth-integrated approximation to a higher-order glaciological flow model“. In: *Journal of Glaciology* 57.201 (2011), S. 157–170. DOI: 10.3189/002214311795306763 (zitiert auf S. 22).

[Gre97] R. Greve. „Application of a polythermal three-dimensional ice sheet model to the Greenland ice sheet: response to steady-state and transient climate scenarios“. In: *Journal of Climate* 10.5 (1997), S. 901–918 (zitiert auf S. 27).

[Gus88] J. L. Gustafson. „Reevaluating Amdahl's law“. In: *Communications of the ACM* 31.5 (Mai 1988), S. 532–533. DOI: 10.1145/42411.42415 (zitiert auf S. 38).

[HBH+05] M. A. Heroux, R. A. Bartlett, V. E. Howle, R. J. Hoekstra, J. J. Hu, T. G. Kolda, R. B. Lehoucq, K. R. Long, R. P. Pawlowski, E. T. Phipps, A. G. Salinger, H. K. Thornquist, R. S. Tuminaro, J. M. Willenbring, A. Williams, K. S. Stanley. „An overview of the Trilinos project“. In: *ACM Transactions on Mathematical Software* 31.3 (Sep. 2005), S. 397–423. DOI: 10.1145/1089014.1089021 (zitiert auf S. 22).

[HM01] R. C. Hindmarsh, E. L. Meur. „Dynamical processes involved in the retreat of marine ice sheets“. In: *Journal of Glaciology* 47.157 (2001), S. 271–282. DOI: 10.3189/172756501781832269 (zitiert auf S. 22, 28).

[Heg12] K. Hegedűs. „Scenarios and roots of climate change“. In: (Jan. 2012). URL: http://unipub.lib.uni-corvinus.hu/1008/1/Price_2012_Hegedus.pdf (zitiert auf S. 31).

[Hut83] K. Hutter. *Theoretical Glaciology*. Springer-Verlag GmbH, März 1983. URL: https://www.ebook.de/de/product/39440864/k_hutter_theoretical_glaciology.html (zitiert auf S. 22, 28).

[ICSB19] E. I. Ioannidis, N. Cheimarios, A. N. Spyropoulos, A. G. Boudouvis. „On the performance of various parallel GMRES implementations on CPU and GPU clusters“. In: (Juni 2019). arXiv: 1906.04051 [cs.DC] (zitiert auf S. 86).

[ISG15] T. Isaac, G. Stadler, O. Ghattas. „Solution of Nonlinear Stokes Equations Discretized By High-Order Finite Elements on Nonconforming and Anisotropic Meshes, with Application to Ice Sheet Dynamics“. In: *SIAM Journal on Scientific Computing* 37.6 (Jan. 2015), B804–B833. DOI: 10.1137/140974407 (zitiert auf S. 22).

[JPRB08] G. Jouvet, M. Picasso, J. Rappaz, H. Blatter. „A new algorithm to simulate the dynamics of a glacier: theory and applications“. In: *Journal of Glaciology* 54.188 (2008), S. 801–811. DOI: 10.3189/002214308787780049 (zitiert auf S. 22).

[Jar08] A. Jarosch. „Icetools: A full Stokes finite element model for glaciers“. In: *Computers & Geosciences* 34.8 (Aug. 2008), S. 1005–1014. DOI: 10.1016/j.cageo.2007.06.012 (zitiert auf S. 22).

[Jen77] D. Jenssen. „A Three-Dimensional Polar Ice-Sheet Model“. In: *Journal of Glaciology* 18.80 (1977), S. 373–389. DOI: 10.3189/s0022143000021067 (zitiert auf S. 22).

[KWS+20] H. Kaiser, B. A. L. A. Wash, M. Simberg, T. Heller, A. Bergé, J. Biddiscombe, R. Auriane, A. Bikineev, G. Mercer, A. Schäfer, K. Huck, A. S. Lemoine, Taeguk Kwon, J. Habraken, M. Anderson, M. Copik, S. R. Brandt, M. Stumpf, D. Bourgeois, D. Blank, Rstobaugh, S. Jakobovits, V. Amatya, L. Viklund, Nikunj Gupta, P. Diehl, Z. Khatami, Devang Bacharwar, Tapasweni Pathak, Shuangyang Yang. *STEllAR-GROUP/hpx: HPX V1.5.1: The C++ Standards Library for Parallelism and Concurrency*. 2020. DOI: 10.5281/ZENODO.4059746 (zitiert auf S. 79).

[Khr19] C. Khrulev. *PISM: What does it do? And how does it work?* Praesentation. Okt. 2019. URL: https://github.com/ckhroulev/pism-nims/raw/f9c7d8d85ca913b71422293a0d6cb8b54ccf6227/pism-inner-workings.pdf (zitiert auf S. 26, 77–79).

[Kum11] P. Kumbhar. „Performance of petsc gpu implementation with sparse matrix storage schemes“. Diss. Master's thesis, The University of Edinburgh (Aug 2011), 2011. URL: https://static.epcc.ed.ac.uk/dissertations/hpc-msc/2010-2011/PramodKumbhar.pdf.

[LBDP89] V. Lipenkov, N. Barkov, P. Duval, P. Pimienta. „Crystalline Texture of the 2083 m Ice Core at Vostok Station, Antarctica“. In: *Journal of Glaciology* 35.121 (1989), S. 392–398. DOI: 10.3189/s0022143000009321 (zitiert auf S. 27).

[LC85] C. S. Lingle, J. A. Clark. „A numerical model of interactions between a marine ice sheet and the solid earth: Application to a West Antarctic ice stream“. In: *Journal of Geophysical Research* 90.C1 (1985), S. 1100. DOI: 10.1029/jc090ic01p01100 (zitiert auf S. 27).

[LFV+13] W. H. Lipscomb, J. G. Fyke, M. Vizcaíno, W. J. Sacks, J. Wolfe, M. Vertenstein, A. Craig, E. Kluzek, D. M. Lawrence. „Implementation and Initial Evaluation of the Glimmer Community Ice Sheet Model in the Community Earth System Model“. In: *Journal of Climate* 26.19 (Sep. 2013), S. 7352–7371. DOI: 10.1175/jcli-d-12-00557.1 (zitiert auf S. 21).

[LJG+12] W. Leng, L. Ju, M. Gunzburger, S. Price, T. Ringler. „A parallel high-order accurate finite element nonlinear Stokes ice sheet model and benchmark experiments“. In: *Journal of Geophysical Research: Earth Surface* 117.F1 (Jan. 2012), n/a–n/a. DOI: 10.1029/2011jf001962 (zitiert auf S. 22).

[LJGP14] W. Leng, L. Ju, M. Gunzburger, S. Price. „A Parallel Computational Model for Three-Dimensional, Thermo-Mechanical Stokes Flow Simulations of Glaciers and Ice Sheets“. In: *Communications in Computational Physics* 16.4 (Okt. 2014), S. 1056–1080. DOI: 10.4208/cicp.310813.010414a (zitiert auf S. 22).

[LMP+07] T. M. Lenton, R. Marsh, A. R. Price, D. J. Lunt, Y. Aksenov, J. D. Annan, T. Cooper-Chadwick, S. J. Cox, N. R. Edwards, S. Goswami, J. C. Hargreaves, P. P. Harris, Z. Jiao, V. N. Livina, A. J. Payne, I. C. Rutt, J. G. Shepherd, P. J. Valdes, G. Williams, M. S. Williamson, A. Yool. „Effects of atmospheric dynamics and ocean resolution on bi-stability of the thermohaline circulation examined using the Grid ENabled Integrated Earth system modelling (GENIE) framework“. In: *Climate Dynamics* 29.6 (Juni 2007), S. 591–613. DOI: 10.1007/s00382-007-0254-9 (zitiert auf S. 21).

[LOC95] M. Lozier, W. Owens, R. G. Curry. „The climatology of the North Atlantic“. In: *Progress in Oceanography* 36.1 (Jan. 1995), S. 1–44. DOI: 10.1016/0079-6611(95)00013-5 (zitiert auf S. 19).

[LPH+19] W. H. Lipscomb, S. F. Price, M. J. Hoffman, G. R. Leguy, A. R. Bennett, S. L. Bradley, K. J. Evans, J. G. Fyke, J. H. Kennedy, M. Perego et al. „Description and evaluation of the Community Ice Sheet Model (CISM) v2. 1“. In: *Geoscientific Model Development* 12.1 (2019), S. 387–424. URL: https://opensky.ucar.edu/islandora/object/articles:22287/datastream/PDF/download/citation.pdf (zitiert auf S. 21, 22).

[LS12] W. Lipscomb, W. Sacks. „The CESM Land Ice Model (CISM): Documentation and User's Guide“. In: (2012). URL: http://www.cesm.ucar.edu/models/cesm1.2/clm/models/glc/cism/doc/cism_doc.pdf (zitiert auf S. 22).

[LSMR12] E. Larour, H. Seroussi, M. Morlighem, E. Rignot. „Continental scale, high order, high spatial resolution, ice sheet modeling using the Ice Sheet System Model (ISSM)“. In: *Journal of Geophysical Research: Earth Surface* 117.F1 (März 2012), n/a–n/a. DOI: 10.1029/2011jf002140 (zitiert auf S. 22, 23).

[MBH+02] D. T. Marr, F. Binns, D. L. Hill, G. Hinton, D. A. Koufaty, J. A. Miller, M. Upton. „Hyper-Threading Technology Architecture and Microarchitecture.“ In: *Intel Technology Journal* 6.1 (2002) (zitiert auf S. 46).

[MC98] A. Mangeney, F. Califano. „The shallow ice approximation for anisotropic ice: Formulation and limits“. In: *Journal of Geophysical Research: Solid Earth* 103.B1 (Jan. 1998), S. 691–705. DOI: 10.1029/97jb02539 (zitiert auf S. 22).

[MVUB+08] E. van Meijgaard, L. Van Ulft, W. Van de Berg, F. Bosveld, B. Van den Hurk, G Lenderink, A. Siebesma. *The KNMI regional atmospheric climate model RACMO, version 2.1*. Techn. Ber. Koninklijk Nederlands Meteorologisch Instituut, 2008. URL: http://bibliotheek.knmi.nl/knmipubTR/TR302.pdf (zitiert auf S. 21).

[MWH+11] M. A. Martin, R. Winkelmann, M. Haseloff, T. Albrecht, E. Bueler, C. Khroulev, A. Levermann. „The Potsdam Parallel Ice Sheet Model (PISM-PIK) – Part 2: Dynamic equilibrium simulation of the Antarctic ice sheet“. In: *The Cryosphere* 5.3 (Sep. 2011), S. 727–740. DOI: 10.5194/tc-5-727-2011 (zitiert auf S. 26).

[MWR+17] M. Morlighem, C. N. Williams, E. Rignot, L. An, J. E. Arndt, J. L. Bamber, G. Catania, N. Chauché, J. A. Dowdeswell, B. Dorschel, I. Fenty, K. Hogan, I. Howat, A. Hubbard, M. Jakobsson, T. M. Jordan, K. K. Kjeldsen, R. Millan, L. Mayer, J. Mouginot, B. P. Y. Noël, C. O'Cofaigh, S. Palmer, S. Rysgaard, H. Seroussi, M. J. Siegert, P. Slabon, F. Straneo, M. R. van den Broeke, W. Weinrebe, M. Wood, K. B. Zinglersen. „BedMachine v3: Complete Bed Topography and Ocean Bathymetry Mapping of Greenland From Multibeam Echo Sounding Combined With Mass Conservation“. In: *Geophysical Research Letters* 44.21 (Nov. 2017). DOI: 10.1002/2017gl074954 (zitiert auf S. 19).

[MZ87] L. W. Morland, R. Zainuddin. „Plane and Radial Ice-Shelf Flow with Prescribed Temperature Profile“. In: *Dynamics of the West Antarctic Ice Sheet*. Springer Netherlands, 1987, S. 117–140. DOI: 10.1007/978-94-009-3745-1_7 (zitiert auf S. 28).

[Mac89] D. R. MacAyeal. „Large-scale ice flow over a viscous basal sediment: Theory and application to ice stream B, Antarctica“. In: *Journal of Geophysical Research: Solid Earth* 94.B4 (1989), S. 4071–4087. DOI: 10.1029/jb094ib04p04071 (zitiert auf S. 28).

[Mah76] M. W. Mahaffy. „A three-dimensional numerical model of ice sheets: Tests on the Barnes Ice Cap, Northwest Territories“. In: *Journal of Geophysical Research* 81.6 (Feb. 1976), S. 1059–1066. DOI: 10.1029/jc081i006p01059 (zitiert auf S. 22).

[Mor87] L. W. Morland. „Unconfined Ice-Shelf Flow“. In: *Dynamics of the West Antarctic Ice Sheet*. Springer Netherlands, 1987, S. 99–116. DOI: 10.1007/978-94-009-3745-1_6 (zitiert auf S. 28).

[NBAO+13a] S. Nowicki, R. A. Bindschadler, A. Abe-Ouchi, A. Aschwanden, E. Bueler, H. Choi, J. Fastook, G. Granzow, R. Greve, G. Gutowski, U. Herzfeld, C. Jackson, J. Johnson, C. Khroulev, E. Larour, A. Levermann, W. H. Lipscomb, M. A. Martin, M. Morlighem, B. R. Parizek, D. Pollard, S. F. Price, D. Ren, E. Rignot, F. Saito, T. Sato, H. Seddik, H. Seroussi, K. Takahashi, R. Walker, W. L. Wang. „Insights into spatial sensitivities of ice mass response to environmental change from the SeaRISE ice sheet modeling project I: Antarctica“. In: *Journal of Geophysical Research: Earth Surface* 118.2 (Juni 2013), S. 1002–1024. DOI: 10.1002/jgrf.20081 (zitiert auf S. 23).

[NBAO+13b] S. Nowicki, R. A. Bindschadler, A. Abe-Ouchi, A. Aschwanden, E. Bueler, H. Choi, J. Fastook, G. Granzow, R. Greve, G. Gutowski, U. Herzfeld, C. Jackson, J. Johnson, C. Khroulev, E. Larour, A. Levermann, W. H. Lipscomb, M. A. Martin, M. Morlighem, B. R. Parizek, D. Pollard, S. F. Price, D. Ren, E. Rignot, F. Saito, T. Sato, H. Seddik, H. Seroussi, K. Takahashi, R. Walker, W. L. Wang. „Insights into spatial sensitivities of ice mass response to environmental change from the SeaRISE ice sheet modeling project II: Greenland“. In: *Journal of Geophysical Research: Earth Surface* 118.2 (Juni 2013), S. 1025–1044. DOI: 10.1002/jgrf.20076 (zitiert auf S. 23).

[NS03] N. Nethercote, J. Seward. „Valgrind: A program supervision framework“. In: *Electronic notes in theoretical computer science* 89.2 (2003), S. 44–66. DOI: 10.1016/S1571-0661(04)81042-9 (zitiert auf S. 37).

[NS07] N. Nethercote, J. Seward. „Valgrind: a framework for heavyweight dynamic binary instrumentation“. In: *ACM Sigplan notices* 42.6 (2007), S. 89–100. DOI: 10.1145/1273442.1250746 (zitiert auf S. 37).

[PD12] D. Pollard, R. M. DeConto. „Description of a hybrid ice sheet-shelf model, and application to Antarctica“. In: *Geoscientific Model Development* 5.5 (Okt. 2012), S. 1273–1295. DOI: 10.5194/gmd-5-1273-2012 (zitiert auf S. 22).

[PGB12] M. Perego, M. Gunzburger, J. Burkardt. „Parallel finite-element implementation for higher-order ice-sheet models“. In: *Journal of Glaciology* 58.207 (2012), S. 76–88. DOI: 10.3189/2012jog11j063 (zitiert auf S. 22).

[Pa15] the PISM authors. *PISM, a Parallel Ice Sheet Model*. 2015. URL: http://www.pism-docs.org (zitiert auf S. 26–29, 54).

[Pat03] F. Pattyn. „A new three-dimensional higher-order thermomechanical ice sheet model: Basic sensitivity, ice stream development, and ice flow across subglacial lakes“. In: *Journal of Geophysical Research* 108.B8 (2003). DOI: 10.1029/2002jb002329 (zitiert auf S. 22, 28).

[Pat17] F. Pattyn. „Sea-level response to melting of Antarctic ice shelves on multi-centennial timescales with the fast Elementary Thermomechanical Ice Sheet model (f.ETISh v1.0)“. In: *The Cryosphere* 11.4 (Aug. 2017), S. 1851–1878. DOI: 10.5194/tc-11-1851-2017 (zitiert auf S. 21).

[QDR+18] A. Quiquet, C. Dumas, C. Ritz, V. Peyaud, D. M. Roche. „The GRISLI ice sheet model (version 2.0): calibration and validation for multi-millennial changes of the Antarctic ice sheet“. In: *Geoscientific Model Development* 11.12 (Dez. 2018), S. 5003–5025. DOI: 10.5194/gmd-11-5003-2018 (zitiert auf S. 21).

[RHHP09] I. C. Rutt, M Hagdorn, N. Hulton, A. Payne. „The Glimmer community ice sheet model“. In: *Journal of Geophysical Research: Earth Surface* 114.F2 (2009). DOI: 10.1029/2008JF001015 (zitiert auf S. 21).

[RLH+19] L. Räss, A. Licul, F. Herman, Y. Y. Podladchikov, J. Suckale. „Modelling thermo-mechanical ice deformation using a GPU-based implicit pseudo-transient method (FastICE v1.0)“. In: (Sep. 2019). DOI: 10.5194/gmd-2019-249 (zitiert auf S. 23).

[RMS+19] E. Rignot, J. Mouginot, B. Scheuchl, M. van den Broeke, M. J. van Wessem, M. Morlighem. „Four decades of Antarctic Ice Sheet mass balance from 1979–2017“. In: *Proceedings of the National Academy of Sciences* 116.4 (Jan. 2019), S. 1095–1103. DOI: 10.1073/pnas.1812883116 (zitiert auf S. 31).

[Rig11] E. Rignot. „Is Antarctica melting?“ In: *Wiley Interdisciplinary Reviews: Climate Change* 2.3 (Apr. 2011), S. 324–331. DOI: 10.1002/wcc.110 (zitiert auf S. 19).

[SBG04] B. Smith, P. Bjorstad, W. Gropp. *Domain decomposition: parallel multilevel methods for elliptic partial differential equations*. Cambridge university press, 2004 (zitiert auf S. 57).

[SG16] H. Seddik, R. Greve. „Full Stokes finite-element modeling of ice sheets using a graphics processing unit“. In: *AGUFM* 2016 (2016), T23C–2947 (zitiert auf S. 23).

[SH10] C. Schoof, R. C. A. Hindmarsh. „Thin-Film Flows with Wall Slip: An Asymptotic Analysis of Higher Order Glacier Flow Models“. In: *The Quarterly Journal of Mechanics and Applied Mathematics* 63.1 (Jan. 2010), S. 73–114. DOI: 10.1093/qjmam/hbp025 (zitiert auf S. 22).

[SNS+19] H. Seroussi, S. Nowicki, E. Simon, A. Abe-Ouchi, T. Albrecht, J. Brondex, S. Cornford, C. Dumas, F. Gillet-Chaulet, H. Goelzer, N. R. Golledge, J. M. Gregory, R. Greve, M. J. Hoffman, A. Humbert, P. Huybrechts, T. Kleiner, E. Larour, G. Leguy, W. H. Lipscomb, D. Lowry, M. Mengel, M. Morlighem, F. Pattyn, A. J. Payne, D. Pollard, S. F. Price, A. Quiquet, T. J. Reerink, R. Reese, C. B. Rodehacke, N.-J. Schlegel, A. Shepherd, S. Sun, J. Sutter, J. V. Breedam, R. S. W. van de Wal, R. Winkelmann, T. Zhang. „initMIP-Antarctica: an ice sheet model initialization experiment of ISMIP6“. In: *The Cryosphere* 13.5 (Mai 2019), S. 1441–1471. DOI: 10.5194/tc-13-1441-2019 (zitiert auf S. 23).

[SQM+07] S Solomon, D Qin, M Manning, Z Chen, M Marquis, K Averyt, M Tignor, H Miller. *The physical science basis, 235–337, IPCC report AR4, New York and Cambridge*. 2007. URL: https://www.ipcc.ch/site/assets/uploads/2018/05/ar4_wg1_full_report-1.pdf#page=246 (zitiert auf S. 23).

[SRR+18] W. Steffen, J. Rockström, K. Richardson, T. M. Lenton, C. Folke, D. Liverman, C. P. Summerhayes, A. D. Barnosky, S. E. Cornell, M. Crucifix, J. F. Donges, I. Fetzer, S. J. Lade, M. Scheffer, R. Winkelmann, H. J. Schellnhuber. „Trajectories of the Earth System in the Anthropocene“. In: *Proceedings of the National Academy of Sciences* 115.33 (Aug. 2018), S. 8252–8259. DOI: 10.1073/pnas.1810141115 (zitiert auf S. 19).

[SS86] Y. Saad, M. H. Schultz. „GMRES: A generalized minimal residual algorithm for solving nonsymmetric linear systems“. In: *SIAM Journal on scientific and statistical computing* 7.3 (1986), S. 856–869. DOI: 10.1137/0907058 (zitiert auf S. 58, 86).

[Sch06] C. Schoof. „A variational approach to ice stream flow“. In: *Journal of Fluid Mechanics* 556 (Mai 2006), S. 227. DOI: 10.1017/s0022112006009591 (zitiert auf S. 28).

[TO09] J. Turner, J. Overland. „Contrasting climate change in the two polar regions“. In: *Polar Research* 28.2 (2009), S. 146–164. DOI: 10.1111/j.1751-8369.2009.00128.x (zitiert auf S. 31).

[TPS+15] I. K. Tezaur, M. Perego, A. G. Salinger, R. S. Tuminaro, S. F. Price. „Albany/FELIX: a parallel, scalable and robust, finite element, first-order Stokes approximation ice sheet solver built for advanced analysis“. In: *Geoscientific Model Development* 8.4 (Apr. 2015), S. 1197–1220. DOI: 10.5194/gmd-8-1197-2015 (zitiert auf S. 22).

[TZS+08] X. Tang, Z. Zhang, B. Sun, Y. Li, N. Li, B. Wang, X. Zhang. „Antarctic ice sheet GLIMMER model test and its simplified model on 2-dimensional ice flow“. In: *Progress in Natural Science* 18.2 (Feb. 2008), S. 173–180. DOI: 10.1016/j.pnsc.2007.07.014 (zitiert auf S. 21).

[VG04] G. J.-M. C. L. Vieli, G. H. Gudmundsson. „On estimating length fluctuations of glaciers caused by changes in climatic forcing“. In: *Journal of Geophysical Research: Earth Surface* 109.F1 (Feb. 2004). DOI: 10.1029/2003jf000027 (zitiert auf S. 22).

[WD87] O. Widlund, M Dryja. „An additive variant of the Schwarz alternating method for the case of many subregions“. In: (1987) (zitiert auf S. 57).

[WGH99] M. Weis, R. Greve, K. Hutter. „Theory of shallow ice shelves“. In: *Continuum Mechanics and Thermodynamics* 11.1 (Feb. 1999), S. 15–50. DOI: 10.1007/s001610050102 (zitiert auf S. 28).

[WMH+11] R. Winkelmann, M. A. Martin, M. Haseloff, T. Albrecht, E. Bueler, C. Khroulev, A. Levermann. „The Potsdam Parallel Ice Sheet Model (PISM-PIK) Part 1: Model description“. In: *The Cryosphere* 5 (2011), S. 715–726. DOI: 10.5194/tc-5-715-2011. URL: http://www.the-cryosphere.net/5/715/2011/tc-5-715-2011.pdf (zitiert auf S. 26–29, 54).

[WZH+03] W. Wang, H. J. Zwally, C. L. Hulbe, M. J. Siegert, I. Joughin. „Anisotropic ice flow leading to the onset of Ice Stream D, West Antarctica: numerical modelling based on the observations from Byrd Station borehole“. In: *Annals of Glaciology* 37 (2003), S. 397–403. DOI: 10.3189/172756403781815591 (zitiert auf S. 21).

[Wol14] M. Wolff. *Heaptrack - A Heap Memory Profiler For Linux*. https://milianw.de/blog/heaptrack-a-heap-memory-profiler-for-linux.html. Dez. 2014. URL: https://www.mcs.anl.gov/petsc (zitiert auf S. 37).

[YHLD17] I. Yamazaki, M. Hoemmen, P. Luszczek, J. Dongarra. „Improving performance of GMRES by reducing communication and pipelining global collectives“. In: *2017 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW)*. IEEE. 2017, S. 1118–1127. DOI: 10.1109/ipdpsw.2017.65 (zitiert auf S. 86).

Alle URLs wurden zuletzt am 13. 12. 2020 geprüft.

Erklärung

Ich versichere, diese Arbeit selbstständig verfasst zu haben. Ich habe keine anderen als die angegebenen Quellen benutzt und alle wörtlich oder sinngemäß aus anderen Werken übernommene Aussagen als solche gekennzeichnet. Weder diese Arbeit noch wesentliche Teile daraus waren bisher Gegenstand eines anderen Prüfungsverfahrens. Ich habe diese Arbeit bisher weder teilweise noch vollständig veröffentlicht. Das elektronische Exemplar stimmt mit allen eingereichten Exemplaren überein.

Ort, Datum, Unterschrift