Efficient Numerical Algorithms and Software Engineering for High Performance Computing

Effiziente numerische Algorithmen und Softwareentwicklung für hochparallele Rechensysteme

Department Informatik
Technische Fakultät
Friedrich-Alexander Universität Erlangen-Nürnberg

Abschlussbericht zum Habilitationsverfahren
und
kumulative Habilitationsschrift

vorgelegt von

Dr.-Ing. Harald Köstler

Erlangen – 2014

Als Habilitationsvorhaben genehmigt von
der Technischen Fakultät der
Universität Erlangen-Nürnberg

Tag der Einreichung:	January 23, 2014
Fachmentorat:	
	Prof. Dr. U. Rüde
	Prof. Dr.-Ing. J. Hornegger
	Prof. Dr. H.-J. Bungartz
Externe Gutachter:	
	Prof. Dr. P. Bastian
	Prof. H. De Sterck, PhD

Abstract

The field "Computational Science and Engineering" (CSE) lies in between computer science, applied mathematics, and various application fields like natural sciences, engineering, or medical science. Leading edge research in CSE requires often interdisciplinary teams consisting of specialists from each of these fields. Additionally, CSE experts are essential to bridge the gap between the traditional sciences. Currently, one of the biggest challenges in CSE, driven by the progress in computer hardware, is the efficient utilization of massively parallel architectures like Graphics Processing Units (GPUs) for smaller problems and high performance computing (HPC) clusters for larger problems. From a computer science point of view, two challenges have to be addressed. First, efficient parallel algorithms have to be developed that fit to the underlying hardware and allow the user to solve problems in reasonable time, second, HPC software has to be designed such that it can be easily ported to new HPC architectures and helps to increase the productivity of its users.

Contents

Part I.

Abschlussbericht zum Habilitationsverfahren

1. Überblick Forschung und Lehre

1.1. Forschungsarbeiten

Aufgrund der Interdisziplinarität des Faches Computational Science and Engineering sind an vielen Forschungsprojekten Kooperationspartner aus der Mathematik oder einer Anwendungswissenschaft beteiligt. Dadurch ergibt sich eine breite Streuung der besuchten Konferenzen und der gewählten Zeitschriften für die Veröffentlichungen. Obwohl mein Schwerpunkt in der Informatik anzusiedeln ist, können sowohl die Entwicklung von Modellen für verschiedene Anwendungen als auch von numerischen Methoden nicht außer Acht gelassen werden. Zur besseren Strukturierung habe ich meine Forschungstätigkeiten in verschiedene Themengebiete unterteilt.

1.1.1. Themengebiete

Bereich High Performance Computing (HPC) Ich war an der Entwicklung der Softwarekonzepte des waLBerla Frameworks zur Simulation von Multi-Physik-Problemen auf HPC Clustern beteiligt [10, 1],[30, 44], wobei ich mich hier vor allem mit GPUs [5] und heterogenen GPU-CPU Clustern [2, 3] beschäftigt habe. Methodisch gilt mein besonderes Interesse Mehrgitterverfahren zur effizienten Lösung von Gleichungssystemen, die von mir auch in waLBerla integriert und deren Skalierbarkeit auf GPU Clustern gezeigt wurde [37, 34]. Zudem habe ich mich mit hochparallelen Mehrgitterverfahren für CPU Cluster, z.B. für Anwendungen aus der Quantenchemie [11, 18], und Performance Modellierung zu Laufzeitvorhersage beschäftigt [4, 12]. Performance Modellierung oder Performance Engineering findet auf mehreren Ebenen statt. Während für HPC Cluster die Modellierung der Kommunikationszeiten zwischen den Rechenknoten wichtig ist, muss natürlich auch ein Performance Modell für einen einzelnen Rechenknoten, der eine oder mehrere CPUs oder GPUs enthalten kann, erstellt werden, um valide Modelle zu erhalten. Neben der Simulation auf Höchstleistungsrechnern, wie z.B. JUQUEEN, SuperMUC und Tsubame 2.0, ist auch die Echtzeitsimulation auf GPUs ein aktuelles Forschungsthema [14, 19],[64].

Bereich HPC Software Engineering Die Erfahrung bei der Entwicklung von HPC Software zeigt, dass die Komplexität der Modelle und damit auch der Umfang der Software in den letzten Jahren stetig steigen. Wir haben daher versucht, durch eine abstrakte Modellierung von Mehrgitteralgorithmen und automatische Codegenerierung eine Steigerung der Produktivität bei der Softwareentwicklung [24, 43] zu erreichen. Seit Anfang 2013 bin ich in das im Rahmen des DFG-Schwerpunktprogramms 1648 (Software for Exascale Computing) geförderte Projekt „Advanced Stencil-Code Engineering (ExaStencils)“ involviert. Ziel ist es dabei, eine domänenspezifische Sprache für stencil-basierte Mehrgitterverfahren zu entwickeln, die mit Hilfe von Domänenwissen, moderner Software Engineering Methoden,

Compilertechniken und -werkzeugen eine automatische, hardware- und problemspezifische Optimierung von Mehrgitteralgorithmen für Höchstleistungsrechner erlaubt. Hier können die Vorarbeiten zur Implementierung von parallelen Mehrgitterverfahren [40], zur automatischen Codegenerierung [15] und zur Mehrzieloptimierung [8] genutzt werden.

Bereich Bildverarbeitung In diesem Bereich knüpfe ich an meine Dissertation im Themenbereich effiziente Mehrgitterverfahren in der Bildverarbeitung an [59]. Performance Engineering ist auch für verschiedene Anwendungen aus der Bildverarbeitung nützlich, um realzeitfähige Algorithmen (z.B. Mehrgitterverfahren und Sparse Coding) effizient auf aktuelle Hardware zu portieren. Als Anwendungen habe ich mich mit optischem Fluss [17], High Dynamic Range Imaging [12] auf GPUs, dem Entrauschen [22, 41, 47] von medizinischen Datensätzen, sowie verschiedenen Verfahren zur Bildsegmentierung [6],[46] befasst. Derzeit arbeite ich im Rahmen eines Industrieprojektes an einer realzeitfähigen Bildregistrierung von medizinischen Datensätzen auf GPUs.

Bereich Numerik Einen gegebenen numerischen Algorithmus auf eine spezielle Hardware zu portieren reicht meist nicht aus, um ein optimales Verfahren zu erhalten. Meist ist es nötig, auch den Algorithmus selbst an die Hardware anzupassen und z.B. eine Variante zu wählen, die sich besser parallelisieren lässt. Daher habe ich auch die problemspezifische Anpassung der verschiedenen Mehrgitterkomponenten (Interpolation, Grobgitterapproximation) untersucht [20]. Daneben war ich an der Entwicklung von Lösungsmethoden für allgemeine lineare least-squares Probleme [16] und von Diskretisierungstechniken für Dipole in den Quelltermen von partiellen Differentialgleichungen [21] beteiligt. Letzteres war auch schon Gegenstand meiner früheren Forschung [9].

1.1.2. Vorträge (seit 2010)

9/2013 **Köstler, H.**; Kuckuk, S.; Gmeiner, B.; Rüde, U.: *A Generic Prototype to Benchmark Algorithms and Data Structures for Hierarchical Hybrid Grids*. International Conference on Parallel Computing - ParCo2013, Munich, Germany.

9/2013 **Pickl, K.**; Hofmann, M.; Preclik, T.; Köstler, H.; Smith, A.; Rüde, U.: *Parallel Simulations of Self-propelled Microorganisms*. International Conference on Parallel Computing - ParCo2013, Munich, Germany.

6/2013 **Lengauer, Ch.**; Köstler, H.; Apel, S.; Größlinger, A.: *Modern Software Technology for Exascale Computing*. International Supercomputing Conference (ISC'13), Leipzig, Germany.

5/2013 **Köstler, H. (invited)**: *Lattice Boltzmann simulations on heterogeneous CPU-GPU clusters*. 2nd International Symposium Computer Simulations on GPU, Freudenstadt, Germany.

4/2013 **Köstler, H. (invited)**: *Algorithm and software development for efficient multigrid methods on modern HPC systems*. Weizmann Workshop 2013 on Multilevel Computational Methods and Optimization, Rehovot, Israel.

3/2013 **Bartuschat, D.**; Köstler, H.; Rüde, U.: *Fast Multigrid Solvers Long Range Potentials.* SIAM Conference on Computational Science and Engineering, Boston, USA.

2/2013 **Bartuschat, D.**; Köstler, H.; Kluge, A.; Godenschwager, Ch.: *An anisotropic nonlinear diffusion filter for 3D CTA image processing.* SIAM Conference on Computational Science and Engineering, Boston, USA.

2/2013 **Köstler, H.**: *Sparse Solving.* 16th Copper Mountain Conference on Multigrid Methods, Copper Mountain, Colorado, USA.

12/2012 **Gmeiner, B.**; Donnert, G.; Köstler, H.: *Optimizing Opening Strategies in a Real-time Strategy Game by a Multi-objective Genetic Algorithm.* SGAI International Conference on Artificial Intelligence, Cambridge, UK.

7/2012 Feichtinger, Ch.; **Köstler, H.**: *Complex Flow Simulations using Heterogeneous Supercomputers.* 10th World Congress on Computational Mechanics (WCCM), Sao Paulo, Brazil.

5/2012 **Köstler, H.**: *Large Scale Imaging on Current Many-Core Platforms.* SIAM Conference on Imaging Science, Philadelphia, USA.

2/2012 Schornbaum, F.; Feichtinger, Ch.; **Köstler, H.**; Rüde, U.: waLBerla*: Towards an Adaptive, Dynamically Load-Balanced, Massively Parallel Lattice Boltzmann Fluid Simulation.* SIAM Conference on Parallel Processing for Scientific Computing, Savannah, Georgia, USA.

11/2011 **Köstler, H. (invited)**: *Efficient Imaging Algorithms on Many-Core Platforms.* Seminar Efficient Algorithms for Global Optimisation Methods in Computer Vision, Dagstuhl, Germany.

8/2011 **Gmeiner, B.**; Köstler, H.; Rüde, U.: *Towards real-time image processing with Hierarchical Hybrid Grids.* IDK Summer School, Pommersfelden, Germany.

7/2011 **Köstler, H. (invited)**: *Multicore and GPU Implementation of Multigrid and LBM.* International Workshop on New Algorithms and Programming Models for the Manycore Era (APMM 2011), Istanbul, Turkey.

6/2011 **Köstler, H. (invited)**: *Numerical Methods in Simulation and Imaging.* Max-Planck-Institut für biophysikalische Chemie, Göttingen, Germany.

5/2011 **Feichtinger, Ch.**; Habich, J.; Köstler, H.; Rüde, U.; Wellein, G.: waLBerla*: Heterogeneous Simulation of Particulate Flows in GPU Clusters.* 23rd International Conference on Parallel Computational Fluid Dynamics 2011 (ParCFD 2011), Barcelona, Spain.

3/2011 **Köstler, H. (invited)**: *Numerical Codes on Multi-GPU Architectures.* ASIM Workshop on Trends in Computational Science and Engineering, Garching, Germany.

3/2011 **Köstler, H.**: *A robust geometric multigrid solver within the* waLBerla *framework.* 15th Copper Mountain Conference on Multigrid Methods, Copper Mountain, Colorado, USA.

3/2011 **Stürmer, M.**; Köstler, H.; Rüde, U.: *How to Optimize Geometric Multigrid Methods on GPUs*. 15th Copper Mountain Conference on Multigrid Methods, Copper Mountain, Colorado, USA.

2/2011 Köstler, H.; **Röhrle, O.**: *A Fast GPU-based Method for Image Segmentation*. SIAM Conference on Computational Science and Enginnering (CSE11), Reno, Nevada, USA.

2/2011 **Köstler, H.**; Feichtinger, Ch.; Götz, J.; Donath, S.; Rüde, U.: *HPC Software Design for Computational Engineering Simulations*. SIAM Conference on Computational Science and Engineering (CSE11), Reno, Nevada, USA.

2/2011 **Stürmer, M.**; Rathgeber, F.; Köstler, H.: *Performance Engineering of an Orthogonal Matching Pursuit Algorithm*. Second International Workshop on New Frontiers in High-performance and Hardware-aware Computing, San Antonio, Texas, USA.

9/2010 **Köstler, H.**: *Multigrid algorithms on multi-GPU architectures*. European Multigrid Conference 2010, Ischia, Italy.

9/2010 **Ritter, D.**; Feichtinger, Ch.; Köstler, H.; Rüde, U.: *Multigrid in Quantum Chemistry on Multiple GPUs*. European Multigrid Conference 2010, Ischia, Italy.

7/2010 Köstler, H.; Ritter, D.; Feichtinger, Ch.; **Rüde, U.**: *Performance of Multigrid Solvers on GPUs*. 9th World Congress on Computational Mechanics, Minisymposium on GPUs and Modern Many-Core Processors, Sydney, Australia.

7/2010 **Köstler, H. (invited)**: *Software and Performance Engineering for numerical codes on GPU clusters*. GPU Solutions to Multiscale Problems in Science and Engineering, Harbin, China.

6/2010 **Köstler, H.**,; Stürmer, M.: *Optimized fast wavelet transform utilizing a multicore-aware framework for stencil computations*. Para 2010: State of the Art in Scientific and Parallel Computing, Reykjavik, Iceland.

5/2010 **Popa, C.**; Köstler, H.; Preclik, T.; Rüde, U.: *On Kaczmarz's Projection Iteration as a Direct Solver for Linear Least Squares Problems*. Conference on Applied Linear Algebra, Novi Sad, Serbia.

4/2010 **Köstler, H.**; Tatavarty, S.: *Accelerating image registration on GPUs*. SIAM Conference on Imaging Science (IS10), Chicago, USA.

4/2010 **Dietrich, I.**; Köstler, H.; German, R.; Rüde, U.: *Modeling Multigrid Algorithms for Variational Imaging*. 21st Australian Software Engineering Conference (ASWEC 2010), Auckland, New Zealand.

3/2010 **Tatavarty, S.**; Köstler, H. ; Rüde, U.: *Accelerating Image Registration on GPUs - A proof of concept migration of FAIR to CUDA*. Inf/Math Kolloquium der Universität Lübeck, Lübeck, Germany.

3/2010 **Köstler, H.**: *Numerical Algorithms on Multi-GPU Architectures*. 2nd International Workshop on Advances in Computational Mechanics - IWACOM-II, Yokohama, Japan.

2/2010 **Köstler, H.**; Ritter, D.; Rüde, U.; Stürmer, M.: *Multigrid for Multicore*. SIAM Conference on Parallel Processing for Scientific Computing, Seattle, USA.

1.1.3. Organisation von Konferenzen/Workshops/Minisymposia (seit 2010)

- International Workshop on New Algorithms and Programming Models for the Manycore Era (APMM), 2011.
- International Workshop on New Algorithms and Programming Models for the Manycore Era (APMM), 2013.
- Computing Frontiers, 2013.
- 2nd International Workshop on High-performance and Hardware-aware Computing (HipHaC), 2011.
- 11th International Symposium on Parallel and Distributed Computing (ISPDC), 2012.
- 12th International Symposium on Parallel and Distributed Computing (ISPDC), 2013.
- 9th International Conference on Parallel Processing and Applied Mathematics (PPAM), 2011.
- 4th Workshop on UnConventional High Performance Computing (UCHPC), 2011.
- 5th Workshop on UnConventional High Performance Computing (UCHPC), 2012.
- 6th Workshop on UnConventional High Performance Computing (UCHPC), 2013.
- 11th International Meeting on High-Performance Computing for Computational Science (VECPAR), 2014.
- 1st International Workshop on High-Performance Stencil Computations (HiStencils), 2014.

1.1.4. Forschungsprojekte (seit 2010)

- Beteiligt am Kompetenznetzwerk für Technisch-Wissenschaftliches Hoch- und Höchstleistungsrechnen (KONWIHR), z.B. im Rahmen des Projekts *walberlaMC: Entwicklung von HPC Software zur Simulation von Partikeln in Strömungen für Many-Core Systeme*
- Beteiligt am DFG/AiF Cluster *Proteinschäume in der Lebensmittelindustrie*
- Seit Anfang 2013 bin ich in das im Rahmen des DFG-Schwerpunktprogramms 1648 (Software for Exascale Computing) geförderte Projekt *Advanced Stencil-Code Engineering (ExaStencils)*[1] involviert.
- Projektleitung und Projektdurchführung bei den Industriekooperationen

[1] `www.exastencils.org`

- *Entwicklung von Rekonstruktionsprozessen bei Magnet-Resonanz-Tomographie* (Siemens Healthcare)
- *Parallele Rechnerarchitekturen zur Beschleunigung von komplexen Echtzeitprozessen* (Siemens Industry)
- *Bereitstellung und Anpassung eines GPU-basierten Poisson-Lösers* (Siemens Healthcare)
- *Entwicklung eines effizienten Spannungsmodells in Zylinderkoordinaten* (Siemens Industry)
- *Entwicklung von Simulationsmodellen für die Berechnung der Myocard-Perfusion* (Siemens Healthcare)
- *Erweiterung eines echtzeitfähigen Temperaturmodells* (Siemens Industry)
- *Portierung einer affinen Registrierung auf GPUs* (Siemens Healthcare)

1.2. Lehre

Bereits während meines Studiums und meiner Promotion konnte ich Erfahrungen bei der Betreuung von Studenten sammeln. Zudem war ich auch aktiv an der Gestaltung innovativer Lehrkonzepte beteiligt [25, 27],[67]. Um meine eigenen didaktischen und pädagogischen Fähigkeiten weiter zu verbessern, habe ich im Jahr 2011 das Zertifikat Hochschullehre vom Fortbildungszentrum Hochschullehre (FBZHL) erworben (insgesamt 88 Arbeitseinheiten). Offiziell wurde ich am 11.5.2011 als Habilitand im Fachgebiet *Informatik* an der Technischen Fakultät der Friedrich-Alexander-Universität Erlangen-Nürnberg (FAU) angenommen.

1.2.1. Lehrveranstaltungen

Seit meiner Dissertation im Jahr 2008 führe ich regelmäßig selbständig Lehrveranstaltungen durch. Dazu gehören die von mir entworfene Vorlesung mit Übungen *Advanced Programming Techniques*, die ich seit WS 09/10 vor allem für Studenten im Fach Computational Engineering (CE), aber auch im Fach Informatik anbiete. Hier geht es vor allem um ein tieferes Verständnis für die Programmiersprache C++ und für die ihr zugrunde liegenden objektorientierten Konzepte. Daneben habe ich im SS 10 in Vertretung von Prof. Ulrich Rüde die Vorlesung mit Übungen *Simulation und wissenschaftliches Rechnen 2* gehalten. Von den unterschiedlichen Seminaren, an denen ich beteiligt war, möchte ich vor allem das Seminar *Scientific Writing* herausstellen, in dem Grundlagen dafür vermittelt werden sollen, selbstständig wissenschaftliche Texte zu verfassen. Es wurde ursprünglich für Studenten der Bavarian Graduate School of Computational Engineering (BGCE) entwickelt. Die BGCE ist Teil des Elitenetzwerks Bayern und verbindet die Masterstudiengänge CE an der FAU sowie Computational Mechanics (COME) und Computational Science and Engineering (CSE) an der TU München. Seit 2004 betreue ich als Koordinator der BGCE in Erlangen die Studenten mit und beteilige mich auch an gemeinsamen Lehrveranstaltungen. Auch außerhalb Bayerns konnte ich schon Lehrerfahrungen in einwöchigen Kompaktkursen sammeln. Dazu gehören der Kurs *Efficient Multigrid Methods in Computer Vision and Medical Image Processing*, den ich 2009 an der KTH Stockholm angeboten habe und der Kurs *Introduction to OpenCL for Medical Imaging* im Jahr 2010 an der Universität zu Lübeck. Auch im Frühjahr 2014

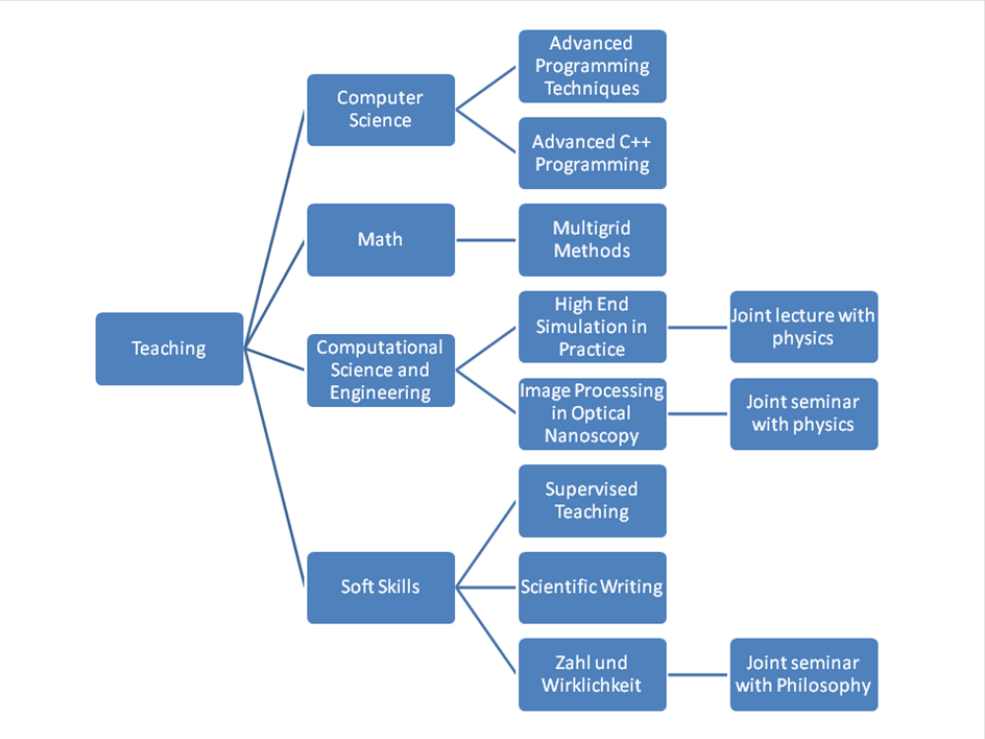


Abbildung 1.1.: Überblick über die thematische Einordnung der Lehrveranstaltungen.

werde ich wieder einen einwöchigen Kurs an der KTH Stockholm zum Thema *Computational Fluid Dynamics with the lattice Boltzmann Method* anbieten. Im Folgenden möchte ich meine Lehrtätigkeiten seit dem Sommersemester 2011 im Detail auflisten.

Daten Lehrbericht 2011

SS 11: Im SS11 habe ich zusammen mit dem Physiker M. Bannerman die interdisziplinäre Vorlesung mit Übungen *High End Simulation in Practice* ins Leben gerufen (Globalindikator Evaluation 1.28). Die Idee war hier, eine Veranstaltung zu kreieren, die im Kernbereich von CSE liegt und die auf den Vorkenntnissen aufbaut, die die Studenten in meiner informatiknahen Vorlesung Advanced Programming Techniques erwerben. Ausgehend von einem physikalischen Modell und dessen mathematischer Beschreibung sollen die Studenten in die Lage versetzt werden, numerische Algorithmen für bestimmte Simulationsaufgaben zu verstehen und in effizienten und parallelen Code umzusetzen. Konkret geht es dabei um Partikelsimulation in OpenCL auf moderner Grafikhardware. Daneben war ich noch an drei Seminaren beteiligt, die alle bereits mehrfach stattfanden: Zusammen mit Klaus Iglberger am Seminar *Advanced C++ Programming*, das sich ebenfalls an Studenten richtet, die ihre Kenntnisse aus Advanced Programming Techniques vertiefen wollen, zusammen mit dem Physiker Gerald Donnert an *Image Processing in Optical Nanoscopy*, das neben Studenten in CE und Informatik auch besonders Studenten im Masterprogramm Advanced Optical Technologies (MAOT) angesprochen hat. Hier ging es inhaltlich vor allem um die physika-

lischen Grundlagen und den Aufbau von neuartigen, hochauflösenden Mikroskopen und das Verbessern der Bildqualität der resultierenden Aufnahmen durch Image Deblurring. Zudem habe ich zusammen mit dem Philosophen/Theologen Benjamin Gleede das Soft Skills Seminar *Zahl und Wirklichkeit* betreut, in dem versucht wurde, einen Abriss vom Wandel des Zahlverständnisses in Philosophie und Mathematik von der Antike bis hin zur heutigen Zeit vorzunehmen.

WS 11/12: Wie auch in den vorangegangenen Wintersemestern bot ich die Vorlesung mit Übungen *Advanced Programming Techniques* an (Globalindikator Evaluation V/Ü 1.3/1.4; Platz 4 von 28 in der Kategorie ÜW5). Zudem war ich auch wie schon früher in das Soft Skills Angebot *Supervised Teaching* involviert. Hierbei handelt es sich um eine Lehrveranstaltung speziell für Studierende der BGCE, die meist im Rahmen einer Übungsleitertätigkeit von erfahrenen Dozenten begleitet werden. Dazu gehören auch der Besuch eines eintägigen Didaktikseminars und ein individuelles Coaching, das von einer professionellen Trainerin durchgeführt wird.

Daten Lehrbericht 2012

SS 12: Auch im SS12 fand die interdisziplinäre Vorlesung mit Übungen *High End Simulation in Practice* statt (Globalindikator Evaluation V/Ü 1.26/1.31), diesmal mit Unterstützung von Severin Strobl. Des Weiteren vertrat ich Prof. Christoph Pflaum in der Vorlesung mit Übungen *Multigrid* (Globalindikator Evaluation 2.18). Diese ist in der angewandten Mathematik (Numerik) angesiedelt und behandelt mit Multigrid eines der wichtigsten und effizientesten iterativen Lösungsverfahren für sehr große, dünn besetzte Gleichungssysteme. Diese treten z.B. nach der Diskretisierung von partiellen Differentialgleichungen auf.

WS 12/13: Im WS 12/13 wurde die Vorlesung mit Übungen *Advanced Programming Techniques* von mir überarbeitet (Globalindikator Evaluation V/Ü 1.32/1.62; Platz 2 von 40 in der Kategorie VW10), da ein neuer C++ Sprachstandard eingeführt wurde. Zudem wird nun etwas mehr Zeit auf die Vermittlung von Konzepten zur parallelen Programmierung verwendet, die nun besser und direkter unterstützt werden. Vor eine besondere Herausforderung wurde ich durch die gewachsene Anzahl an Hörern gestellt, die sich im Vergleich zu den Vorjahren auf etwa 75 verdoppelt hatte. Dies bedeutete für das ganze Team vor allem bei den Programmierübungen einen deutlichen Mehraufwand.

Daten Lehrbericht 2013

SS 13: Wiederum im SS13 fand die interdisziplinäre Vorlesung mit Übungen *High End Simulation in Practice* statt (Globalindikator Evaluation V/Ü 1.41/1.13), auch dieses Mal mit Unterstützung von Severin Strobl. Ausserdem habe ich zusammen mit dem Physiker Gerald Donnert wieder die Vorlesung *Image Processing in Optical Nanoscopy* angeboten.

WS 13/14: Im WS 13/14 wurde die Vorlesung mit Übungen *Advanced Programming Techniques* von mir zum ersten Mal zweigeteilt, zum einen aufgrund der hohen Hörerzahl und

zum anderen aufgrund der Heterogenität der Vorkenntnisse der Studenten. Die Version *Advanced Programming Techniques für CE* (2 Stunden Vorlesung und 4 Stunden Übung pro Woche) richtet sich nun vor allem an (vorwiegend ausländische) Studenten im Masterstudiengang CE, die nicht den Bachelorstudiengang CE in Erlangen absolviert haben. Hier liegen die Schwerpunkte auf der Vermittlung von grundlegenderen Informatikkenntnissen. An alle anderen Studenten richtet sich die Version *Advanced Programming Techniques* (4 Stunden Vorlesung und 2 Stunden Übung pro Woche), die tiefergehende Konzepte der objektorientierten und parallelen Programmierung vermitteln soll. In der dazugehörigen Übung wird nun in Gruppenarbeit ein größeres Softwareprojekt fast über die gesamte Dauer des Semesters durchgeführt.

1.2.2. Betreuung studentischer Arbeiten (seit 2010)

Einen Überblick über die von mir betreuten Abschlussarbeiten findet sich im gesonderten Literaturverzeichnis [69]-[74]. Daneben habe ich auch bei der Betreuung von Doktoranden mitgewirkt [102, 107] und bin Mentor der Erlangen Graduate School in Advanced Optical Technologies (SAOT). In dieser Funktion unterstütze ich Doktoranden bei fachlichen Fragen oder gebe Hilfestellung bei organisatorischen Problemen.

1.3. Organisatorische Tätigkeiten

Studienfachberatung Master Computational Engineering und Elitestudiengang BGCE Das Aufgabenfeld umfasst unter anderem

- Organisation des Studiengangs
- Mitglied der Zulassungskommission
- Mitglied der Studienkommission
- Erstellen und Pflegen der individuellen Studienpläne
- Beratung der Studenten

COSSE-Koordinator Das Erasmus-Mundus-Programm *Computer Simulations for Science and Engineering (COSSE)* ist ein internationales Doppelabschlussprogramm, bei dem es die FAU zusammen mit den Universitäten KTH Stockholm, TU Berlin und TU Delft den Studenten ermöglicht, an zwei der vier Universitäten innerhalb von zwei Jahren einen Masterabschluss zu erlangen[2]. Hierbei bin ich beteiligt an

- Auswahl der COSSE Studenten
- Betreuung der COSSE Studenten
- Organisation von COSSE Workshops
- Betreuung spezieller Lehrveranstaltungen

[2] `http://www.kth.se/en/studies/master/em/cosse`

Erasmus-Koordinator des Departments Informatik

- Auswahl der Erasmus-Studenten (Incoming and Outgoing)
- Beratung über Studienmöglichkeiten im Ausland
- Beratung von Austauschstudenten an der FAU
- Genehmigung von Learning Agreements

Mitgliedschaften

- CUDA Research Center an der FAU
- SIAM (Society for Industrial and Applied Mathematics)
- GAMM Fachausschuss *Computational Science and Engineering*
- GAMM Fachausschuss *Mathematical Signal and Image Processing*
- Zentralinstitut für Scientific Computing (ZISC), FAU
- SAOT – Erlangen Graduate School in Advanced Optical Technologies, FAU
- Elitenetzwerk Bayern – Bavarian Graduate School of Computational Engineering (BGCE)

Teil II.

Kumulative Habilitationsschrift

2. Overview own Publications

The main focus of my research lies on development and implementation of efficient, parallel numerical algorithms for applications in computational science and engineering (CSE) on HPC systems. In general, it is often not enough to take an existing numerical algorithm and simply implement it on a certain hardware architecture. To obtain optimal efficiency and scalability one usually has to adapt the algorithm for example to enable a better parallelization. One class of methods in CSE for the iterative solution of large (non)linear systems are multigrid methods. They play an important role, since they can reach an optimal complexity that grows linearly with the problem size.

2.1. Publications during Master and PhD Thesis (2003-2008)

My first publications involving multigrid algorithms stem from my master thesis in computer science [57], where I investigated the accurate treatment of singularities in the source term of partial differential equations (PDEs) within a multigrid solver [38]. Later, I added a formal derivation of the discretization error expansion [9]. One application for this technique is found in EEG source reconstruction, where the source term consists of one or more dipoles [21],[51, 26],[70].

During my PhD thesis I was mainly concerned with multigrid methods for applications in medical imaging and computer vision [59] like geometric multigrid solvers for optical flow [7, 17],[55, 61],[23, 52], video compression [65, 63],[50, 49], sinogram interpolation [39], image denoising [42, 41], image registration [17], image segmentation [6], or algebraic multigrid solvers for image reconstruction [35, 36, 45],[62].

Besides multigrid I also worked on orthogonal matching pursuit [47] that is used in context of sparse representations of signals, e.g., for image denoising [53],[22].

2.2. Publications after PhD Thesis (since 2008)

GPU computing After my PhD I developed parallel algorithms especially for GPUs and massively parallel HPC clusters. One of the first applications was high dynamic range compression on GPU [12]. In addition to that, I supervised master theses on nonrigid image registration [77] and 3D anisotropic diffusion [73] on GPU. Further, I was involved in a larger industry project to achieve real-time simulations of temperature in hot rolling rolls and the heat-induced elastic deformations of the rolls on GPU [19],[69, 76]. Another publication deals with fast wavelet transform on GPU [48].

waLBerla framework In order to combine GPU and large clusters, I provided a first GPU support within the waLBerla (widely applicable lattice Boltzmann solver from Erlangen)

multi-physics simulation framework. GPU support was then fully integrated in waLBerla and extended to heterogeneous GPU-CPU clusters during a PhD thesis [2, 3]. I am currently leading the waLBerla group at the chair of system simulation. Chapter 4 describes waLBerla in more detail, I primarily contributed to its software design concepts [1, 10], implemented a massively parallel multigrid solver in waLBerla [37, 34], helped with adding OpenCL support for LBM [5], and co-supervised a master thesis on free surface LBM on GPU [78]. Besides GPU clusters waLBerla also performs well on current CPU clusters, e.g., to do large scale blood flow simulations based on LBM [31, 30], or large particulate flow simulations, e.g., to model self-propelled microorganisms in a fluid [44],[72]. For the particle simulation, the rigid body physics engine *pe* is used. It was also developed at our chair and can be fully coupled to waLBerla. Resulting from a student project for my lecture *High End Simulation in Practice*, an interactive particle simulation on GPU was implemented. It can be controlled via the Microsoft Kinect device [14]. This work was extended to enable interactions with waLBerla and *pe* simulations, which run on HPC clusters [74],[64]. In [16], an iterative linear least squares solver based on an extended Kaczmarz method for rigid multibody dynamics was proposed. I also supervised a thesis to solve the 3D bidomain equation in waLBerla [71].

Efficient and scalable multigrid algorithms As already mentioned, due to the changing computer architectures, new parallel numerical algorithms are required. I implemented adapted multigrid components, e.g., to be able to handle highly jumping coefficients [20] within geometric multigrid. While for standard geometric multigrid we deal with structured grids, hybrid hierarchical grids (HHG) are more flexible, since they work on an unstructured coarse grid that is refined regularly [54]. This makes it possible to apply efficient data structures and memory accesses, and thus achieve a very efficient and highly scalable multigrid solver [40, 29, 32]. Based on performance models the runtime on HPC clusters can be predicted [4]. One possible application field for multigrid is quantum chemistry [11, 18]. As a basic example we provided a simple geometric and algebraic multigrid solver for teaching multigrid [66, 60].

Increasing productivity in HPC software development There is one drawback of multigrid: Its components and parameters are highly problem-dependent. Thus, for a new application, or even for the same application running on a different architecture, a completely different implementation may be necessary. Furthermore, our experience with HPC software development shows that the complexity of the models, and thus of the software, grows over the years, due to the higher capabilities of the hardware. It is therefore necessary to address the productivity of software development. One approach is to provide the algorithm designer a graphical front-end to exchange multigrid components [24], another to automatically generate code from a high level C++ description of multigrid components for different platforms [43],[15]. One can also learn multigrid parameters from previous simulation runs [33]. In chapter 6 we will outline one promising approach for an automatic code generation for CSE applications to increase productivity.

Applications Other applications I have considered over the last years are blood flow estimation based on angiographic images sequences [68], segmentation of muscle fibres from

skeletal muscle cross-section images [46],[75], and optimization of build orders in a strategy game using a multi-objective genetic algorithm [8],[28]. In the latter, a discrete-event simulation is performed rather than a continuous simulation as for most CSE problems.

Education and teaching in CSE At FAU, CSE is called Computational Engineering (CE) and has a strong focus on computer science and engineering, because it is located at the engineering department. Since 2003, I am involved in the CE program. I am one of the coordinators of the Bavarian Graduate School of Computational Engineering[1] (BGCE), an elite program that is part of the Elitenetzwerk Bayern. BGCE builds upon the regular CE master program and offers additional soft skills courses, specific compact courses held by international experts, regular meetings, and an elite project [25, 27]. Since 2009, I am also the study adviser for the regular CE master program. Here, an interesting challenge is to work with the international students. About 80 % of the 40-50 students per year come from outside of Germany, most of them from Asia. I was also involved in a study on the behavior of the students when working in international teams [67].

2.3. Publications contributing to Habilitation Thesis

The following publications contribute to this thesis and are centered mainly around software development for CSE applications and parallel multigrid methods. They are listed in the order of occurrence in the text and attached at the end of the Habilitation thesis.

1. *The CSE software challenge – covering the complete stack* [10]: gives an overview over software development for CSE applications with waLBerla as one exemplary CSE software framework. I wrote most of the text (see chapter 3).

2. waLBerla*: HPC Software Design for Computational Engineering Simulations* [1]: first journal publication introducing the software design concepts within waLBerla. I contributed to the text, especially to the software quality section (see section 4.2).

3. *A Framework for Hybrid Parallel Flow Simulations with a Trillion Cells in Complex Geometries* [30]: shows our latest LBM scaling results with waLBerla on two of the TOP 10 supercomputers in the world, JUQUEEN and SuperMUC. I supervised the work (see section 4.2).

4. *A Flexible Patch-Based Lattice Boltzmann Parallelization Approach for Heterogeneous GPU-CPU Clusters* [2]: introduces software design concepts for porting waLBerla to heterogeneous GPU-CPU Clusters. I contributed to to the ideas for these concepts (see section 4.2).

5. *Performance Modeling and Analysis of Heterogeneous Lattice Boltzmann Simulations on CPU-GPU Clusters* [3]: here excellent LBM scaling results on Tsubame 2.0 up to more than 1000 GPUs are shown. I contributed to the text (see section 4.2).

6. *Parallel Simulations of Self-propelled Microorganisms* [44]: shows scaling results on SuperMUC and JUQUEEN for coupled waLBerla and *pe* swarm simulations. I supervised the work (see section 4.3).

[1] `http://www.bgsce.de/`

7. *Interactive particle dynamics using OpenCL and Kinect* [14]: emerged from one of my lectures and a subsequent student project. I contributed to the text (see section 4.3).

8. *A Framework for Interactive Physical Simulations on Remote HPC Clusters* [64]: introduces *VIPER*, a computational steering framework that can be coupled to waLBerla and *pe* and enables interactive simulations. I supervised the work (see section 4.3).

9. *A Geometric Multigrid Solver on Tsubame 2.0* [34]: shows multigrid scaling results on Tsubame 2.0 up to more than 1000 GPUs. I implemented the multigrid solver within waLBerla, performed the experiments, and wrote most of the text (see section 5.2).

10. *Performance Engineering to achieve Real-time High Dynamic Range Imaging* [12]: a highly tuned multigrid solver on GPU is developed and its runtime is predicted based on a performance model. I did the original implementation on GPU for an industry project, contributed to the performance model, especially the LFA prediction, and the text (see section 5.2).

11. *Parallel multigrid on hierarchical hybrid grids: a performance study on current high performance computing clusters* [4]: shows multigrid scaling results on the HPC clusters JUGENE and lima and includes a performance model that is able to predict runtimes on large clusters. I contributed to the text and the performance model (see section 5.3).

12. *A Generic Prototype to Benchmark Algorithms and Data Structures for Hierarchical Hybrid Grids* [40]: shows multigrid scaling results on JUQUEEN up to more than 450.000 cores. I supervised the work (see section 5.3).

13. *A practical framework for the construction of prolongation operators for multigrid based on canonical basis functions*[20]: deals with advanced multigrid components like matrix-dependent interpolation for geometric multigrid. I implemented the multigrid solver, performed parallel experiments, and contributed to the text (see section 5.4).

14. *Modeling Multigrid Algorithms for Variational Imaging* [24]: presents a graphical front-end based on UML to describe multigrid components. I provided the multigrid knowledge, a basic multigrid implementation and contributed to the text (see section 5.5).

15. *Towards a Performance-portable Description of Geometric Multigrid Algorithms using a Domain-specific Language* [15]: shows how one can automatically generate CPU and GPU code for multigrid components from a high level C++ description. I provided the multigrid knowledge, a basic multigrid implementation and contributed to the text (see section 5.5).

16. *A parallel multigrid accelerated Poisson solver for ab initio molecular dynamics applications* [11]: shows how multigrid can accelerate ab initio molecular dynamics simulations. I wrote most of the text and integrated the parallel geometric multigrid solver into the Python software RSDFT[2] (see section 6.1.2).

[2] `http://www.rsdft.org`

17. Section 6.2 contains original research that is not yet published. I have implemented a prototype in Scala[3] for generating CPU and GPU code from an abstract problem description of a CSE application.

18. *A Multi-objective Genetic Algorithm for Build Order Optimization in StarCraft II* [8]: contains an example for a discrete-event simulation and a heuristic optimization application. I contributed to the implementation, did the parallelization, and wrote most of the text (see section 6.2.2).

[3] `http://www.scala-lang.org`

3. The CSE Software Challenge

This chapter summarizes and complements the article *The CSE software challenge – covering the complete stack*[10].

3.1. Computational Science and Engineering

Computational Science and Engineering (CSE) is an interdisciplinary field that at its core develops computer simulations to solve complex (physical) problems. Often, simulation is considered to be a third pillar in science beside theory and experiment. More general, typical CSE applications try to extract information based on a model out of given data purely relying on computations. Examples are weather and climate prediction, aircraft design, conceptual development of medical devices, computation of molecular structures, or understanding of complex biological phenomena. Thus, CSE is an own discipline and more than the sum of different fields. It lies in between computer science, applied mathematics, and application fields like engineering or natural sciences (see Figure 3.1). In order to establish the field CSE it is important to be clear about its scope and unique and distinguishing methods compared to classical fields. The SIAM Working Group on CSE Education[1] gives a nice and comprehensive overview over CSE. In Germany, the GAMM activity group on Computational Science and Engineering (CSE)[2] was launched recently as one organ for CSE, and more and more study courses in CSE have been offered by German universities in the last years.

The central research problem in CSE is to develop efficient algorithms for the various applications. Typically, these (numerical) algorithms tend to be complex and their understanding and design require deep mathematical knowledge that only well–trained experts have. However, the algorithmic structures in CSE applications follow often certain computational patterns [79]. These are classified as

- dense matrix,
- sparse matrix,
- spectral methods,
- particle methods,
- structured grids,
- unstructured grids, and

[1] `http://www.siam.org/students/resources/report.php`
[2] `http://www.uni-stuttgart.de/gamm/fa-cse`

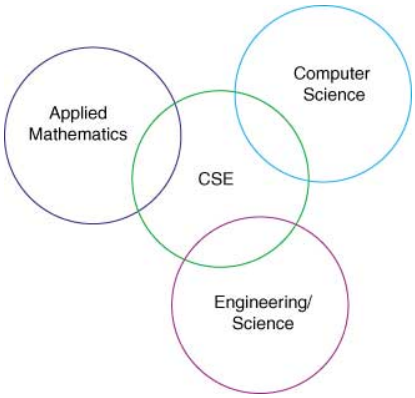


Figure 3.1.: The CSE discipline lies in between computer science, applied mathematics, and applications fields.

- Monte Carlo methods.

Real-life examples will often require a combination and coupling of several models and thus several of these patterns.

Having the goal to develop efficient algorithms in mind, one can provide a more detailed picture of the most relevant parts from the related fields for CSE in Figure 3.2. Note that in math also optimization and stochastics are becoming more and more important. Within computer science, we roughly distinguish between *high-performance computing (HPC)*, *high-throughput computing (HTC)*, and *many-task computing (MTC)*. HTC typically focuses on grid computing, where many (independent) tasks are running over a long period of time on a lot of compute resources. MTC also deals with both independent or dependent tasks on a lot of compute resources running over a shorter period of time, where data is usually exchanged via file system operations. In CSE mainly HPC is considered. Here, in most (physical) simulations tightly coupled parallel tasks run over a short period of time. In that case it becomes crucial to use parallel algorithms with optimal complexity and implement them as efficiently as possible on current computer architectures.

Depending on the hardware, one can distinguish *real-time simulations* and *large-scale simulations*. For the first kind, often accelerators like GPUs are used, and computational steering, i.e., a direct interaction between user and simulation becomes possible. For many realistic, especially multi-physics and multi-scale simulations, one requires tremendous computational resources, e.g., huge data has to be processed, and therefore large-scale simulations must be done on HPC clusters. In the TOP 500 list[3] one finds the currently largest running machines in the world. In November 2013 the biggest machine, Tianhe-2[4] (MilkyWay-2), was located at the National Super Computer Center in Guangzhou, China. It is equipped with Intel Xeon E5-2692 processors and Intel Xeon Phi 31S1P accelerators, summing up to overall 3.12 million compute cores, and a peak performance of 54.9 PFLOP/s.

[3] `http://www.top500.org`
[4] `http://www.top500.org/system/177999`

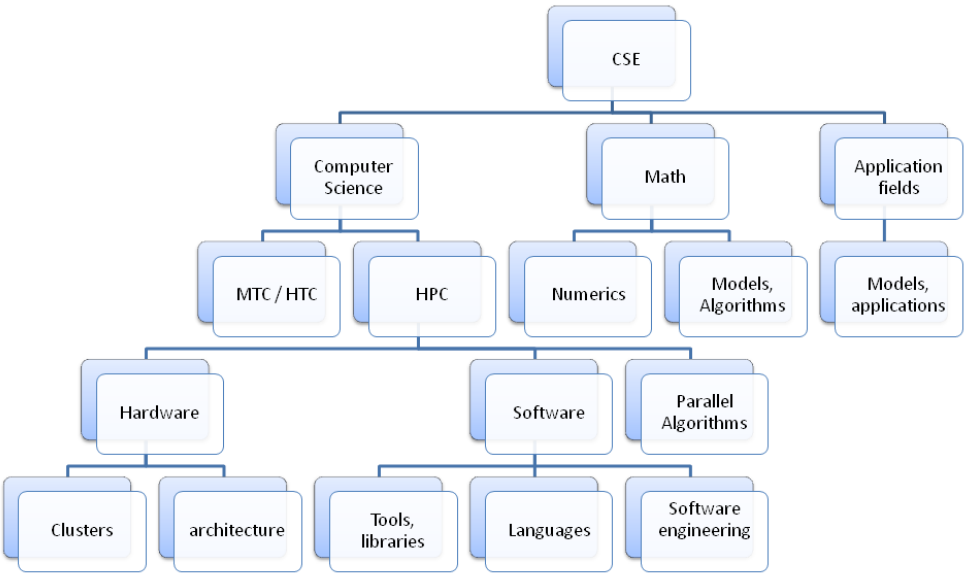


Figure 3.2.: Most relevant parts for CSE from computer science, applied math, and applications fields.

3.2. Software Development in CSE

Developing software for applications in CSE is becoming more and more challenging because of many reasons. First, with the continued increase in available compute power and improved algorithms, the (physical) models of interest become more and more complex. Furthermore, several models may have to be combined from different fields. This includes examples like fluid-structure interaction, or simulations involving effects on different time and length scales. Generally, more detailed theoretical models require improved numerical methods and faster algorithms. Thus, the complexity of the computational algorithms and data structures is increasing radically.

From the implementation point of view, large amounts of data have to be processed in order to obtain physically meaningful and sufficiently accurate results. A challenge is here to post-process the obtained results on larger HPC clusters, simply because of the huge amount of data that is produced. One way is to analyze the results during the simulation, another way is to store only part of the simulation data. Parallel algorithms have to be developed that fit to the underlying parallel hardware, and tools are necessary to analyze and tune the code. With the current trend to multicore technology, it is foreseeable that the number of processors will continue to rise sharply. This will additionally drive the complexity of CSE software upwards.

In order to pinpoint the challenges and requirements for software development in CSE,

Figure 3.3 depicts the schematic workflow for a prototypical CSE application. At the first stage an expert from the application field defines the problem, e.g., a doctor has a patient with an aneurysm and wants to estimate the risk of rupturing in order to plan further treatment. With data from modern techniques like computed tomography, digital subtraction angiography, or magnetic resonance imaging, and a suitable physical model, a simulation of the blood flow through the vessels of the patient can provide information about hemodynamic values, like shear stress or pressure, at the aneurysm. The expert from the application field can interpret the simulation results and then draw the necessary conclusions to solve his problem. The CSE expert usually starts with a formal description of the problem given by, e.g., a partial differential equation (PDE) or an optimization problem. After that, often in cooperation with a mathematician, a formal solution process is derived, e.g., the PDE is discretized and a suitable numerical solver is chosen. Finally, a suitable (parallel) algorithm that fits to the target HPC hardware is developed and implemented.

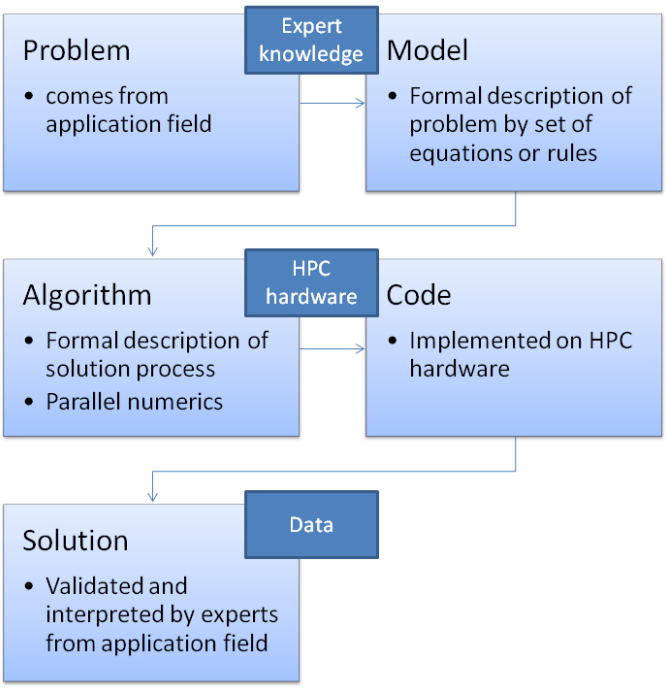


Figure 3.3.: Schematic workflow for a prototypical CSE application.

This approach for writing software is specific for the CSE field. In order to reduce the required time and cost both for the developer and later also for the user, it is reasonable to think about software quality factors (see Figure 3.4).

For CSE applications, special attention must usually be paid to *performance* because many applications have constraints on runtime or memory consumption. They quickly become infeasible, unless an efficient implementation on current HPC systems is available. Note that also power costs can become an issue. However, the development of efficient parallel CSE software is difficult since the underlying hardware evolves quickly, typically much faster than the software. Maximal performance can be achieved only on high-end clusters and the latest

hardware architectures, which become obsolete after only few years. To achieve good performance with reasonable effort, one can practice structured performance engineering. This means, that usually based on performance models, the runtime of a certain implementation is predicted, and compared to the measured performance. Supported by performance analysis tools, bottlenecks can be detected, and then they can be resolved either by optimizing the code manually, guiding the compiler to do so, or using auto-tuning tools. In order to write optimized code, not only the algorithm and hardware itself must be considered, but also the used computing language, compiler, operating system, and thus the whole software environment.

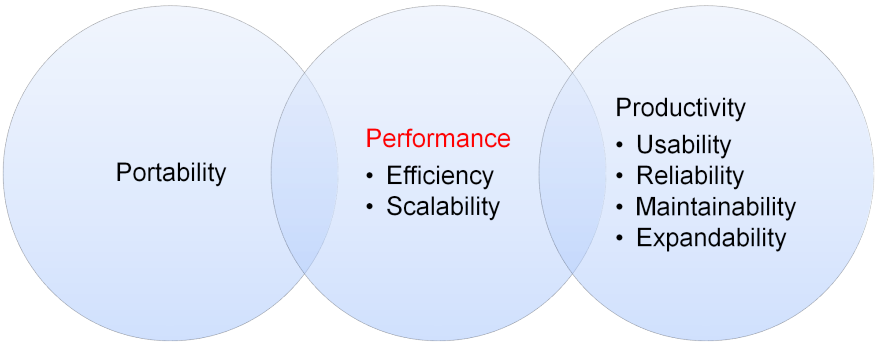


Figure 3.4.: Software quality factors.

In CSE *portability* ensures that the software does not have to be completely rewritten for the next generation of HPC clusters or accelerator architecture. Unfortunately, so far often different programming languages and paradigms are required within one code, e.g., C++ for a standard CPU and CUDA for a GPU. Besides functionality, also non-functional parameters, such as run-time efficiency and parallel scalability must be portable. In full generality, this may be difficult or impossible to achieve, so that hardware-aware re-implementations become necessary. To limit the need for such re-implementations, the software architecture must be modular also with respect to these performance aspects. CSE software may use a special *kernel*-layer, where performance critical data structures and algorithms are suitably encapsulated and can be easily replaced depending on the hardware under consideration.

Productivity is important in all software projects. For CSE applications, the user often expects support for visualizing the results or an easy-to-use interface to change, e.g., input parameters. It should be possible to run many simulation setups for parameter studies or optimization tasks. Especially for physical simulations *reliability* is essential, if, e.g., a real crash test is replaced by a simulation it has to be ensured that the simulation result is correct. A modular software design is also crucial to be able to *maintain* larger simulation codes. Since the application experts learn from experiments and simulations, how the underlying

models can be refined, it should also be easily possible to *extend* the simulation software and add new models and algorithms without major changes of the code. For many complex applications it is also necessary to combine different frameworks, or to make use of external libraries and thus well-defined, clear interfaces must be provided.

Since CSE is a cross-disciplinary field, the software must be easy to understand, easy to use, and easy to extend also for non-specialists in computer science. This enables efficient teamwork that is necessary to handle the growing complexity of the applications.

Summarizing, CSE software has to be designed such that

- it fulfills its purpose to create information out of given data based on a model using simulations,
- it can be used and extended by people from different fields,
- it runs efficiently on HPC systems.

In the following, waLBerla will be described as an example for a larger CSE software framework, multigrid methods are shown to be one of the most efficient and highly scalable numerical solvers, and a new trend in CSE to generate and optimize software automatically will be discussed.

4. waLBerla Framework

This chapter summarizes my contributions to the waLBerla framework [1]. Mainly, I was involved in providing GPU [2, 3, 5] and multigrid support [34, 37] for waLBerla. Since 2012, I am the manager of the waLBerla group and concerned with coordinating the integration of new algorithms and applications into the framework [30, 44].

4.1. Brief waLBerla History

One approach to develop software for CSE applications is to combine fast, specific, hardware-dependent, shared-memory parallel kernels with a general, easy-to-use, simple, extendable, distributed-memory or hybrid parallel framework.

A prototype implementation that follows this paradigm is named waLBerla (widely applicable lattice Boltzmann from Erlangen). waLBerla is a state-of-the-art C++ framework for multi-physics simulations centered around the lattice Boltzmann method (LBM) [95]. A brief history of its development is shown in Figure 4.1, Table 4.1 lists the specifications of the depicted HPC clusters. HRLB-II[1], SuperMUC[2], JUGENE[3], and JUQUEEN[4] are CPU-based HPC clusters located at German supercomputing sites. Tsubame 2.0[5] is a multi-GPU cluster, which is besides CPUs also equipped with 3 NVIDIA Tesla M2050 per node. The GPUs deliver most of its performance (around 2.2 PFLOP/s). The development started 2006 with four former PhD students, S. Donath [99], Ch. Feichtinger [102], J. Götz [109], and K. Iglberger [118]. All of them required massively parallel fluid simulations based on LBM for their projects, e.g., for fluid in arbitrary geometries to simulate blood flow, for fluid-structure interaction to simulate moving particles in fluid, for simulating fluids having a free surface like a rising bubble, or to simulate ionized fluid. The particles in the fluid were modeled via rigid bodies, and to solve electrostatics a multigrid method was integrated. Since the data structures and algorithms differ for rigid body dynamics and fluid dynamics, the rigid body dynamics simulation was realized in the separate software framework *pe* (physcis engine) [120] from the beginning, which is closely coupled to waLBerla in order to enable fluid-structure interaction. Already in the first technical reports about the waLBerla software concepts [110] and scalability [103], the main focus was on performance. In 2008, waLBerla ran on the HLRB-II Altix 4700 cluster [119] using 8192 cores and simulating $3.3 \cdot 10^{10}$ fluid cell with LBM. Here, a performance of more than 18 GFLUPS (billion

[1] http://www.lrz.de/services/compute/museum/hlrb2
[2] http://www.lrz.de/services/compute/supermuc
[3] http://www.fz-juelich.de/ias/jsc/EN/Expertise/Supercomputers/JUGENE/JUGENE_node.html
[4] http://www.fz-juelich.de/ias/jsc/EN/Expertise/Supercomputers/JUQUEEN/JUQUEEN_node.html
[5] http://www.gsic.titech.ac.jp/en/tsubame2

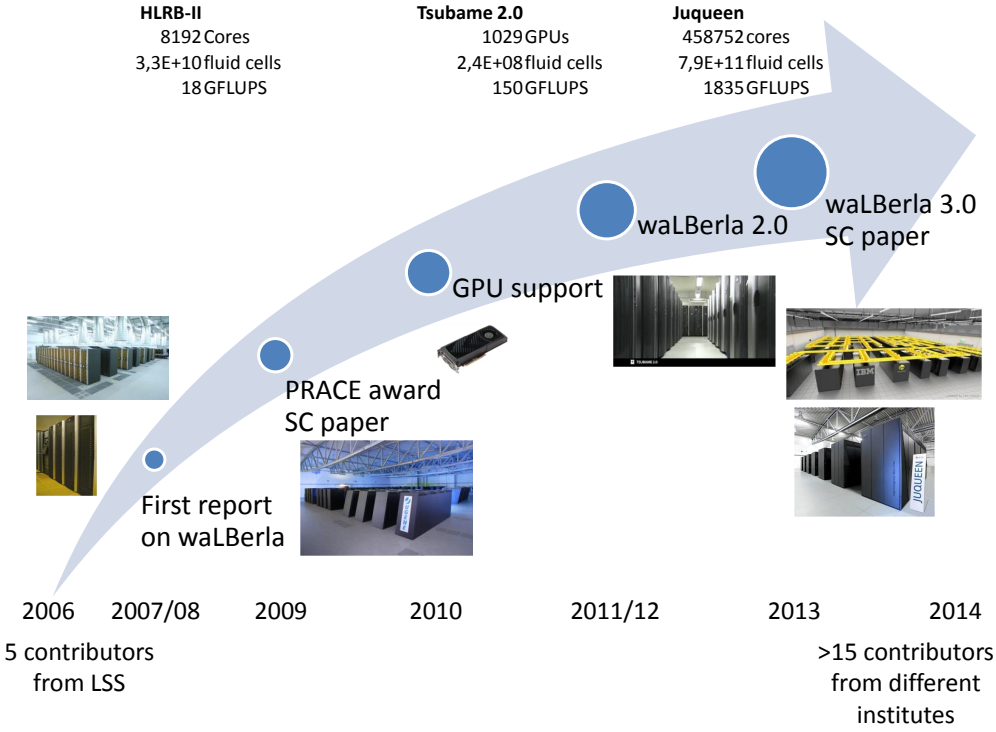


Figure 4.1.: Important milestones in the waLBerla history including performance on different HPC clusters. While at the beginning, the software was mainly developed at the chair for system simulation (LSS) in Erlangen, there are now contributors from different universities and companies.

fluid lattice updates per second) was achieved. A paper about massively parallel granular flow simulations [121] was submitted to the International Supercomputing Conference (ISC) in 2010. It got the PRACE (Partnership for Advanced Computing in Europe) award. In the same year a paper was presented at the International Conference for High Performance Computing, Networking, Storage and Analysis conference (SC10) [112]. Here, numerical experiments for the segregation of suspensions of particles of different density with up to 264 million particles and 150 billion LBM cells were performed on up to 294.912 cores of the BlueGene/P system JUGENE. The first journal article dealing with the waLBerla software design was published in 2011 [1]. At that time, a major code revision was undertaken mainly by Ch. Feichtinger, on the one hand to support heterogeneous CPU-GPU clusters, and on the other hand to improve maintainability of waLBerla, which faced a growing user and contributor base over the years. With waLBerla 2.0, we were able to obtain 150 GFLUPS on 1029 GPUs of Tsubame 2.0 on a computational domain consisting of $2.4 \cdot 10^8$ fluid cells. In order to support adaptive grid refinement within waLBerla and to further increase productivity, we have now almost completed the next major code revision waLBerla 3.0, where the three core developers are M. Bauer, Ch. Godenschwager, and F. Schornbaum. A first publication was already accepted at SC13 [30]. Here, LBM simulations

Table 4.1.: Specifications for HPC clusters, where waLBerla was tested on. Besides the total number of available compute cores, the theoretical peak performance (floating point operations per second) is provided.

Name	Location	Architecture	No. of cores	Peak Performance
JUQUEEN	Jülich, Germany	BlueGene/Q	458,752	5.9 PFLOP/s
SuperMUC	Munich, Germany	IBM System x iDataPlex	147,456	3.2 PFLOP/s
Tsubame 2.0	Tokyo, Japan	HP ProLiant, NVIDIA	4224 GPUs	2.4 PFLOP/s
JUGENE	Jülich, Germany	BlueGene/P	294,912	1.0 PFLOP/s
HLRB-II	Munich, Germany	SGI Altix 4700	9,728	56 TFLOP/s

were performed on two of the recent TOP 10 supercomputers within the TOP 500 list, SuperMUC and JUQUEEN. Our largest simulation with $7.9 \cdot 10^{11}$ fluid cells achieved 1835 GFLUPS on 458.752 cores of the BlueGene/Q system JUQUEEN. Next, we will develop waLBerla into open-source software in February 2014[6]. Currently, more than 15 people from different universities and companies are contributing to waLBerla and we expect this number to increase steeply with adding additional CSE applications to the framework in the coming years.

4.2. Software Design Concepts

In the following some of the important design concepts for the different waLBerla versions are highlighted. A comprehensive summary of the software features of the first waLBerla implementation is given in [1].

4.2.1. Domain Partitioning

All simulations are performed in a 3D physical domain. At first, the simulation domain was a regular patch (see Figure 4.2), which was subdivided into several regular blocks consisting of cells. Within one block it was possible to have different kinds of data, e.g. a list of particles or Cartesian data for the fluid. For being able to simulate complex geometries with obstacles, blocks were also allowed to be empty, i.e., no memory was allocated for the block data.

In order to support adaptive grid refinement in waLBerla 3.0, the patch concept was changed. Now, first the simulation domain is divided into blocks (see Figure 4.3) and then these blocks can be further subdivided in 3D into eight smaller blocks. The current domain partitioning represents a forest of octrees [92], where each initial block is the root of one octree. Note that no concepts and structures typically associated with trees like father-child connections or inner nodes are used. Each block only knows all of its direct neighbors, therefore this approach is well-suited for parallelization. Grid refinement and load balancing is the focus of our core developer F. Schornbaum.

[6] `http://www.walberla.net`

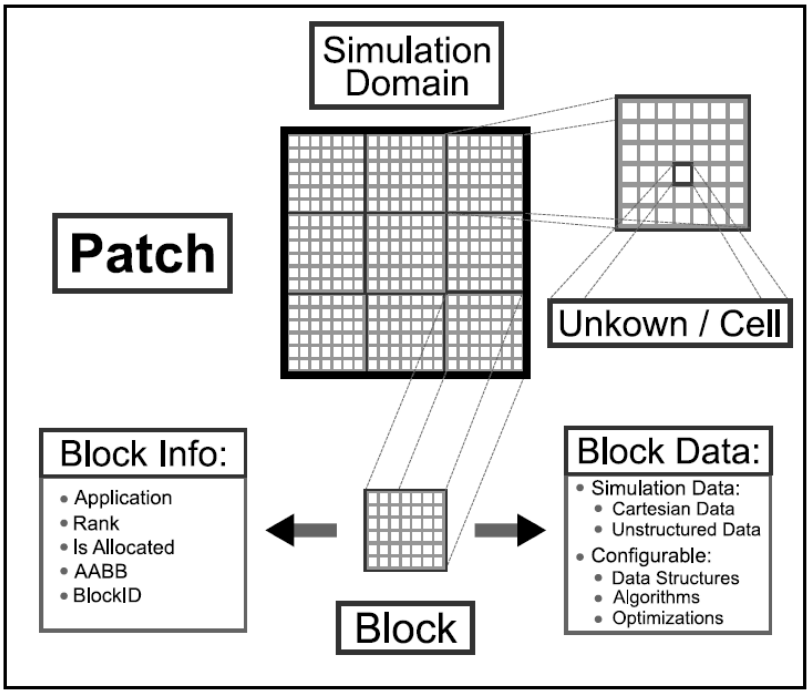


Figure 4.2.: Patch concept [1].

Summary: A Framework for Hybrid Parallel Flow Simulations with a Trillion Cells in Complex Geometries [30] Performance and scalability results of our lattice Boltzmann flow solver within waLBerla are presented for SuperMUC, at that time the world's fastest x86-based supercomputer ranked number 6 on the Top500 list, and JUQUEEN, a Blue Gene/Q system at that time ranked as number 5. It reaches resolutions of more than one trillion cells and performs up to 1.93 trillion cell updates per second using 1.8 million parallel threads. Based on a careful performance analysis, elaborate node level optimizations and vectorization with specific SIMD instructions on both machines result in highly optimized compute kernels for the single- and two-relaxation-time LBM. Excellent weak and strong scaling for hybrid OpenMP and MPI parallelization is achieved, also for a complex vascular geometry of the human coronary tree (see Figure 4.4), where simple load balancing strategies had to be applied. Note that the initialization phase and the setup of the simulation is completely parallelized.

4.2.2. Parallel Communication Concept

By introducing ghost layers at block boundaries it is easily possible to split up the computational domain for parallelization. Recently, more and more GPU HPC clusters have been installed, and thus there was a need to adapt waLBerla to multi-GPU environments. Figure 4.5 shows different kinds of data transfers used to realize multi-GPU support in

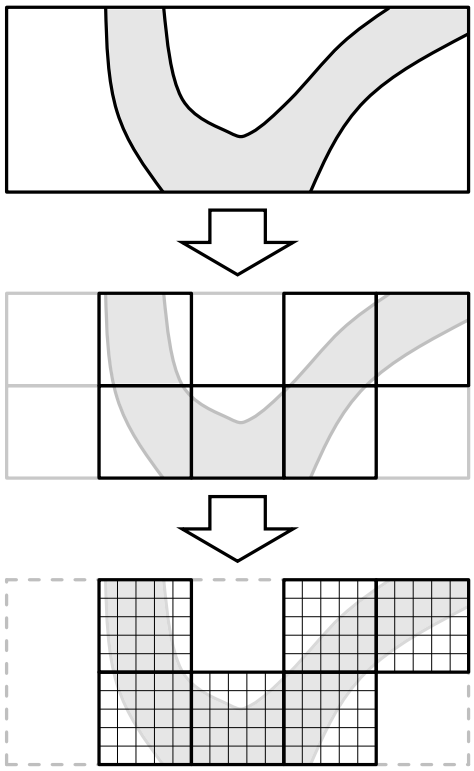
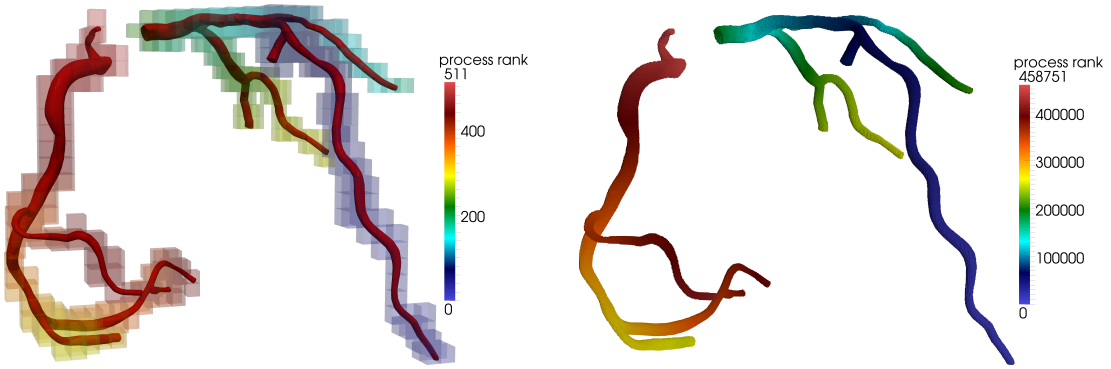


Figure 4.3.: Two-stage domain partitioning: block division and subsequent grid generation [30].



(a) One JUQUEEN nodeboard (512 processes, 485 blocks)

(b) Whole JUQUEEN (458752 processes, 458184 blocks)

Figure 4.4.: Domain partitioning of a coronary tree dataset with a target of one block per process [30].

waLBerla 2.0. Note that if two neighboring blocks are located at the same GPU, no communication is necessary and only the buffers holding ghost layer data are swapped. The GPU kernels are written in CUDA and also partly in OpenCL (Open Computing Language), and we use the corresponding functions to realize CPU-GPU data transfers via the PCI express bus. A manually created thread pool enables heterogeneous CPU-GPU simulations

and overlapping of computation and communication for multi-GPU simulations [2, 3].

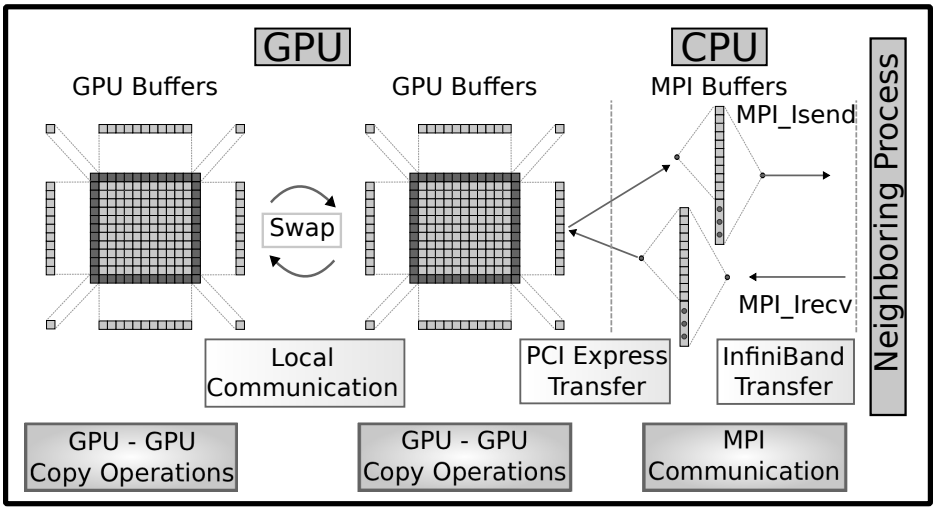


Figure 4.5.: Multi-GPU design [2].

Summary: A Flexible Patch-Based lattice Boltzmann Parallelization Approach for Heterogeneous GPU–CPU Clusters [2] Sustaining a large fraction of single GPU performance in parallel computations can be a problem on multi-GPU clusters. This issue is addressed by a multi-GPU implementation of LBM in waLBerla using a block-structured MPI parallelization, which is suitable for load balancing and heterogeneous computations on CPUs and GPUs. It is demonstrated that most of the kernel performance can be sustained for weak scaling on InfiniBand clusters, leading to excellent parallel efficiency. However, in strong scaling scenarios using multiple GPUs is much less efficient than running CPU-only simulations on CPU-based HPC clusters. Hence, a cost analysis must determine the best course of action for a particular simulation task and hardware configuration. Weak scaling results of heterogeneous simulations conducted on CPUs and GPUs simultaneously are also presented.

Summary: Performance Modeling and Analysis of Heterogeneous lattice Boltzmann Simulations on CPU-GPU Clusters [3] As already mentioned, software design concepts for an efficient and scalable multi-GPU parallelization within waLBerla support a pure-MPI and a hybrid parallelization approach. Thus, waLBerla is capable of running heterogeneous simulations on both CPUs and GPUs in parallel. Weak and strong scaling performance results obtained on the Tsubame 2.0 cluster for more than 1000 GPUs are presented. With the help of a model, a detailed and structured performance analysis is done. As one possible application of multi-GPU simulations, results of strong scaling experiments for flows through a porous medium are shown.

4.2.3. Sweep Concept

Most physical simulations are interested in effects showing up at or after a certain physical time, and therefore most simulations require time stepping or involve an iterative process. In each time step, usually operations on all blocks of the domain are performed. This time loop is modeled as sweep concept in waLBerla. A number of sweeps are executed sequentially, where in each sweep one or more kernels are called, which work on single blocks or the whole domain. Before or after each sweep, local communication between neighboring blocks, global communication like a reduction, timing, or visualization of the results can be done. A new sweep can be easily added by implementing a C++ class that respects the common sweep interface.

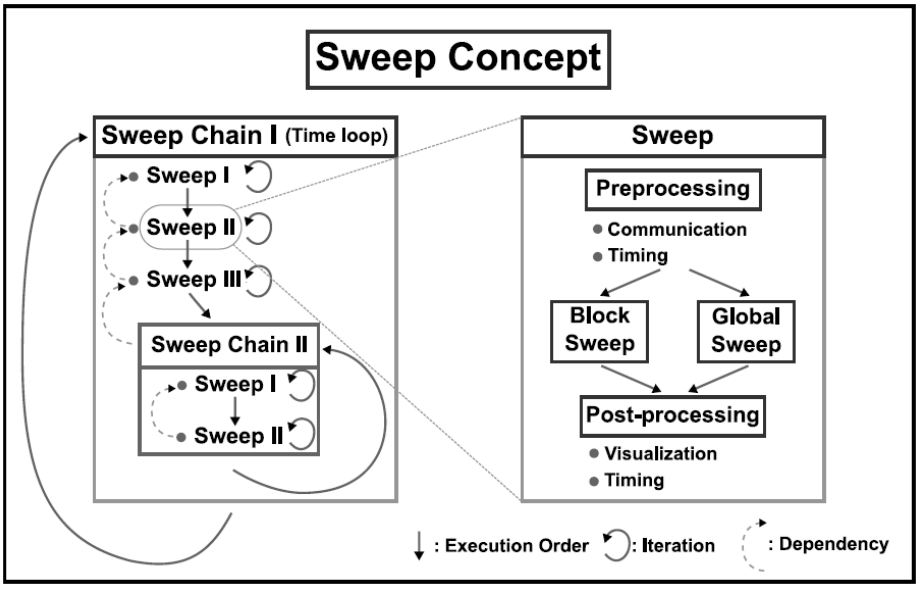


Figure 4.6.: Sweep concept [1].

4.2.4. Software Quality

As already mentioned, *performance* was the most important design goal for waLBerla at first. It can be split up into efficiency, i.e., the possibility to integrate optimized, hardware-adapted kernels in the simulations, and scalability, i.e., the support of massively parallel simulations. It was shown that waLBerla runs efficiently on various architectures and scales on current HPC clusters.

For *portability*, it is most often necessary to adapt the code to new architectures. This is obvious for GPU clusters, but also for new CPU clusters, algorithms have to be modified, e.g., to scale up to the latest HPC clusters it became important to do a hybrid OpenMP-MPI parallelization instead of a pure MPI parallelization to obtain best performance. Within single-node kernels, often architecture-specific SIMD instructions have to be used.

The third software quality factor that must not be underestimated is *productivity* that can be subdivided into *usability*, *reliability*, *maintainability*, and *expandability*.

Usability means an easy integration of new simulation scenarios and numerical methods also by non-programming experts. A compact and complete documentation is provided, which is generated by the tool *doxygen* and several tutorials are available to learn basic waLBerla concepts. To keep track of the development history and to manage multi-developer access, the version control system *git*[7] is used. The main user interface of waLBerla is currently a single, clearly structured input file activating and configuring the application. Simple input example files document the possibilities and give an insight in all features. This kind of user interface is especially useful for batch processing on large HPC clusters. For smaller or faster simulation runs, waLBerla provides a simple graphical user interface (GUI), and it is possible to couple waLBerla to our computational steering framework [14] *VIPER*, where one can interactively visualize the simulation results and manipulate simulation parameters.

Reliability in waLBerla concerns two main aspects: first, the robustness of the simulation, e.g., the certainty that the code will deliver numerically correct results and will terminate with a well-defined error message, when this is not possible, and second, coding errors. The first aspect is addressed by runtime checks that test whether the simulation is still in a correct physical state. As an example, the simulation will be interrupted, if physical parameters are no longer in a valid range. Support to help with coding errors is provided by several mechanisms. Since many of them impact the performance without contributing to the simulation itself, they are only active in a so-called debug execution mode. A comprehensive logging system supporting several levels of logging granularity and parallel simulations is used to report runtime information. If the monitoring system is activated, coding errors and problems originating from the physical model can be analyzed in detail. For most modules and classes, unit tests exist that are executed automatically after a change in the affected part of the code, when checking it in to the repository. For whole applications there are integration tests for the interplay between different modules.

Expandability addresses the integration of new functionality and applications without a major restructuring of code or a modification of core parts within the framework. It has been demonstrated by the integration of various applications, ranging from simple flow scenarios for scalability studies, over complex real-world multi-phase computations, to heterogeneous simulations that run concurrently on several types of hardware like GPU and CPU. The sweep concept in waLBerla also ensures a good maintainability and expandability. It is easily possible to adapt the internal data structures of the simulation to the needs of the application. Additionally, we provide clear and well documented interfaces that ensure that all parts of the framework have unique functionality.

4.3. Applications in waLBerla

Figure 4.7 shows some of the early CFD applications. The main focus in waLBerla 2.0 was on parallelization concepts for heterogeneous GPU–CPU Clusters. However, it was also possible to integrate new applications as highlighted in Figure 4.8. More details about the recent projects and an image gallery can be found at the websites of waLBerla[8] and *pe*[9].

[7] `http://git-scm.com/`
[8] `http://www.walberla.net`
[9] `http://www10.informatik.uni-erlangen.de/Research/Projects/pe/`

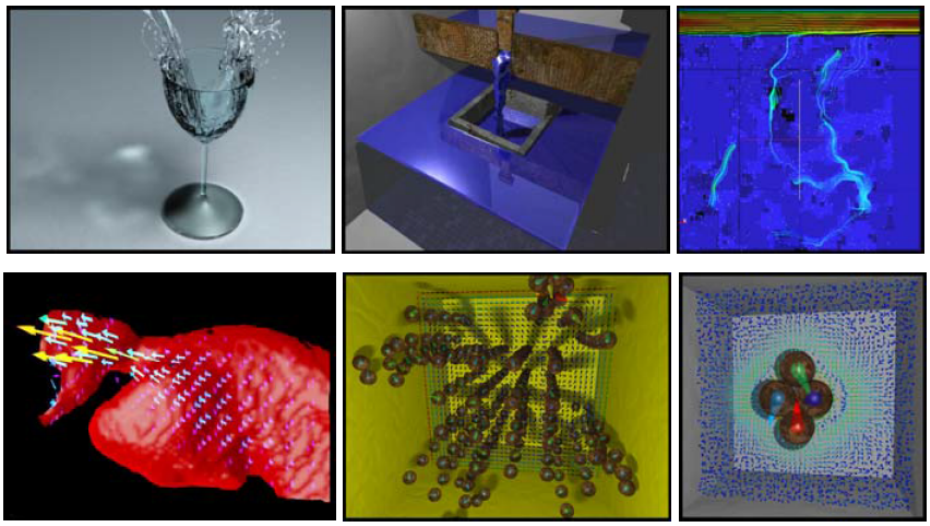


Figure 4.7.: Selection of simulation tasks in waLBerla [1]. The implementation has also been carried out within several joint projects and partly by master students. Form top left to lower right, one finds free-surface flows [100], free-surface flows with floating objects [86], flows through porous media, clotting processes in blood vessels [113], particulate flows for several million volumetric particles [111] on up to 8192 cores, and a fluctuating lattice Boltzmann [126] for nanoscale flows where thermal fluctuations are important.

Details how waLBerla and *pe* are coupled in order to realize combined fluid and particle simulations are depicted in Figure 4.9. The *pe* simulation domain is decomposed into a regular grid of cuboids matching with the domain decomposition of waLBerla.

Summary: Parallel Simulations of Self-propelled Microorganisms [44] Weak scalability of a parallel swarm simulation of self-propelled microorganisms in a fluid in the regime of low Reynolds numbers is investigated on JUQUEEN and SuperMUC. For this, waLBerla is coupled with the rigid body physics engine *pe*. The latter is enhanced with spring-like pair-wise interactions that can not only span over neighboring process domains, but as well act as long-range pair-wise interactions between distant process domains. The communication is designed in such a way that only neighboring processes and the processes hosting the objects, which are attached to the springs, have to exchange information directly.

Because of the increasing compute power, especially of accelerators like GPUs, real-time simulations and computational steering becomes possible. The user is able to visualize the simulation results in real-time and to interact with a running simulation, e.g., in order to change parameters or input data. Resulting from a student's project within my lecture *High End Simulation in Practice*, a computational steering framework called *VIPER* was developed by S. Kuckuk and coupled to waLBerla and *pe* [74].

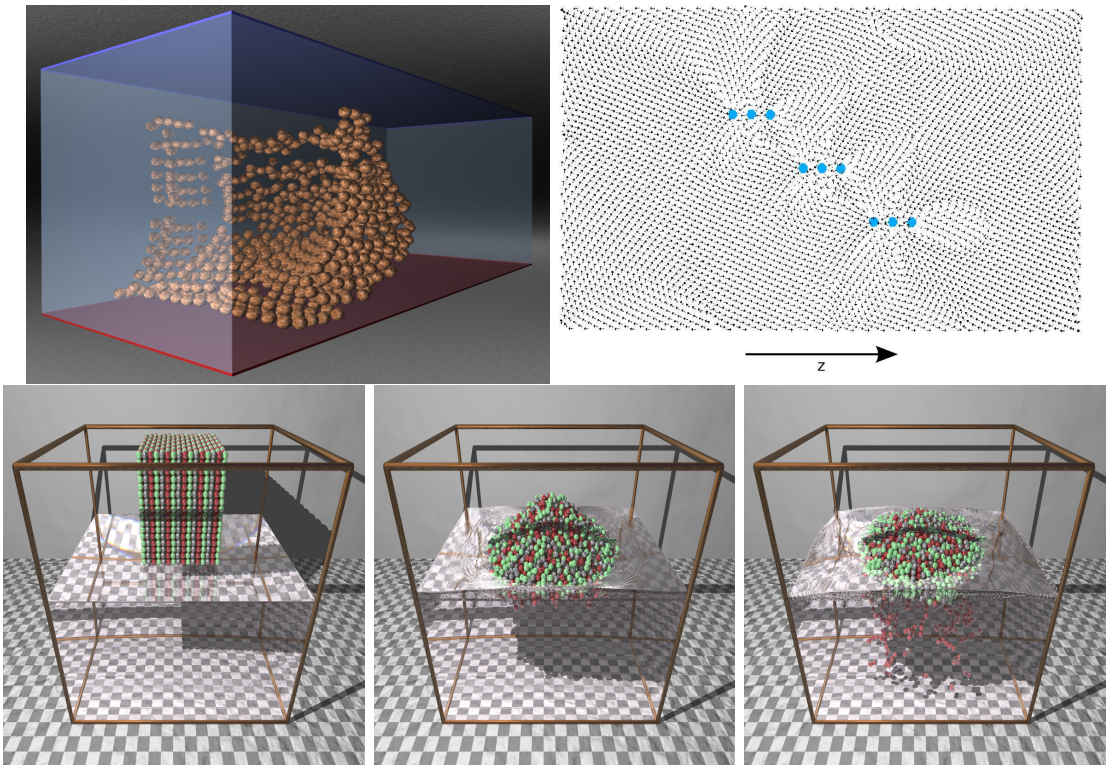


Figure 4.8.: Some of the newer multi-physics simulations coupling waLBerla and *pe*. Form top left to lower right, the attraction of homogeneously charged particles in fluid flow by an oppositely charged surface [82], the flow field of three three-sphere swimming devices [129], and particle segregation in a water basin [87] are visualized.

Summary: Interactive particle dynamics using OpenCL and Kinect [14] An interactive real-time simulation of granular, spherical particles which is able to run on a single workstation is described. It is based on a discrete element method approach and implemented in OpenCL and thus can be executed on CPUs and GPUs. The simulation results are visualized using DirectX™ 10 and instancing. Furthermore, the user can control the visualization and the simulation in a very intuitive way by supporting user tracking and speech recognition, both realized via the Microsoft® Kinect™ sensor. The performance of different implementation strategies on both CPUs and GPUs is compared, and, as a sample application, the Brazil nut effect is simulated.

Summary: A Framework for Interactive Physical Simulations on Remote HPC Clusters [64] The computational steering framework for **v**isualization and **i**nteractivity for **p**hysics **e**ngines in **r**eal-time, for short *VIPER*, is introduced. It is able to execute various physical simulations and to visualize the simulation results in real-time. Especially inter-

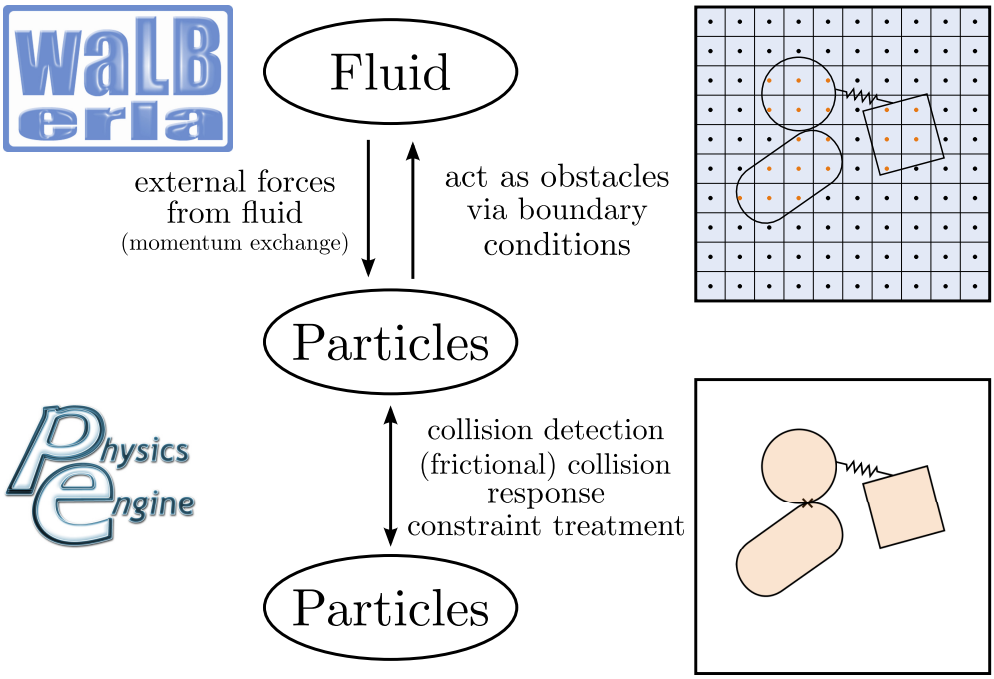


Figure 4.9.: Illustration of the four-way coupling of the waLBerla LBM fluid solver and *pe* rigid body physics engine [44].

esting in this context are simulations running on remotely accessible HPC clusters. As an example, we present a coupled waLBerla and *pe* simulation, together with the chosen visualization strategy and steering possibilities. Additionally, performance evaluations and a performance model concerning the update rate for remote simulations in the context of the *VIPER* framework are provided.

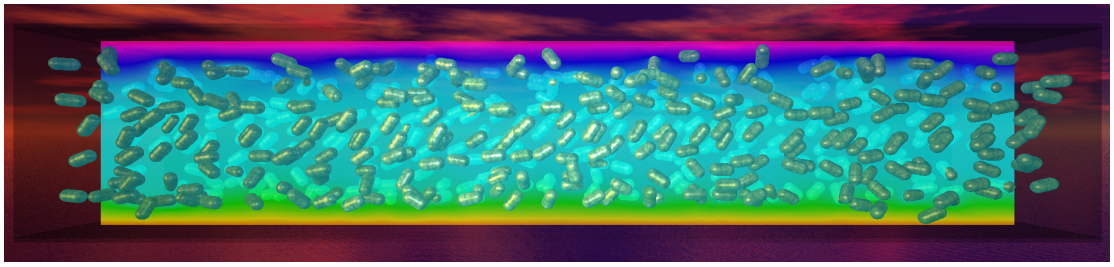


Figure 4.10.: Interactive visualization of a coupled waLBerla and *pe* simulation. Several capsules are located inside a box filled with liquid, at top and bottom opposing flows exist. The velocity in x direction is visualized [64].

4.4. Extensions and Maintenance of the Framework

Besides the traditional applications we currently extend waLBerla to be capable of doing micro-structure simulations based on phase-field models [125, 135, 137]. Here, we will benefit from adaptive grid refinement in massively parallel simulations, what will enable us to study physical effects that could not be simulated so far because of limited compute power.

As already mentioned, beginning 2014 we make waLBerla an open-source software and hope this step will help us to build up an even larger user base for our framework. Internally, the next major step is to finish the grid refinement and show that we are able to sustain performance for it also in massively parallel simulations.

The biggest issue with growing code complexity is maintainability. While new applications can be added easily, it is not always clear how they interact with other modules and if one can combine different modules without having to change them. This especially holds for parts of the code running on different hardware like GPU or CPU. Here, often kernels must be implemented twice, e.g., one CUDA and one C++ version. Thus, it is difficult to have GPU support for all applications. To my opinion this shows the limits of the classical framework approach in general, because the developers have to provide a large library to fulfill all requirements. In the next chapters, I will therefore start from the class of multigrid algorithms, and then show a different approach to provide implementations on different hardware or for different CSE applications, which require adapted algorithmic components or parameters. Instead of writing a larger framework manually, which becomes more and more complex to maintain, the user will be able to specify his problem in an abstract way, and then a very specific implementation will be created automatically for him. Of course, it is very difficult to create code for a whole CSE application. Therefore, a first step is to generate only parts of the code, e.g., single compute kernels. Combined with a framework like waLBerla, this approach can on the one hand help us to extend the framework much faster, and on the other hand it will be simpler for users to include their own applications.

5. Efficient Multigrid Algorithms

5.1. Massively Parallel Multigrid Solvers

Multigrid methods belong to the important class of algorithms in CSE that deal with the numerical solution of large, sparse, (non)linear systems arising, e.g., after the discretization of (elliptic) partial differential equations (PDEs) [136, 90]. The most basic elliptic PDE is Poisson's equation

$$\Delta u = f \quad \text{in } \Omega \tag{5.1a}$$
$$u = 0 \quad \text{on } \partial\Omega \tag{5.1b}$$

with homogeneous Dirichlet boundary conditions. It has proven to be a good test case especially for showing scalability of parallel multigrid algorithms, because of its ratio between computation and data transfer, which makes its performance mainly bounded by memory bandwidth and communication costs.

One way to distinguish multigrid methods is with respect to the type of grid. In the following, we concentrate on geometric multigrid methods on regular and on hierarchical hybrid grids, i.e., unstructured coarse grids that are regularly refined. On a regular grid Eq.(5.1) can be discretized, e.g., by finite differences, on hierarchical hybrid grids, e.g., by linear finite elements. Both lead to a linear system

$$A^h u^h = f^h \,, \quad \sum_j a_{ij}^h u_j^h = f_i^h \,, i, j \in \{1, \ldots, N\} \tag{5.2}$$

with system matrix $A^h \in \mathbb{R}^{N \times N}$, unknown vector $u^h \in \mathbb{R}^N$ and right hand side (RHS) vector $f^h \in \mathbb{R}^N$ on a discrete grid Ω^h. N denotes the total number of grid points and corresponds to the number of unknowns in the linear system.

The main part of a standard iterative multigrid solver is the V-cycle listed in algorithm 5.1. It traverses fine and coarse grids in the grid hierarchy [136].

Over the last years our group gained experience with different multigrid methods on various platforms, where the emphasis is on highly parallel geometric multigrid solvers. In Figure 5.1 our most important performance results obtained for parallel multigrid for Eq.(5.1) are pinpointed together with early parallel multigrid solvers from some of the multigrid pioneers. In [81] one finds performance results for one of the first vectorized multigrid Poisson solvers on the CDC Cyber 205, a system with four vector pipelines, theoretically delivering 400 MFLOPs (double precision). Here, a linear system with 16.129 unknowns was solved and one V(2,1)-cycle took about 5.9 s. Note that besides modern GPUs like NVIDIA GTX 480, where we apply similar algorithmic transformations [12], this is the only architecture, where the ratio between number of unknowns and unknowns per second and thus the solver runtime is much smaller than 1 s. In 1986, it was possible to solve a linear

Algorithm 5.1 Parallel recursive V-cycle: $u_h^{(k+1)} = V_h(u_h^{(k)}, A^h, f^h, \nu_1, \nu_2)$

1: **if** coarsest level **then**
2: solve $A^h u^h = f^h$ by direct or iterative method in parallel
3: **else**
4: $\bar{u}_h^{(k)} = \mathcal{S}_h^{\nu_1}(u_h^{(k)}, A^h, f^h)$ // ν_1 *pre-smoothing steps and exchange ghost layers of solution*
5: $r^h = f^h - A^h \bar{u}_h^{(k)}$ // *compute residual*
6: $r^H = R^h r^h$ // *restrict residual and exchange ghost layers of residual*
7: $e^H = V_H(0, A^H, r^H, \nu_1, \nu_2)$ // *recursion*
8: $e^h = P^h e^H$ // *interpolate error*
9: $\tilde{u}_h^{(k)} = \bar{u}_h^{(k)} + e^h$ // *coarse grid correction and then exchange ghost layers of solution*
10: $u_h^{(k+1)} = \mathcal{S}_h^{\nu_2}(\tilde{u}_h^{(k)}, A^h, f^h)$ // ν_2 *post-smoothing steps and exchange ghost layers of solution*
11: **end if**

system with around one million unknowns on one Intel 80286/80287 machine, but it took around 14 hours and one had to use hard disk space, since there was not enough main memory to store all data [130]. Around that time several parallel multigrid algorithms were developed [134, 124] and tested, e.g., on the Caltech Mark II Hypercube, which was based on 8086/8087, and the Intel iPSC/2 consisting of 16 Intel 80386 processors. Some years later multigrid methods had already grown up and were used in many applications and on many platforms [80, 89, 83, 140], e.g., on a Cray T3D (3D torus) supercomputer with 256 processing elements. Since 2003 the hierarchical hybrid grids framework HHG was developed, first by B. Bergen [85, 84], then by T. Gradl [32], and recently by B. Gmeiner [107]. Over the years HHG always scaled nicely to some of the biggest machines available, e.g., Hitachi SR8000-F1 [84], HLRB II [32], JUGENE [4], JUQUEEN [108], and SuperMUC [108] (see Table 4.1). Especially for regular grids and GPU clusters, a multigrid solver was also integrated in waLBerla. Its scalability on the multi-GPU cluster Tsubame 2.0 is reported in [34]. On the currently biggest machine, Tianhe-2, one could theoretically solve Eq.(5.1) on a regular grid with around 20 trillion unknowns. The limiting factor for the number of unknowns is the available main memory.

5.2. Multigrid on GPU

Geometric multigrid methods are perfectly suitable for GPUs due to their regular memory access patterns. This fact can, e.g., be exploited to achieve real-time image processing on GPUs.

Summary: Performance Engineering to achieve Real-time High Dynamic Range Imaging [12] Image processing applications like high dynamic range imaging can be done efficiently in the gradient space. For it, the image has to be transformed to gradient space and back. While the forward transformation to gradient space is fast by using simple finite differences, the backward transformation requires the solution of Eq.(5.1) via multigrid.

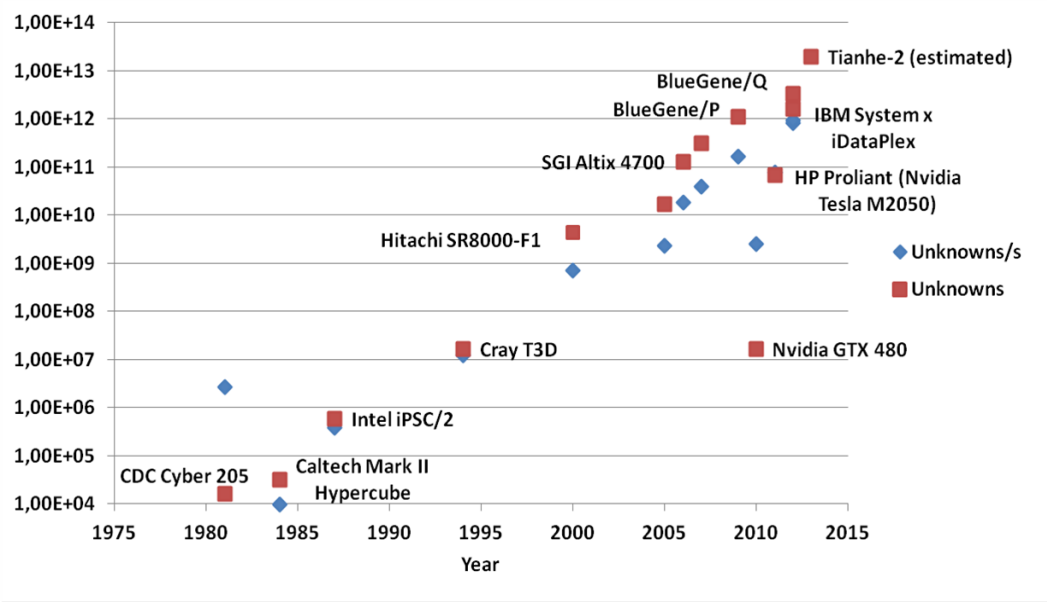


Figure 5.1.: Problem sizes (number of unknowns) and runtimes in unknowns per second over the years for geometric multigrid solvers applied to Eq.(5.1). The year corresponds to the first time the specific machine was installed.

In order to predict the optimal performance on a certain architecture, and to judge the quality of an implementation, a performance model is developed. For a standard multigrid algorithm the overall runtime

$$T = t_V \cdot log_{\rho_\infty} \kappa \tag{5.3}$$

depends on the time for one V-cycle t_V, the desired error reduction factor κ, and the asymptotic convergence rate ρ_∞. Convergence rates can be predicted for geometric multigrid solvers via local Fourier analysis (LFA) [138]. t_V is determined by computation and communication time of the algorithm summed over all levels.

The performance model guides us to an efficient implementation, where we achieve an overall performance of more than 25 frames per second for 16.8 Megapixel 2D X-ray images, doing full high dynamic range compression including data transfers between CPU and GPU. The most important optimizations are on the one hand algorithmic changes, like switching to a red-black version of the smoother and splitting the data arrays accordingly, on the other hand a reduction of the memory transfers by, e.g., cache blocking techniques. Together with a simple OpenGL visualization it becomes possible to perform real-time parameter studies on medical data sets.

As already mentioned we extended waLBerla to support geometric multigrid algorithms for the numerical solution of PDEs on multi-GPU clusters [37]. Basically, the grid hierarchy was realized by a number of nested simulation domains in waLBerla, and efficient compute kernels for the multigrid components were integrated. As solver on the coarsest grid a conjugate gradient (CG) method is used. Alternatives would be parallel direct solvers or a

reduction of the coarser grids to less processes.

Summary: A Geometric Multigrid Solver on Tsubame 2.0 [34] In the proceedings of the Dagstuhl seminar [91] one finds weak and strong scaling results on Tsubame 2.0 with up to 1029 GPUs for our waLBerla multigrid solver. Tsubame 2.0 was at that time number 5 in the Top 500 list. Although a large fraction of the performance can be sustained, for an increasing number of GPUs the efficiency of the CG method on the coarsest grid decreases and becomes a bottleneck for the parallel efficiency.

5.3. Multigrid on Hierarchical Hybrid Grids

The HHG framework is based on unstructured tetrahedral finite elements that are regularly refined to obtain a block-structured computational grid. We call this hierarchical hybrid grid.

Summary: Parallel multigrid on hierarchical hybrid grids: a performance study on current high performance computing clusters [4] Performance and scalability of a geometric multigrid solver for Eq.(5.1) and implemented within HHG is studied on HPC clusters up to nearly 300,000 cores. One challenge was the parallel mesh generation from an unstructured input grid, which roughly approximates a human head from a 3D magnetic resonance imaging data set. This grid is then regularly refined to create the HHG grid hierarchy. As test platforms, JUGENE and lima, an Intel Xeon 5650 cluster located at the local computing center in Erlangen, are chosen. To estimate the quality of our implementation and to predict runtime for the multigrid solver, a detailed performance and communication model is developed and used to evaluate the measured single node performance, as well as weak and strong scaling experiments on both clusters. Thus, for a given problem size, one can predict the number of compute nodes that minimize the overall runtime of the multigrid solver. Overall, HHG scales up to the full machines, where the biggest linear system solved on JUGENE had more than one trillion unknowns. Note that CG is again applied as solver on the coarsest grid and is included in the performance model. In contrast to Tsubame 2.0, also for larger test runs CG does not influence the parallel efficiency severely.

Summary: A Generic Prototype to Benchmark Algorithms and Data Structures for Hierarchical Hybrid Grids [40] In order to generalize the data structures and multigrid components required to solve elliptic PDEs on hierarchical hybrid grids, a generic 2D prototype for hierarchical hybrid grids was implemented including a standard multigrid solver. Starting with rectangles, our goal is to extend the implementation to various primitives like triangles, cubes, tetrahedra, or prism. The resulting multigrid algorithm is again highly scalable up to more than $450,000$ cores of JUQUEEN. To solve the problem on the coarsest grid, CG

is compared to a parallel algebraic multigrid, which is realized by the BoomerAMG [101] implementation included in the HYPRE[1] package.

5.4. Advanced Multigrid Components

If one is interested in solving more complex PDEs, e.g., involving jumping coefficients, the multigrid components usually have to be adapted, because they are problem-dependent. This means suitable smoothers, transfer operators, and coarse representations of the systems have to be found. We neglect here the selection of the coarse grid points as done in algebraic multigrid, because this is trivial for geometric multigrid on regular grids. For parallel multigrid, often block smoothers or hybrid smoothers are used [136]. Note that the latter lead in general to a worse convergence rate, since communication at process boundaries is neglected or delayed. Prolongation depends on the system matrix in general, and restriction is often chosen to be its transpose for symmetric problems. The coarse grid system matrix can be computed via direct or Galerkin coarsening.

Summary: A practical framework for the construction of prolongation operators for multigrid based on canonical basis functions [20] A general approach for the construction of prolongation operators for multigrid methods is discussed. It turns out that classical black-box prolongation or prolongation operators based on smoothed aggregation can be classified as special cases. The approach is suitable both for geometric and for purely algebraic multigrid settings. It allows for a simple and efficient implementation and parallelization by introducing canonical basis functions. Numerical results are shown for several diffusion problems with strongly varying or jumping coefficients. Possible applications for the method are 3D medical image segmentation or non-symmetric convection-diffusion problems. Instead of traditional Galerkin coarsening collocation coarse approximation (CCA) [139] is applied. This keeps the number of non-zero entries in each row of the coarse system matrix constant.

5.5. Abstract Description of Multigrid Algorithms

Multigrid consists of several components and the algorithm requires various parameters to be set, like the number of smoothing steps or the cycle type. Thus, if multigrid is integrated into a software framework like waLBerla, it should be easily possible to exchange the components and adapt parameters dependent on the current problem that has to be solved. As outlined in the last chapter, one requires therefore an abstract description of the multigrid algorithm.

[1] http://www.llnl.gov/CASC/hypre/

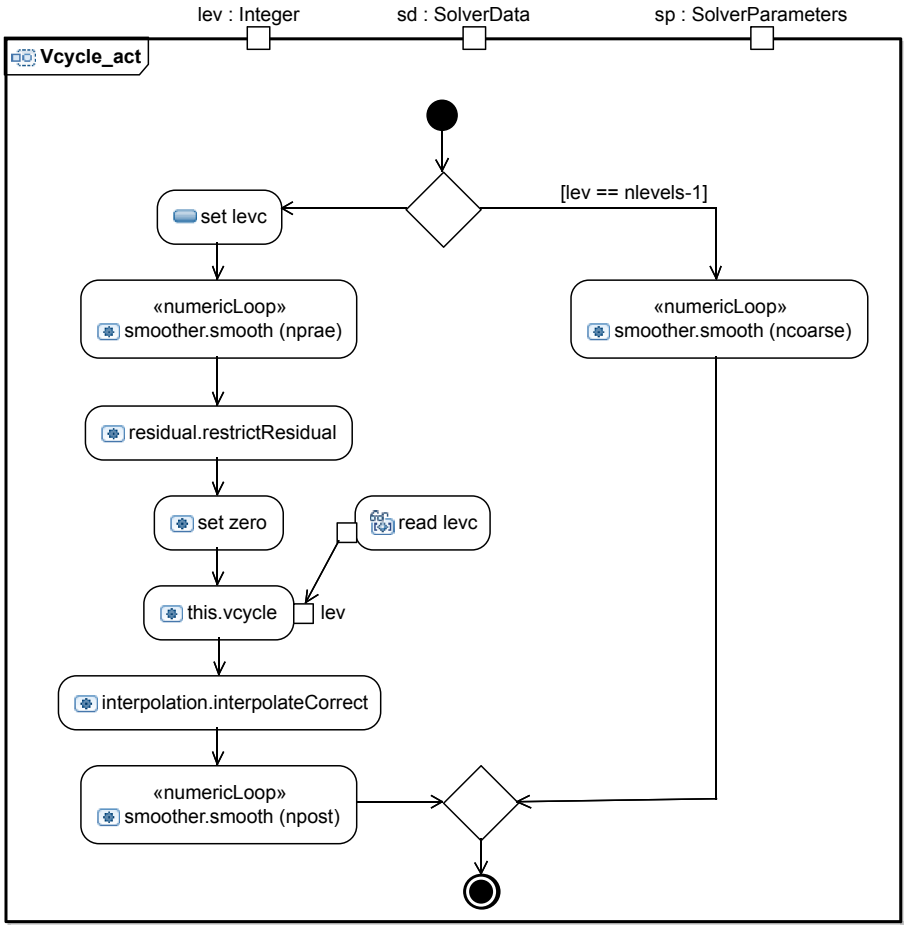


Figure 5.2.: UML activity diagram representing an abstract implementation of the V-cycle algorithm 5.1 [24].

Summary: Modeling Multigrid Algorithms for Variational Imaging [24] One approach based on a graphical representation of the multigrid algorithm using UML is found in Figure 5.2. UML-based modeling is becoming increasingly popular in many software development projects. One of the key aspects is the possibility to support automatic code generation from UML models while keeping the easy to use modeling abstraction for the software developer. The framework *Syntony*[2] has been developed to generate discrete-event simulations from standard-compliant UML models, in order to support simulation-based performance evaluation of systems. Syntony was extended to include automatic code generation in the context of large-scale continuous simulations, which require the numerical solution of PDEs. As examples multigrid solvers for applications in variational imaging are chosen. Since the V-cycle exhibits a fixed sequential structure, and only the single steps are problem dependent, these can be implemented in separate, efficient kernels, e.g., in a high-level language like C++. Syntony provides a modeling framework, which can assemble new application code from basic modules and data structures in C++ and abstract descriptions of algorithms and classes provided as UML class and activity diagrams. The approach is evaluated in a case study for image denoising. The generated code is a fully working application, which computes a denoised output image from a given input image using the methods specified in the UML model. The key benefit lies in the abstraction from low level programming when building complex denoising algorithms. In addition, it is shown that the code generation and compilation process runs significantly faster than the compilation of the entire framework. The run-time overhead introduced by the generated code is negligible.

While this first approach to automatically generate multigrid codes worked in principle, there were still some unresolved issues. While the abstract description via UML is very intuitive, its expressiveness and tool-support to create the correct diagrams was limited. Furthermore, the compute kernels still had to be provided manually. Thus, we followed another path in order to obtain (performance)-portable implementations of multigrid solvers [43].

Summary: Towards a Performance-portable Description of Geometric Multigrid Algorithms using a Domain-specific Language [15] Different processor types can be found on a single node of current HPC systems including accelerators such as GPUs. To cope with the challenge of programming such heterogeneous systems, a domain-specific approach to automatically generate code tailored to different processor types is presented. Low-level CUDA and OpenCL is generated from a high-level description of a geometric multigrid algorithm written in a Domain-Specific Language (DSL), which is embedded in C++ instead of writing hand-tuned code for GPU accelerators directly. The DSL is part of the Heterogeneous Image Processing Acceleration (HIPAcc) framework[3]. It was extended in this work to handle grid hierarchies in order to model different cycle types like V-cycles or W-cycles within the multigrid algorithm. The proposed approach allows to generate efficient implementations that achieve similar performance compared to hand-tuned codes [12].

[2] `http://www7.informatik.uni-erlangen.de/syntony`

Building upon this previous work, a more general way to design an abstract description not only of a multigrid algorithm, but of a whole CSE application, is introduced in the next chapter.

[3] `http://sourceforge.net/projects/hipacc`

6. Future Trends in Software Design for CSE Applications

This chapter describes ongoing research done within the project *Advanced Stencil-Code Engineering* (ExaStencils). It started in 2013 and is supported by the German Research Foundation (DFG) through the Priority Programme 1648 *Software for Exascale Computing (SPPEXA)*. The project partners are listed in Figure 6.1.

Figure 6.1.: ExaStencils project.

Main goal of the project is to develop new concepts to increase productivity, performance, and portability of geometric multigrid codes. Currently, five PhD students are working on ExaStencils. In Erlangen, S. Kuckuk deals with massively parallel multigrid algorithms on hierarchical hybrid grids [40], Ch. Schmitt with domain specific language design, code transformations, and platform-specific code adaptations. In Passau, S. Kronawitter considers loop optimization in the polyhedron model and other low-level, hardware-specific optimizations [123], A. Grebhahn domain-specific optimization using parameter tuning and machine learning [33],[133]. In Wuppertal, H. Rittich works on multigrid theory and develops an LFA tool to estimate geometric multigrid convergence rates for various multigrid components and parameters. In the following, the current state of the project from my point of view is summarized in more detail.

6.1. Generic Description of CSE algorithms

6.1.1. Domain-driven Software Development

In Figure 6.2, one common way to develop CSE applications is described. Here, the CSE expert mainly focuses on parallel algorithms, implementation or extension of a framework, and tuning to specific hardware. The users of the framework have to port the application to it, i.e., the model and the solution method have to be formulated in the language of the framework. If the framework already supports the application, this means that just suitable modules have to be used, if not, the framework has to be extended. For waLBerla, a C++ interface is provided, other frameworks also provide interfaces to high-level languages like Python. But in both cases, the user has to implement his application directly, perhaps with the help of the framework developers.

In contrast to a classical framework, which contains a collection of algorithms and applications and can be extended manually, our approach generates implementations adapted to a specific application and hardware. The code generation process itself has to work not only for a single application, but a certain class of applications (also called *domain*) typically defined by the underlying algorithms (see Figure 6.3). Note that in most cases, frameworks will not become obsolete, but will be complemented by this approach, since they can be integrated in the generated code as external modules.

Our domain consists of classical CSE applications that require the solution of PDEs. Users can specify a problem on a high level of abstraction. The domain expert brings in algorithmic knowledge and is usually a CSE expert, the mathematician sets constraints to the algorithmic components or provides knowledge to estimate the (problem-dependent) efficiency for certain algorithms, the software specialist implements the code generation framework, and the hardware specialist may add problem-independent tuning of the implementation to a certain platform. The feature model allows to build up a global optimization problem to find the optimal implementation for a certain problem, where algorithms, their components and parameters, and possible tuning strategies to a specific hardware are selected automatically from a list of possibilities. The search space for the optimization problem is constrained by domain knowledge, either specified by the domain experts, or found based on machine learning techniques.

6.1.2. Prototypical CSE Application

In fact, this is an ambitious goal that one can reach only step by step. First, the process to describe a prototypical CSE application is summarized in Figure 6.4. To illustrate the complexity of real-world CSE applications, we consider an example from quantum chemistry. It will be later also the first test case within the ExaStencils project.

Problem The goal is to simulate the dynamics of a molecular system consisting of n_n nuclei and n_e electrons [18].

Continuous Domain and Model The *time-dependent Schrödinger equation* [132]

$$i\hbar\frac{\partial}{\partial t}\Psi = \hat{H}\Psi \tag{6.1}$$

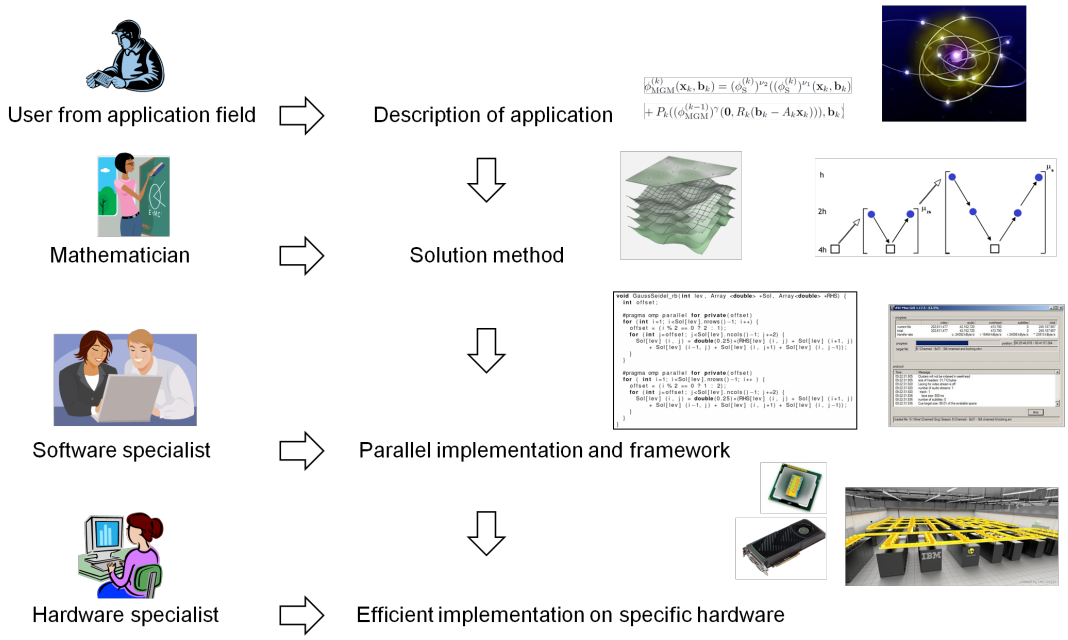


Figure 6.2.: State of the Art: Application-driven Projects.

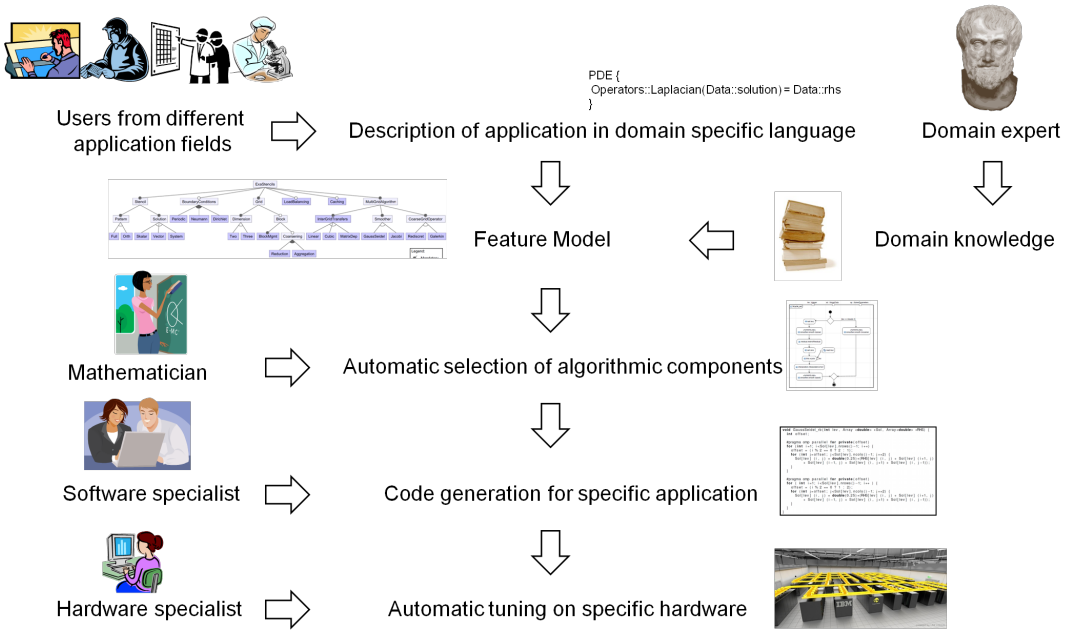


Figure 6.3.: Proposed: Domain-driven Projects.

on the regular 3D domain $\Omega \subset \mathbb{R}^3$ describes, how quantum systems evolve over time t. i is the imaginary unit, $\hbar$ the reduced Planck constant, and $\hat{H}$ is a Hamiltonian operator that characterizes the total energy in the system. The wave function Ψ, which is in general

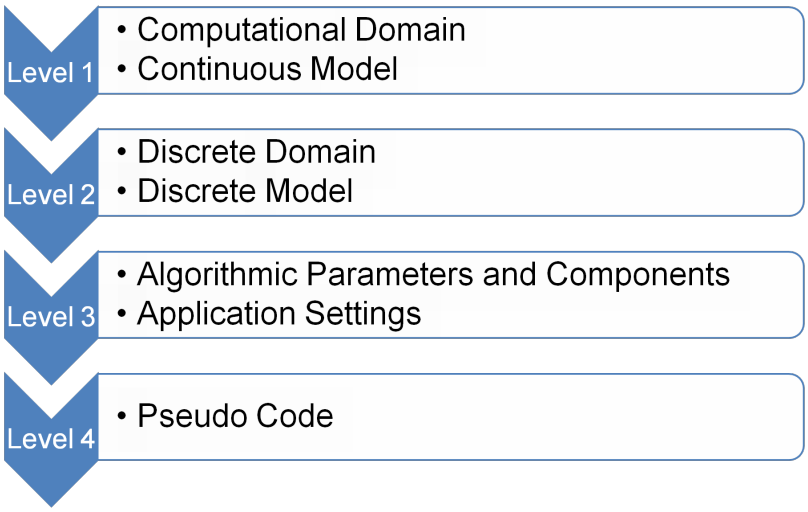


Figure 6.4.: Different levels or steps for describing a prototypical CSE application.

complex, describes the quantum state of an elementary particle, $|\Psi|^2$ is the probability density of observing the particle at a certain time at a certain position. A quantum state is a state vector, e.g., for an electron in an atom, it is simply the principal or first quantum number. A complete description of the electron requires the four quantum numbers energy, angular momentum, magnetic moment, and spin.

In Eq.(6.1) the wave function depends on the spatial coordinates of all elementary particles and thus one would have to solve a $3 \cdot (n_n + n_e)$ dimensional time-dependent PDE to obtain it. Already for small systems this becomes infeasible. Therefore, an often applied simplification is the *Born-Oppenheimer (BO) approximation* [88]. It splits the wave function into a component for the electrons and a component for the nuclei and then solves the Schrödinger equation for both parts independently. This is motivated by the high ratio between nuclear and electronic masses. Thus, in a first step the positions of the nuclei are fixed and modeled as external potential when computing the electron movement. In a second step, only the nuclei are moved. Thus, we have to solve one $3n_n$ and one $3n_e$ dimensional time-dependent PDE now.

If the Hamiltonian $\hat{H}$ does not depend explicitly on time, one can separate $\Psi(x,t)$ into $\Psi(x,t) = \psi(x)\tau(t)$. Then, $i\hbar\frac{\partial}{\partial t}$ becomes an energy eigenvalue ϵ and the resulting *time-independent Schrödinger equation* is an eigenvalue equation for the Hamiltonian

$$\epsilon\psi = \hat{H}\psi\,. \tag{6.2}$$

The *density functional theory (DFT)*, here specifically the *Kohn-Sham DFT* [116, 122], is a modeling approach to further simplify the simulation of quantum systems especially to investigate the electronic structure of atoms or molecules. Within DFT, the *Kohn-Sham equation* is the Schrödinger equation of a fictitious system of non-interacting electrons that generate the same density as any given system of interacting electrons. Since electrons do not interact, one can further simplify Eq.(6.2). It now reads for each particle i

$$\epsilon_i\psi_i(r) = \left(-\frac{\hbar^2}{2m}\nabla^2 + v_{eff}(r)\right)\psi_i(r)\,. \tag{6.3}$$

Basically the energy characterized by $\hat{H}$ is the sum of kinetic energy and the potential energy of the electron. The local effective (fictitious) external potential v_{eff} is called the *Kohn-Sham potential*.

In principle, the one-electron wave function ψ_i can be seen as an atomic orbital and ϵ_i is its orbital energy. The quantum states of an atom are labeled by a set of quantum numbers usually associated with particular electron configurations, i.e., by occupation schemes of atomic orbitals.

Depending on the density of the system

$$\rho = \sum_{i}^{n_e} |\psi_i|^2 \tag{6.4}$$

one constructs an energy functional

$$E(\rho) = \sum_{i=1}^{n_e} \int dr \psi_i^*(r) \left(-\frac{\hbar^2}{2m} \nabla^2 \right) \psi_i(r) + \int dr v_{ext}(r) \rho(r) + V_H(\rho) + E_{xc}(\rho) \tag{6.5}$$

in order to obtain the total energy of the system. Besides the kinetic energy, $E(\rho)$ contains an external potential v_{ext}, which will later model the influence of the nuclei, the *Hartree (Coulumb) potential* V_H modeling electron-electron interactions, and the energy E_{xc} corresponding to, e.g., the non-classical and non-linear exchange-correlation potential [94, 128], which includes all many-particle interactions.

Method 1: One can now solve the nonlinear eigenvalue problem stated in Eq.(6.2), e.g., by a multigrid algorithm as proposed in [96]. This will be considered later in ExaStencils.

Method 2: For now, we follow the approach from the quantum chemical program RSDFT[1] developed by R. Schmid [18] to simulate ab initio molecular dynamics.

RSDFT uses *Car-Parrinello molecular dynamics* [93], where one introduces a fictitious mass parameter for the wave functions and then performs a joint dynamic of both nuclei and wavefunction. Next we use the bra-ket notation [98] to describe quantum states, e.g., the dot product on a complex vector space for two quantum states Ψ_1 and Ψ_2 can be written as $\langle \Psi_1 | \Psi_2 \rangle$. The above energy functional (6.5) becomes

$$E_{DFT} = \sum_{i=1}^{n_e} \langle \psi_i | -\frac{1}{2} \nabla^2 | \psi_i \rangle + E_{ne}(\rho, R) + E_{nn}(R) + E_{ee}(\rho) + E_{xc}(\rho) \,, \tag{6.6}$$

where R denotes the position of the nuclei, and the single terms correspond to the energy of different kinds of interactions between electrons e and nuclei n. Formulated as eigenvalue problem as above, one obtains

$$\epsilon_i |\psi_i\rangle = \left(-\frac{1}{2} \nabla^2 + V_{ne} + V_{ee} + V_{xc} \right) |\psi_i\rangle \,. \tag{6.7}$$

The Hartree potential V_{ee} can be computed by solving the Poisson equation

$$\nabla^2 V_{ee}(r) = -4\pi \rho(r) \,. \tag{6.8}$$

[1] `http://www.rsdft.org`

For the exchange-correlation potential $V_{xc} = \delta E_{xc}/\delta\rho$ many approximations exist, we choose here generalized gradient approximation functionals depending on the local density and the density gradient.

In order to further reduce the computational effort we note that electrons on inner shells are not playing a significant role in the chemical binding of atoms. Thus, they can be ignored and only the valence electrons have to be considered in many applications, especially if they involve metals. One then works with pseudo-potentials that approximate in V_{ne} the potential felt by the valence electrons [115].

The one-electron wave functions ψ_i must fulfill the orthogonality constraint $\langle\psi_i|\psi_j\rangle = \delta_{ij}$ with Kronecker delta δ_{ij}. This is ensured by a Lagrange multiplier matrix Λ_{ij}.

The overall *Car-Parrinello Lagrangian* reads now

$$\mathcal{L}_{CPMD} = \frac{1}{2}\sum_{n=1}^{n_n} M_n\dot{R}_n^2 + \frac{1}{2}\sum_{i=1}^{n_e}\mu_e\langle\dot{\psi}_i|\dot{\psi}_i\rangle - E_{DFT} + \sum_{ij}\Lambda_{ij}\left(\langle\psi_i|\psi_j\rangle - \delta_{ij}\right) \tag{6.9}$$

M_n is the mass of the nuclei, R_n the position of the nuclei, μ_e the fictitious mass of the electrons. The Euler-Lagrange equations are Newton's equations of motion $F = ma$ for the nuclei and the electron wavefunctions

$$M_n\ddot{R}_n = -\frac{\partial E_{DFT}}{\partial R_n} \tag{6.10}$$

$$\mu_e|\ddot{\psi}_i\rangle = -\frac{\delta E_{DFT}}{\delta\langle\psi_i|} + \sum_j \Lambda_{ij}|\psi_j\rangle \tag{6.11}$$

Note that in the limit $\mu_e \to 0$ we approach the BO approximation.

Discrete Domain and Model In RSDFT the wavefunctions ψ_i are discretized on a regular 3D grid with grid spacing $h = (h_x, h_y, h_z)$ resulting in vectors ψ_i^h. The main issue for the numerical molecular dynamics simulation is energy conservation. Spatial integrals are approximated via a trapezoidal rule, i.e., a simple sum weighted by the cell volume. Derivatives like occurring in the kinetic energy term or the exchange-correlation energy term are evaluated using higher order finite differences. In order to obtain the discrete Poisson equation

$$\nabla_h^2 V_{ee}^h(r_h) = -4\pi\rho(r_h) \tag{6.12}$$

one can apply mehrstellen discretization [136] to achieve a discretization error order $\mathcal{O}(h^4)$.

For time discretization we use a second order Verlet propagator

$$R_n(+) = 2R_n(0) - R_n(-) + \frac{\Delta t^2}{M_n}\left(-\frac{\partial E_{DFT}(0)}{\partial R_n}\right) \tag{6.13}$$

$$|\psi_i(+)\rangle = |2\psi_i(0)\rangle - |\psi_i(-)\rangle + \frac{\Delta t^2}{\mu_e}\left(-\langle\dot{\psi}_i(0)| + \sum_j \Lambda_{ij}|\psi_j(0)\rangle\right) \tag{6.14}$$

$$\delta\rho = \rho(r) - \sum_n \rho_n^{comp}\,. \tag{6.15}$$

Algorithm and Application Settings The problem-dependent parametrization like the number of electrons in the system or physical properties like domain sizes have to be set by the application expert, the other algorithmic parameters can be also tuned automatically.

Pseudo Code In each time step the wave function gradient $\frac{\delta E_{DFT}}{\delta \langle \psi_i |}$ is updated as described in [18] via

1. compute Laplacian of wave function for kinetic energy term,
2. add $V_{nn} = \sum |\rho(R_n)\rangle\langle\rho(R_n)|\psi_i\rangle$,
3. compute density ρ,
4. compute exchange correlation potential V_{xc} depending on ρ,
5. solve Poisson's equation for changes in ρ, compute it before, and
6. compute total potential.

After propagating nuclei and electron wavefunctions, the orthonormality constraint $\langle\psi_i|\psi_j\rangle = \delta_{ij}$ is ensured by using the SHAKE algorithm [131].

Summary: A parallel multigrid accelerated Poisson solver for ab initio molecular dynamics applications [11] A parallel multigrid solver was integrated in RSDFT to solve the Coulomb problem for the charge self-interaction. Techniques such as mehrstellen discretization and τ-extrapolation are used to improve the order of the discretization error. The results show that the expected convergence rates and performance of the multigrid solver are achieved. Within the applied Car-Parrinello molecular dynamics scheme the quality of the solution also determines the accuracy in energy conservation. All forms of discretization employed lead to energy conserving dynamics. In order to test the applicability of our code to larger systems in a massively parallel environment, a 256 atom periodic supercell of bulk gallium nitride is investigated.

6.2. Towards Automatic Generation of Multigrid Solvers

In order to obtain an implementation for ab initio molecular dynamics, one can either describe the whole application on all levels in detail and then generate the full code, or one can build upon existing software and only replace performance critical parts like the multigrid solver by generated code. The latter is usually much easier to do. Note, however, that there is no guarantee that this approach scales very well, since other parts of the existing code can become the bottleneck for performance and portability then.

6.2.1. Abstract Problem Description

As a showcase, I have implemented a first prototype in Scala[2] that is able to generate from abstract descriptions on all levels, depicted in Figure 6.4, CUDA or C++ code for geometric

[2] `http://www.scala-lang.org`

multigrid solvers on regular grids in 2D and 3D. We mainly choose Scala, because of the built-in support for parser combinators [127] and the possibility to do object-oriented and also functional programming, what helps to write code transformations in a compact form. The abstract descriptions are formulated in external domain-specific languages (DSLs) [105], i.e., they are not embedded in another language. Context-sensitive grammars for them can be specified in a notation similar to Extended Backus–Naur Form (EBNF) [104]. Note that only a very limited number of language features is required on most levels except on Level 4.

Although the final DSLs will be most likely similar to LaTeX on Level 1, Matlab[3] on Level 2, XML on Level 3, and a C++-like language on Level 4, we start with a simpler syntax on each level in our first prototype and only provide rudimentary DSLs stored in one file per level.

Continuous problem and domain An example for the description of the continuous problem from Eq. (5.1) on the domain $\Omega = [0, 1]^2$ on Level 1 is found next:

```
Domain d = UnitSquare

Function f = 0
Unknown solution = initrandom
Operator Lapl = Laplacian

PDE pde { Lapl(solution) = f }
PDEBC bc { solution = 0 }

Accuracy = 12
```

On this Level the user is able to name the domain, e.g., changing it to *UnitCube* would result in solving a 3D instead of a 2D problem. *Accuracy = 12* sets the desired mesh size to 2^{-12} in order to determine the number of grid points in each dimension for discretization on Level 2. The solution is initialized randomly. Furthermore, a function can be provided for the right hand side f.

Descriptions of all other levels can be generated automatically from information on Level 1 and internal domain knowledge. However, it is possible to adapt them after the default versions are created.

Discrete problem and domain On Level 2, the description of the default discretization of the above problem is:

```
Fragments f1 = Regular_Square

Discrete_Domain d {
  xsize = 4096
  ysize = 4096
}
```

[3] http://www.mathworks.de

```
field<Double,1>@nodes f
field<Double,1>@nodes solution
field<Double,1>@nodes Res
stencil<Double,FD,2>@nodes Lapl
stencil<Double>@nodes RestrictionStencil
```

A number of decisions had to be made here, either directly based on domain knowledge, or later based on domain-specific optimization. First, the computational domain is partitioned into fragments, then the grid sizes can be derived from the accuracy on Level 1 in this simple case. Further defaults are the underlying data type, here double precision floating point numbers, the number of components per grid point within a field, a node-based location of the grid points, and a second order finite difference discretization. Note that vectors are mapped to fields and sparse matrices to stencils. The restriction stencil is required by the multigrid solver and only declared here, it is defined later on Level 3.

Algorithmic components and parameters On Level 3 mainly default multigrid components and parameters are set (compare to algorithm 5.1). Note that for real-world CSE applications, input parameters and other algorithmic components are also prescribed on this Level.

```
mgcomponents {
  smoother = GaussSeidel
  interpolation = interpolatecorr
  restriction = Restrict
  coarsesolver = GaussSeidel
  cycle = VCycle
}

mgparameter {
  nlevels = 7  // number of multigrid levels
  restr_order = 2 // order of restriction operator
  int_order = 2 // order of interpolation operator
  ncoarse = 10 // number of coarse grid solver iterations
  nprae = 2 // number of pre-smoothing steps
  npost = 1 // number of pre-smoothing steps
  iters = 10 // maximum number of V-cycles
  omega = 1.0 // smoother parameter
}
```

For each of the components the name of a corresponding function in the pseudo code implementation on Level 4 is provided. With the additional information, that the order of the restriction operator is 2, the restriction stencil from Level 2 can now defined to be full weighting, the interpolation to be bilinear. The decisions on this Level can be supported by a performance model including LFA predictions for the multigrid convergence rates and domain knowledge.

Pseudo code implementation Level 4 offers an interface to add algorithms or links to modules from an external framework. After this Level has been generated from previous levels, one is able to formulate own functions and classes in a Scala-like syntax close to concrete code. The application specialist can provide a short description of the application, which will become the main function in the generated code. The CSE expert can add new numerical algorithms, which work on the discrete domain and can access stencils and fields defined on Level 2. One may also prescribe for each function, on which platform it should be executed. Basic language elements can be used, e.g., local variables, statements, expressions, several kinds of loops, and simple I/O like printing to the standard output.

For the above test problem, all multigrid components are automatically generated using domain knowledge. The application is found in the main function, furthermore the V-cycle, the smoother, and the transfer operator functions are shown below.

```
def cpu Application ( ) : Unit
{
  decl res0 : Double = L2Residual ( 0 )
  decl res : Double = res0
  decl resold : Double = 0
  print ( 'startingres' res0 )
  repeat up 10
    resold = res
    VCycle ( 0 )
    res = L2Residual ( 0 )
    print ( 'Residual:' res 'residual reduction:' (res0/res) )
  next
}

def cpu VCycle ( lev:Int  ) : Unit
{
  if coarsestlevel {
    repeat up ncoarse
      GaussSeidel ( lev)
    next
  } else {
    repeat up nprae
      GaussSeidel( lev)
    next
  Residual ( lev )
  Restrict ( (lev+1) f[(lev+1)] Res[lev])
  set( (lev+1) solution[(lev+1)] 0)
  VCycle (lev+1)
  interpolatecorr( lev solution[lev] solution[(lev+1)] )
    repeat up npost
      GaussSeidel ( lev )
    next
  }
```

```
}

def cpu GaussSeidel ( lev:Int ) : Unit
{
    loop innerpoints level lev order rb block 1 1
      solution = solution[lev] + (inverse( diag(Lapl[lev]) ) * omega
           * ( f[lev] - Lapl[lev] * solution[lev] ) )
    next
}

def cpu Restrict ( lev:Int coarse:Array fine:Array) : Unit
{
    loop innerpoints level coarse order lex block 1 1
      coarse = RestrictionStencil * fine | ToCoarse
    next
}

def cpu interpolatecorr( lev:Int uf:Array uc:Array ) : Unit
{
    loop innerpoints level uf order lex block 1 1
      uf += transpose(RestrictionStencil) * uc | ToFine
    next
}
```

Two important language features are the loop concept and the matrix-vector product. The *loop* keyword corresponds to a (nested) for-loop construct, which iterates over parts or the whole discrete computational domain (grid) in a certain manner. The first modifier specifies the part of the grid, currently *allpoints*, *innerpoints*, or *boundarypoints*, *level* on which grid level the loop iterates, *order* the order of traversal through the grid points, and *block*, if a point-wise or block-wise update is done. Inter-grid transfers are triggered by the additional modifiers *ToCoarse* or *ToFine* at the end of the statement within a loop. A matrix-vector (stencil-field) product is usually also found inside a loop over (parts of) the grid, where one requires data from neighboring grid points, when applying the stencil, depending on its shape and size. This is often the most expensive operation and thus has to be implemented efficiently.

What is missing in this example are explicit calls to communication routines necessary for MPI-parallel code. Basically, they are treated just like usual function calls, where the function name is *communicate* and the arguments are the field and the grid level, on which ghost layers should be exchanged. Low level optimizations like vectorization or blocking are not yet supported and will be done internally via code transformations and not specified explicitly in the DSL. Level 4 has to be adapted, if our approach is combined with a new external framework in order to provide an interface to it. In case of waLBerla, Level 4 could be an abstraction of a waLBerla application written in C++ that uses modules and data structures from the framework.

Hardware specification In addition to the problem, the user also lists hardware details in a separate file.

```
Hardware cpu {
  bandwidth = 60
  peak = 118
  cores = 4
}

Node {
  sockets = 1
}

Cluster {
  nodes = 1
  networkbandwidth = 10
}
```

Information about peak performance (in GFLOP/s) and bandwidth (in GB/s) is important for the internal performance model, the type of hardware triggers, if C++ or CUDA code has to be generated. If there is more than one core available on CPU, OpenMP support is enabled, and in case of more than one node, MPI support is enabled.

The features of the prototype that can be set on different levels are summarized in Table 6.1. Currently, we are building up an automatic test suite such that it becomes easier to test all possible feature combinations.

6.2.2. Code Generation Prototype

Structure With the hardware description and information from all levels, one can generate CPU or GPU code as summarized in Figure 6.5. In Figure 6.6 one finds the structure of the pretty-printed source file. The Scala prototype implementation consists of several packages listed in Figure 6.7.

Parsers for all levels are found in the *Parser Package*. The *Generate Package* uses domain knowledge formulated as rules to decide on Level 2, how to discretize the problem, on Level 3, which multigrid components and parameters are suitable, and on Level 4, how the multigrid components are implemented. These rules will be collected in a rule data base later in order to enable an automatic learning process and an easy way to extend the domain knowledge.

The class *DomainKnowledge* stores global data from all levels, global variables, fields, and stencils. It contains, e.g., global rules for determining array sizes and helper functions for index mapping. Furthermore, it defines how stencil operations and boundary conditions are implemented.

Representation of computational domain and discretization A *fragment* represents the geometry of the computational domain. It contains primitive classes, like faces, edges, and vertices. An example for the regular square is found next.

Table 6.1.: A feature model for the first Scala prototype. Blue color denotes that the feature value has not been fully implemented or tested for all feature combinations yet. Features in bold font have to be specified by the application expert, all others can be derived from these.

Feature	Level	Values
Computational domain	1	UnitSquare, UnitCube
Operator	1	Laplacian, ComplexDiffusion
Boundary conditions	1	Dirichlet, Neumann, periodic
Location of grid points	2	node-based, cell-centered
Discretization	2	finite differences, finite elements
Data type	2	single/double accuracy, complex numbers
Multigrid smoother	3	ω-Jacobi, ω-Gauss-Seidel, red-black variants
Multigrid inter-grid transfer	3	constant and linear interpolation and restriction
Multigrid coarsening	3	direct (re-discretization), Galerkin
Multigrid parameters	3	various
Implementation	4	various code optimization strategies
Platform	Hardware	CPU, GPU
Parallelization	Hardware	serial, OpenMP, MPI

```
var fragments: ListBuffer[Fragment] = ListBuffer()

def initfragments() {
  var v: List[Vertex] = List()
  var e: List[Edge] = List()
  var f: List[Face] = List()

  if (DomainKnowledge.fragment_L2.get._2.equals("Regular_Square")) {
    v = List(new Vertex(ListBuffer(0.0, 0.0)), new Vertex(ListBuffer(0.0, 1.0)),
             new Vertex(ListBuffer(1.0, 0.0)), new Vertex(ListBuffer(1.0, 1.0)))
    e = List(new Edge(v(0), v(1)), new Edge(v(0), v(2)),
             new Edge(v(1), v(3)), new Edge(v(2), v(3)))

    f = List(new Face(List(e(0), e(1), e(2), e(3)), v))

    fragments += new Fragment(f, e, v)
  }
}
```

For parallel simulations, a fragment represents a local block, the computational domain is split into fragments of the same shape and size, and each process owns one or several fragments. The handling of boundary conditions and the parallel data exchange via ghost layers is also based on fragments, where a mapping, e.g., for copying data to or from boundary primitives like edges into data buffers, is described by an affine coordinate transformation.

For discretization simple finite differences are supported, the finite element local stiffness matrices can be assembled using the COLSAMM library[4], and for mehrstellen discretization we plan to generate the stencil entries via an approach described in [114].

[4] http://www10.informatik.uni-erlangen.de/People/Alumni/jochen/.www/colsamm.html

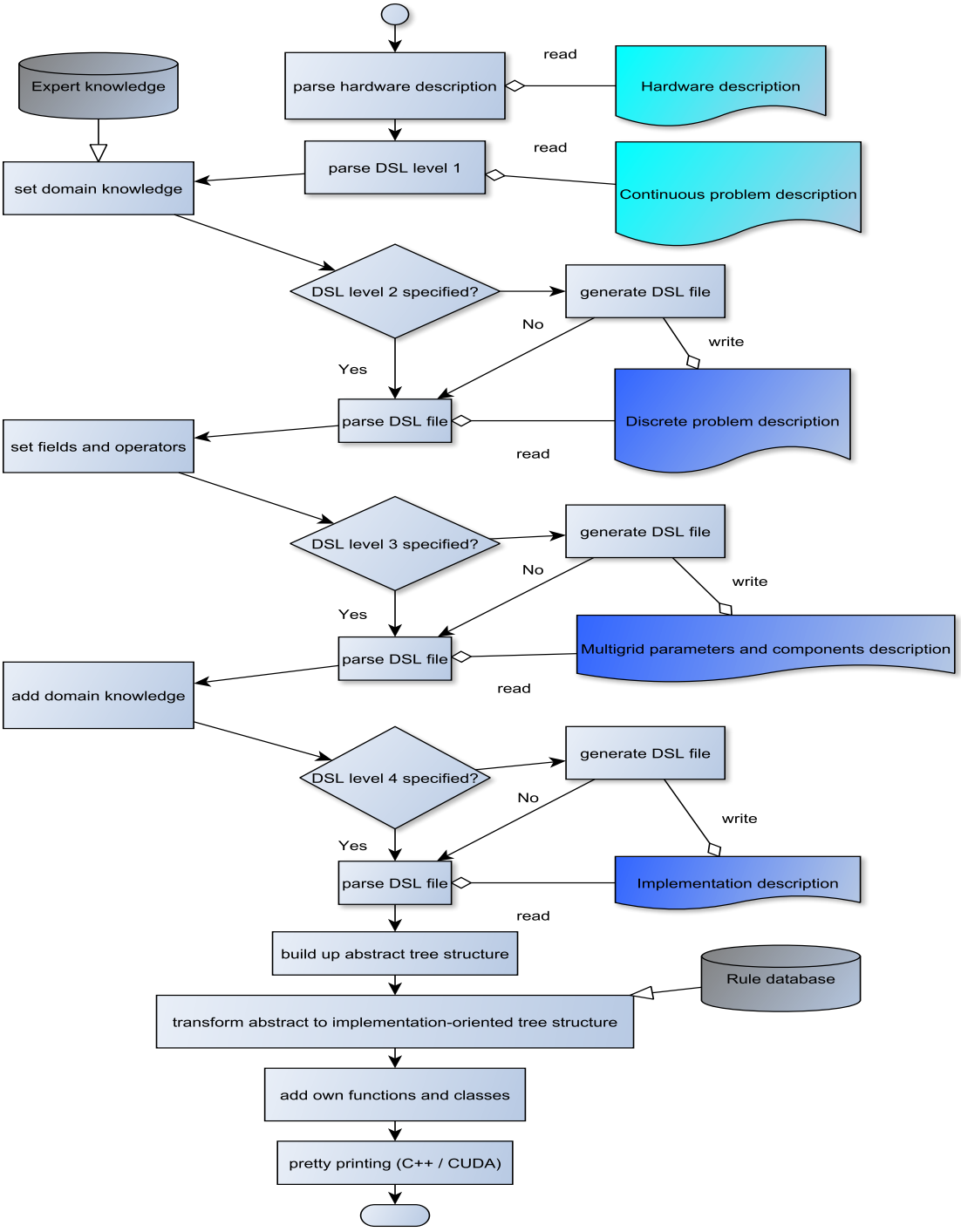


Figure 6.5.: General structure of Scala prototype. DSL descriptions on different levels are parsed, transformed into an implementation-oriented representation, and then pretty-printed to C++ or CUDA source code.

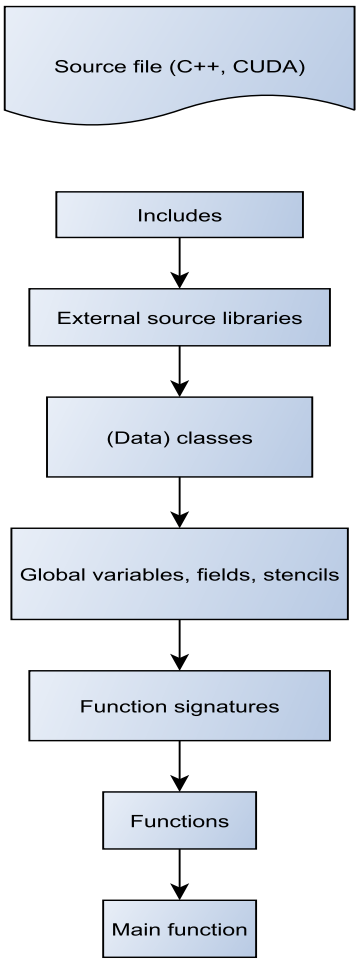


Figure 6.6.: Structure of the pretty-printed source file.

Code Transformations The parsers create for each entry in the abstract description on all levels corresponding objects of one of the classes found in the *Abstract Package*. These objects are transformed into objects of classes from the *Implementation Package* that can already pretty-print themselves to C++ or CUDA code. As an example the Scala classes for fields in both packages are

```
class AbstractField(val name: String, val datatype: String, val location: String)

class ImplField(val name: String, val datatype: String,
                val sizex: Int, val sizey: Int, val sizez: Int, val addpoints: Int)
```

The Scala classes for stencils look similar. For functions and classes it is more complex, here one has to construct an abstract syntax tree (AST) for each of them.

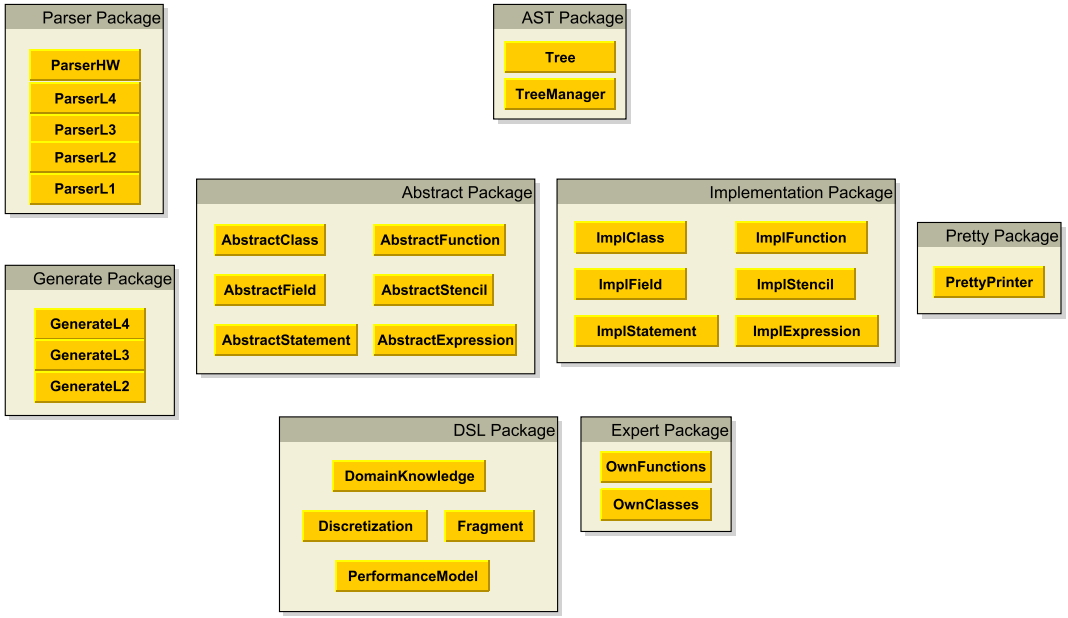


Figure 6.7.: Packages of the Scala prototype.

```
class AbstractFunction(fname: String, location: String, rettype: String,
                       paramlist: List[Param], stmts: List[AbstractStatement]) {
    def transform: ListBuffer[ImplFunction] = { return ... }
}

class ImplFunction(fname: String, location: String, rettype: String,
                   paramlist: ListBuffer[ParameterInfo],
                   bodylist: ListBuffer[ImplStatement]) {
    def toString : String = { return ... }
}
```

Besides the function name, the location, i.e., CPU or GPU, the return type, the list of parameters, and the list of statements in the function body are specified. The routine *transform* models the internal code transformations that involve, in general, information from all levels and domain knowledge. For completeness, in Figure 6.8 all possible abstract statements, and in Figure 6.9 all possible abstract expressions are depicted, which can occur in the AST. The implementation-oriented statements and expressions after code transformation are found in Figure 6.10.

The *AST Package* stores lists of objects for all defined classes, functions, fields, and stencils. Currently, Ch. Schmitt is implementing a more general code transformation framework, where one can apply also external transformations and it is possible to traverse the tree more than once.

In the *Expert Package* predefined classes for fields and stencils are contained including information about data layout and data access, the Stencil class also has member functions for matrix-vector operations, which are inlined later on. Furthermore, there are special functions, e.g., for initializing and cleaning up MPI, OpenMP, or CUDA, allocating fields and stencils, boundary treatment, and copy to or from MPI buffers.

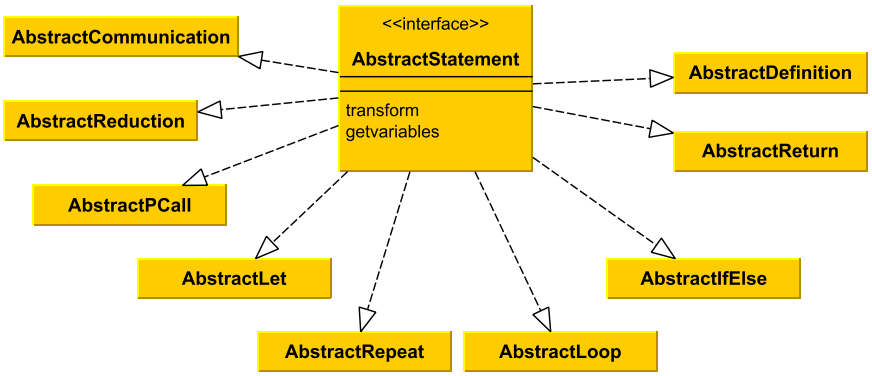


Figure 6.8.: Class hierarchy for abstract statements.

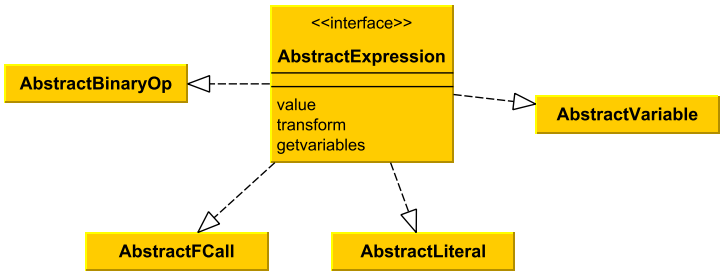


Figure 6.9.: Class hierarchy for abstract expressions.

A routine found in the *Pretty Package* writes the generated source string to a file that can be compiled directly.

GPU Support In principle, an abstract function, specified on Level 4 to be located on GPU, is transformed into a CPU interface function and a GPU kernel function. The first sets up necessary information about GPU kernel block and grid sizes and then calls the GPU kernel. The computations are performed within the GPU kernel. Note that one has to assure that data is also present in GPU memory, i.e., GPU memory must be allocated before on GPU, and data transfers between CPU and GPU via the relatively slow PCIe bus can be necessary. Therefore, it typically makes no sense to switch too often between CPU and GPU kernels, but to transfer data once to GPU at the beginning, then do all computations on GPU, and transfer at the end the results back to CPU.

Performance Modeling Support The Scala prototype also contains concepts for future support of automatic performance modeling (see Figure 6.11). In order to predict runtime during code generation, one can annotate functions, statements, and expressions with static cost information like number of FLOP/s or number of bytes, which have to be loaded from or stored to memory. For a bigger part of the application, e.g., a whole V-cycle, a call graph

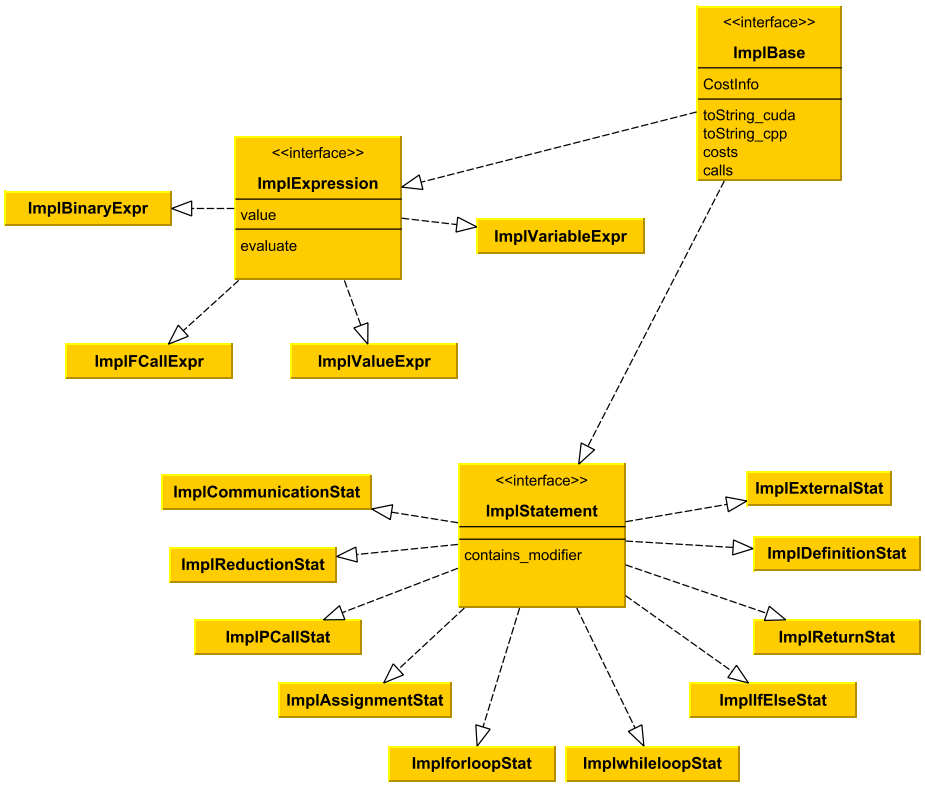


Figure 6.10.: Class hierarchy for implementation-oriented statements and expressions.

can be constructed and static costs of its parts are summed up.

The runtime estimate becomes more accurate, if first computationally expensive parts of the code are generated and then analyzed via external tools like, e.g., likwid[5] or scalasca[6]. The gathered information from performance measurement flows back to the code generator via a more detailed hardware knowledge and additional code transformations. This helps to improve the accuracy of the static cost analysis.

Experimental Results Currently, the Scala prototype is already capable of creating implementations for several different feature combinations, where the problem, the computational domain, and the hardware are varied. As a consequence, discretization, multigrid components and parameters, and thus the overall implementation changes significantly. As test problems Poisson's equation (see Eq.(5.1)) and complex diffusion are considered. Motivated by a simplified time-dependent Schrödinger equation, one can model a nonlinear isotropic complex diffusion process by the time-dependent PDE

$$\operatorname{div}\left(g\left(Im(u)\right)\nabla u\right) \quad = \quad u_t \tag{6.16}$$

[5] `http://code.google.com/p/likwid`
[6] `http://www.scalasca.org`

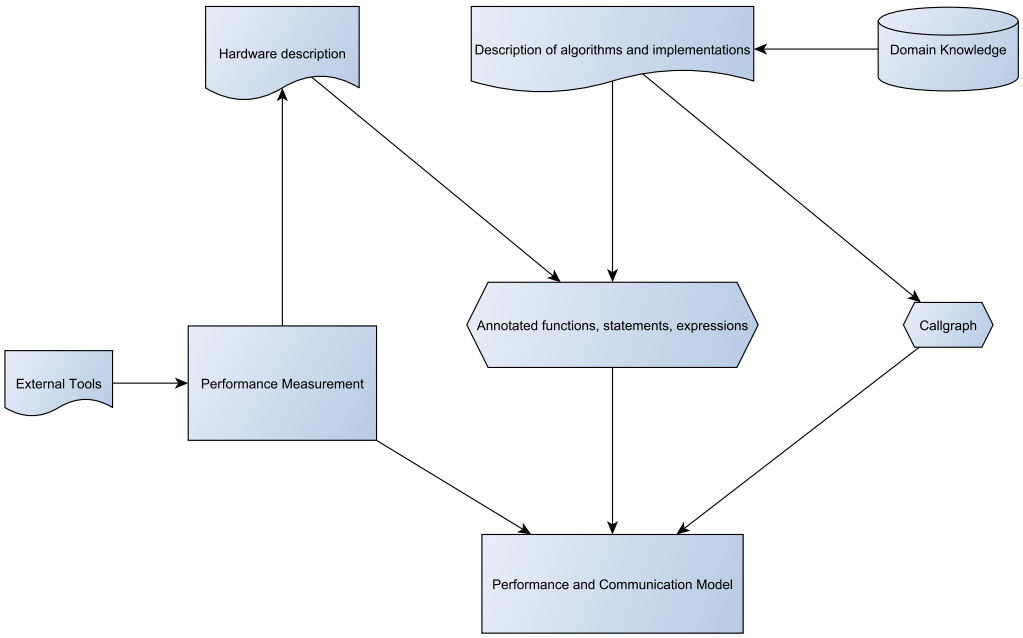


Figure 6.11.: Support for performance modeling.

with Neumann boundary conditions, initial condition $u(0) = u^0$, and time $t = 0$. $Im(u)$ denotes the imaginary part of u and the complex diffusivity function is given by

$$g\left(Im(u)\right) = \frac{e^{i\theta}}{1 + \left(\frac{Im(u)}{k\theta}\right)^2}. \tag{6.17}$$

with small angle θ and scaling parameter $k > 0$. Complex diffusion is used, e.g., for denoising of images [106, 117]. For discretization, averaged finite differences (corresponding to finite volumes) in space and an implicit Euler scheme in time are applied. The arising (non)linear system of equations is solved in each time step via a full approximation (FAS) multigrid scheme [136], where the non-linearity is treated by lagged diffusivity. The transfer operators are cell-centered restriction and constant interpolation. Note that one can change the multigrid V-cycle description on Level 4 to the FAS scheme [24] easily.

Table 6.2 summarizes some preliminary runtime results for generated codes using different feature combinations. The experiments are conducted on a quad-core Intel Xeon Processor E5-1620 v2 running at 3.7 GHz and achieving a maximal peak memory bandwidth of 59.7 GB/s and 118.4 GFLOP/s peak performance, and an NVIDIA GeForce GTX 680 achieving 192.2 GB/s and 3.1 TFLOP/s (single precision) respectively 128.8 GFLOP/s (double precision). As operation system Windows 7 is used, furthermore Visual Studio 2012 and the CUDA 5.5 compilers.

In case of Poisson, the L_2-norm of the residual is reduced by a factor of more than 10^8 within 5 V(2,2)-cycles for all tests, for complex diffusion by a factor of 10^5. Note that the generated code tries to avoid manual problem-specific optimizations as far as possible,

Table 6.2.: Measured runtimes in seconds for one V(2,2)-cycle for different feature combinations. Default settings are a serial run on one CPU core, red-black ω-Gauss-Seidel with $\omega = 1.0$, and double precision. In case of Jacobi smoother $\omega = 0.8$. The number of unknowns is $N = 4095^2$ (node-based) resp. $N = 4096^2$ (cell-centered) in 2D, and $N = 255^3$ in 3D. Direct coarsening down to less than three unknowns per direction on the coarsest grid is applied.

Computational Domain	Platform	PDE	Settings	runtime in s
UnitSquare	CPU	Poisson		0.7
UnitSquare	CPU	Poisson	Jacobi	1.0
UnitSquare	CPU	Poisson	OpenMP 4 threads, Jacobi	0.3
UnitCube	CPU	Poisson		1.1
UnitSquare	GPU	Poisson		0.05
UnitSquare	CPU	ComplexDiffusion	Jacobi	32.1
UnitSquare	CPU	ComplexDiffusion	OpenMP 4 threads, Jacobi	12.8
UnitSquare	GPU	ComplexDiffusion	single precision, Jacobi	0.09

because the selection of suitable code transformations to obtain the most efficient implementation will be part of the global feature value optimization problem. Complex diffusion on CPU is especially slow, where *std::complex<double>* from the C++ standard library is used as built-in data type. For the GPU version a field with two components, one for the real and one for the imaginary part, is allocated and own device functions for complex arithmetic are provided. A hand-tuned implementation for 2D Poisson takes about 0.008 s on GPU in single precision [12], with the same settings a hand-tuned complex diffusion takes about 0.037 s. Optimizations in these implementations include, e.g., vectorization, change of data layout, simplification of index calculations and other computations, loop unrolling, and blocking techniques.

Tuning of Algorithmic Components and Parameters As already mentioned, A. Grebhahn deals with domain-specific optimization of algorithmic components and parameters within ExaStencils [33]. Once the code generation framework is completed, it will be possible to generate thousands of different multigrid implementations already out of feature combinations listed in Table 6.1. The global optimization problem to find the best implementation for a certain application with respect to accuracy of the solution and time to compute the solution is in general infeasible to solve, of course. Our approach will be based on the feature model and first define, which parts of the optimization problem involve functional quantities, e.g., optimal multigrid parameters, and which parts involve non-functional quantities, e.g., the type of discretization or smoother. One idea is to optimize functional quantities based on performance models, LFA predictions, domain knowledge, and sample measurements. The resulting accuracy of the solution together with runtime and convergence rate estimates or measurements can be used as fitness function within a multi-objective heuristic algorithm that optimizes (non-functional) feature strings.

Summary: A Multi-objective Genetic Algorithm for Build Order Optimization in StarCraft II [8] This article presents a modified version of the multi-objective genetic algorithm NSGA II [97] in order to find optimal opening strategies in the real-time strategy game StarCraft® II: Wings of Liberty™ published by Blizzard Entertainment[7]. Based on an event-driven simulator capable of performing an accurate estimate of in-game construction times the quality of different build lists can be judged. These build lists are used as chromosomes within the genetic algorithm. Procedural constraints, e.g., given by the dependencies of build orders or other game mechanisms, are implicitly encoded into them. Typical goals are to find the build list producing most units of one or more certain types up to a certain time (Rush) or to produce one unit as early as possible (Tech-Push). Here, the number of entries in a build list varies and the objective values have in contrast to the search space a very small diversity. The proposed algorithm is tested on different Tech-Pushes and Rushes for all three races, and validated with empirical data of expert StarCraft II players.

Clearly, the above work is far off the traditional CSE applications with respect to the used algorithms. But it performs a simulation of the game, where in each time step the player chooses his next action that has to fulfill several constraints. The same now holds for the CSE expert when developing code. On each Level several choices have to be made under certain restrictions and earlier decisions influence later ones. This domain knowledge about feature combinations and interactions can be extended by machine learning techniques in order to reduce the global search space for the multi-objective genetic algorithm.

[7] ©2010 Blizzard Entertainment, Inc. All rights reserved. Wings of Liberty is a trademark, and StarCraft and Blizzard Entertainment are trademarks or registered trademarks of Blizzard Entertainment, Inc. in the U.S. and/or other countries.

7. Conclusions and Future Work

Researchers in the field of computational science and engineering require knowledge in computer science, applied mathematics, and at least one of the various application fields. Central for CSE are efficient (parallel) algorithms. The types of algorithms are not limited to the ones listed in chapter 3, but CSE methods can also be applied to other kinds of algorithms.

Besides classical iterative algorithms like multigrid, which can be used to solve PDEs numerically, optimization algorithms, statistical algorithms, and data processing algorithms are important within CSE applications (see figure 3.3). Statistics helps to deal with uncertainty in models and data. Furthermore, handling and processing the massive amount of input and output data with larger and larger HPC clusters is a big issue currently. Additional challenges arise with increasing energy consumption and growing heterogeneity of the platforms. Code complexity also increases, because in many real-world applications several physical models are coupled.

One way to address some of these issues from the software side is code generation. It is clear that a tailored implementation for a specific application usually fits all its needs, but very often it cannot be extended easily to new models or hardware. Thus, since hundreds of similar implementations are not feasible, the typical solution is to use abstractions, e.g., generic programming, and to provide libraries or frameworks. This common approach works quite well, as long as the class of applications is not too diverse, and the implementation details like data structures or parallelization strategies do not vary too much. Code generation makes it possible to create many platform-specific implementations, which are usually shorter and can be optimized easier by standard compilers. However, it is absolutely necessary to provide automatic tests for the generated codes in order to ensure maintainability. Another advantage of code generation is that it can be combined with automatic tuning, where one wants to find the best possible implementation for a specific problem based on a feature model, which includes feature constraints derived from domain knowledge.

The waLBerla framework will benefit from our code generation framework, e.g., by adding automatically new LBM kernels, performing parameter tuning, or providing a DSL as application interface. This has the advantage that users have to learn less details about waLBerla software concepts, before they can start to port their own application.

As was shown, my work focuses on algorithm and software development for CSE applications running on current HPC platforms. While performance was always the most important software quality factor for large-scale simulations, with growing model and thus code complexity and arising new (parallel) programming paradigms, portability and productivity become more and more important, too. In the next years, a joint effort from application, hardware, and software side will be necessary to reach the desired exa-scale performance for real-world CSE applications.

Journal Publications

[1] C. Feichtinger, S. Donath, H. Köstler, J. Götz, and U. Rüde, *WaLBerla: HPC Software Design for Computational Engineering Simulations*, Journal of Computational Science **2** (2011), 105–112.

[2] C. Feichtinger, J. Habich, H. Köstler, G. Hager, U. Rüde, and G. Wellein, *A Flexible Patch-Based Lattice Boltzmann Parallelization Approach for Heterogeneous GPU-CPU Clusters*, Journal of Parallel Computing **37** (2011), 536–549.

[3] C. Feichtinger, H. Köstler, J. Habich, T. Aoki, and U. Rüde, *Performance Modeling and Analysis of Heterogeneous Lattice Boltzmann Simulations on CPU-GPU Clusters*, Journal of Parallel Computing (2013), submitted.

[4] B. Gmeiner, H. Köstler, M. Stürmer, and U. Rüde, *Parallel multigrid on hierarchical hybrid grids: a performance study on current high performance computing clusters*, Concurrency and Computation: Practice and Experience (2012), 1–24.

[5] J. Habich, C. Feichtinger, H. Köstler, G. Hager, and G. Wellein, *Performance engineering for the Lattice Boltzmann method on GPGPUs: Architectural requirements and performance results*, Computers & Fluids (2012).

[6] J. Han, C. Bennewitz, H. Köstler, J. Hornegger, and T. Kuwert, *Computer-Aided Validation of Hybrid SPECT/CT Scanners*, Computerized Medical Imaging and Graphics **32** (2008), no. 5, 388–395.

[7] E. M. Kalmoun, H. Köstler and U. Rüde, *3D optical flow computation using a parallel variational multigrid scheme with application to cardiac C-arm CT motion*, Image and Vision Computing **25** (2007), no. 9, 1482–1494.

[8] H. Köstler and B. Gmeiner, *A Multi-objective Genetic Algorithm for Build Order Optimization in StarCraft II*, KI - Künstliche Intelligenz **27** (2013), no. 3, 221–233 (English).

[9] H. Köstler and U. Rüde, *An accurate multigrid solver for computing singular solutions of elliptic problems*, Numerical Linear Algebra with Applications **13** (2006), no. 2-3, 231–249.

[10] H. Köstler and U. Rüde, *The CSE software challenge – covering the complete stack*, it-Information Technology **55** (2013), no. 3, 91–96.

[11] H. Köstler, R. Schmid, U. Rüde, and Ch. Scheit, *A parallel multigrid accelerated Poisson solver for ab initio molecular dynamics applications*, Computing and Visualization in Science **11** (2008), 115–122.

[12] H. Köstler, M. Stürmer, and T. Pohl, *Performance engineering to achieve real-time high dynamic range imaging*, Journal of Real-Time Image Processing (2013), 1–13.

[13] H. Köstler, M. Stürmer, and U. Rüde, *A fast full multigrid solver for applications in image processing*, Numerical Linear Algebra with Applications **15** (2008), no. 2–3, 187–200.

[14] S. Kuckuk, T. Preclik, and H. Köstler, *Interactive particle dynamics using Opencl and Kinect*, International Journal of Parallel, Emergent and Distributed Systems **28** (2013), no. 6, 519–536.

[15] R. Membarth, O. Reiche, Ch. Schmitt, F. Hannig, J. Teich, M. Stürmer, and H. Köstler, *Towards a performance-portable description of geometric multigrid algorithms using a domain-specific language*, Journal of Parallel and Distributed Computing (2013), submitted.

[16] C. Popa, T. Preclik, H. Köstler, and U. Rüde, *On Kaczmarz's projection iteration as a direct solver for linear least squares problems*, Linear Algebra and its Applications **436** (2011), no. 2, 389–404.

[17] K. Ruhnau, H. Köstler, and R. Wienands, *A multigrid method for the computation of the optical flow using a curvature based regularizer*, Numerical Linear Algebra with Applications **15** (2008), no. 2–3, 201–218.

[18] R. Schmid, M. Tafipolsky, P. König, and H. Köstler, *Car-Parrinello molecular dynamics using real space wavefunctions*, Physica status solidi. B. Basic research **243** (2006), no. 5, 1001–1015.

[19] M. Stürmer, J. Dagner, P. Manstetten, and H. Köstler, *Real-time simulation of temperature in hot rolling rolls*, Journal of Computational Science (2013), submitted.

[20] R. Wienands and H. Köstler, *A practical framework for the construction of prolongation operators for multigrid based on canonical basis functions*, Computing and visualization in science **13** (2010), 207–220.

[21] C.H. Wolters, H. Köstler, C. Möller, J. Härdtlein, L. Grasedyck, and W. Hackbusch, *Numerical Mathematics of the Subtraction Method for the Modeling of a Current Dipole in EEG Source Reconstruction Using Finite Element Head Models*, SIAM J. on Scientific Computing **30** (2007), no. 1, 24–45.

Conference Publications

[22] D. Bartuschat, M. Stürmer, and H. Köstler, *An orthogonal matching pursuit algorithm for image denoising on the cell broadband engine*, Parallel Processing and Applied Mathematics (PPAM), Lecture Notes in Computer Science, vol. 6067, Springer-Verlag, Berlin, Heidelberg, New York, 2010, pp. 557–566.

[23] I. Christadler, H. Köstler, and U. Rüde, *Robust and efficient multigrid techniques for the optical flow problem using different regularizers*, Proceedings of 18th Symposium Simulationstechnique ASIM 2005 (F. Hülsemann, M. Kowarschik, and U. Rüde, eds.), Frontiers in Simulation, vol. 15, SCS Publishing House, Erlangen, Germany, 2005, pp. 341–346.

[24] I. Dietrich, R. German, H. Köstler, and U. Rüde, *Modeling multigrid algorithms for variational imaging*, Proceedings of 21st Australian Software Engineering Conference (ASWEC2010), IEEE Computer Society Washington, DC, USA, 2010, pp. 224–234.

[25] U. Fabricius, C. Freundl, H. Köstler, and U. Rüde, *High performance computing education for students in computational engineering*, Computational Science - ICCS 2005 (V.S. Sunderam, G.D.v. Albada, P.M.A. Sloot, and J.J. Dongarra, eds.), Lecture Notes in Computer Science, vol. 3515, Springer-Verlag, Berlin, Heidelberg, New York, 2005, pp. 27–35.

[26] C. Freundl and H. Köstler, *Using ParExPDE for the numerical solution of bioelectric field problems*, Proceedings of 18th Symposium Simulationstechnique ASIM 2005 (F. Hülsemann, M. Kowarschik, and U. Rüde, eds.), Frontiers in Simulation, vol. 15, SCS Publishing House, Erlangen, Germany, 2005, pp. 89–94.

[27] C. Freundl, H. Köstler, and U. Rüde, *Teaching the foundations of computational science on the undergraduate level*, Computational Science - ICCS 2006: 6th International Conference. Proceedings, Part II (Reading, UK) (M.A. Sloot and J. Dongarra, eds.), Lecture Notes in Computer Science, vol. 3992, Springer-Verlag, Berlin, Heidelberg, New York, 2006, pp. 185–192.

[28] B. Gmeiner, G. Donnert, and H. Köstler, *Optimizing opening strategies in a real-time strategy game by a multi-objective genetic algorithm*, Research and Development in Intelligent Systems XXIX (2012), 361–374.

[29] B. Gmeiner, T. Gradl, H. Köstler, and U. Rüde, *Highly parallel geometric multigrid algorithm for hierarchical hybrid grids*, Proceedings NIC Symposium (2012), 323–330 (NIC Series).

[30] Ch. Godenschwager, F. Schornbaum, M. Bauer, H. Köstler, and U. Rüde, *A framework for hybrid parallel flow simulations with a trillion cells in complex geometries*, Proceedings of SC13: International Conference for High Performance Computing, Networking, Storage and Analysis, ACM, 2013.

[31] J. Götz, S. Donath, C. Feichtinger, K. Iglberger, H. Köstler, and U. Rüde, *walberla: Simulation of complex flows on supercomputers*, Proceedings NIC Symposium (2012), 349–356 (NIC Series).

[32] T. Gradl, C. Freundl, H. Köstler, and U. Rüde, *Scalable Multigrid*, High Performance Computing in Science and Engineering. Garching/Munich 2007 (S. Wagner, M. Steinmetz, A. Bode, and M. Brehm, eds.), LRZ, KONWIHR, Springer-Verlag, Berlin, Heidelberg, New York, 2008, pp. 475–483.

[33] A. Grebhahn, N. Siegmund, S. Apel, S. Kuckuk, Ch. Schmitt, and H. Köstler, *Optimizing performance of stencil code via parameter interaction detection*, 9th International Conference on High-Performance and Embedded Architectures and Compilers (HiPEAC) (2013), accepted.

[34] H. Köstler, C. Feichtinger, U. Rüde, and T. Aoki, *A Geometric Multigrid Solver on Tsubame 2.0*, Proceedings Dagstuhl Seminar (2013), in print.

[35] H. Köstler, C. Popa, M. Prümmer, and U. Rüde, *Algebraic Full Multigrid in Image Reconstruction*, Mathematical Modelling of Environmental and Life Sciences Problems. Proceedings of the fifth workshop. September, 2006, Constanta, Romania (S. Ion, G. Marinoschi, and C. Popa, eds.), Editura Academiei Romane, 2008, pp. 123–130.

[36] H. Köstler, C. Popa, U. Rüde, and M. Prümmer, *Towards an algebraic multigrid method for tomographic image reconstruction – improving convergence of ART*, European Conference on Computational Fluid Dynamics (ECCOMAS CFD) (TU Delft, Egmond aan Zee, The Netherlands) (P. Wesseling, E. Onate, and J. Périaux, eds.), 2006.

[37] H. Köstler, D. Ritter, and C. Feichtinger, *A Geometric Multigrid Solver on GPU Clusters*, GPU Solutions to Multi-scale Problems in Science and Engineering (David A. Yuen, Long Wang, Xuebin Chi, Lennart Johnsson, Wei Ge, and Yaolin Shi, eds.), Lecture Notes in Earth System Sciences, Springer-Verlag, Berlin, Heidelberg, New York, 2013, pp. 407–422.

[38] H. Köstler and U. Rüde, *Accurate techniques for computing singular solutions of elliptic problems*, Computational Science - Proceedings ICCS 2004: 4th International Conference, Part IV, Krakow, Poland (M. Bubak, G. v. Albada, and J. Dongarra, eds.), Lecture Notes in Computer Science, vol. 3039, Springer-Verlag, Berlin, Heidelberg, New York, 2004, pp. 410–417.

[39] H. Köstler, U. Rüde, M. Prümmer, and J. Hornegger, *Adaptive variational sinogram interpolation of sparsely sampled CT data*, Proceedings of the 18th International Conference on Pattern Recognition (ICPR), Hongkong, China **3** (2006), 778–781.

[40] S. Kuckuk, B. Gmeiner, H. Köstler, and U. Rüde, *A generic prototype to benchmark algorithms and data structures for hierarchical hybrid grids*, Advances of Parallel Computing (2013), accepted.

[41] M. Mayer, A. Borsdorf, H. Köstler, J. Hornegger, and U. Rüde, *Nonlinear Diffusion vs. Wavelet Based Noise Reduction in CT Using Correlation Analysis*, Proceedings of Vision, Modeling and Visualization (Saarbrücken, Germany), Aka GmbH, IOS Press, 2007, pp. 223–232.

[42] ———, *Nonlinear Diffusion Noise Reduction in CT Using Correlation Analysis*, 3rd Russian-Bavarian Conference on Biomedical Engineering (Erlangen, Germany) (J. Hornegger, E. Mayr, S. Schookin, H. Feußner, N. Navab, Y. Gulyaev, K. Höller, and V. Ganzha, eds.), vol. 1, Union aktuell, 2008, pp. 155–159.

[43] R. Membarth, F. Hannig, J. Teich, and H. Köstler, *Towards Domain-specific Computing for Stencil Codes in HPC*, Proceedings of the 2nd International Workshop on Domain-Specific Languages and High-Level Frameworks for High Performance Computing (WOLFHPC) (Salt Lake City, UT, USA), IEEE, 2012.

[44] K. Pickl, M. Hofmann, T. Preclik, H. Köstler, A. Smith, and U. Rüde, *Parallel simulations of self-propelled microorganisms*, Advances of Parallel Computing (2013), accepted.

[45] M. Prümmer, H. Köstler, U. Rüde, and J. Hornegger, *A full multigrid technique to accelerate an ART scheme for tomographic image reconstruction*, Proceedings of 18th Symposium Simulationstechnique ASIM 2005 (F. Hülsemann, M. Kowarschik, and U. Rüde, eds.), Frontiers in Simulation, vol. 15, SCS Publishing House, Erlangen, Germany, 2005, pp. 532–537.

[46] O. Röhrle, H. Köstler, and M. Loch, *Segmentation of skeletal muscle fibers for applications in computational skeletal muscle mechanics*, Computational Biomechanics for Medicine, Springer-Verlag, Berlin, Heidelberg, New York, 2011, pp. 107–117.

[47] M. Stürmer, H. Köstler, and F. Rathgeber, *Performance engineering of an orthogonal matching pursuit algorithm for sparse representation of signals on different architectures*, High-performance and Hardware-aware Computing (HipHaC'11), San Antonio, Texas, USA (R. Buchty and J.-P. Weiß, eds.), KIT Scientific Publishing, 2011, pp. 17–24.

[48] M. Stürmer, H. Köstler, and U. Rüde, *Fast wavelet transform utilizing a multicore-aware framework*, Applied Parallel and Scientific Computing (K. Jonasson, ed.), Lecture Notes in Computer Science, vol. 7134, Springer-Verlag, Berlin, Heidelberg, New York, 2012, pp. 313–323.

[49] M. Stürmer, D. Ritter, H. Köstler, and U. Rüde, *Experiences with Numerical Codes on the Cell Broadband Engine Architecture*, High performance and hardware aware computing (HipHaC'08), Lake Como, Italy (R. Buchty and J.-P. Weiß, eds.), KIT Scientific Publishing, 2008, pp. 9–16.

[50] M. Stürmer, G. Wellein, G. Hager, H. Köstler, and U. Rüde, *Challenges and Potentials of Emerging Multicore Architectures*, High Performance Computing in Science and Engineering. Garching/Munich 2007 (S. Wagner, M. Steinmetz, A. Bode, and M. Brehm, eds.), LRZ, KONWIHR, Springer-Verlag, Berlin, Heidelberg, New York, 2008, pp. 551–566.

[51] C. Wolters, H. Köstler, C. Möller, J. Härdtlein, and A. Anwander, *Numerical approaches for dipole modeling in finite element method based source analysis*, International Congress Series, BIOMAG2006, 15th Int. Conf. on Biomagnetism, vol. 1300, Elsevier Science Publishers, Amsterdam, The Netherlands, 2007, pp. 189–192.

[52] Y. Zheng, H. Köstler, N. Thürey, and U. Rüde, *Enhanced Motion Blur Calculation with Optical Flow*, Proceedings of Vision, Modeling and Visualization (RWTH Aachen, Germany), Aka GmbH, IOS Press, 2006, pp. 253–260.

Theses and Other Publications

[53] D. Bartuschat, A. Borsdorf, H. Köstler, R. Rubinstein, and M. Stürmer, *A parallel K-SVD implementation for CT image denoising*, Tech. Report 09-1, Department of Computer Science 10 (System Simulation), Friedrich-Alexander-Universität Erlangen-Nürnberg, Germany, 2009.

[54] B. Gmeiner, H. Köstler, and U. Rüde, *Wie viel Unbekannte hat das grösste Gleichungssystem, das man heute lösen kann?*, GAMM Rundbrief (2013), 18–23.

[55] E. M. Kalmoun, H. Köstler and U. Rüde, *Parallel multigrid computation of the 3D optical flow*, Tech. Report 04-4, Department of Computer Science 10 (System Simulation), Friedrich-Alexander-Universität Erlangen-Nürnberg, Germany, 2004.

[56] H. Köstler, *Analyse von eukaryontischen Promotorregionen mit exhaustiver Suche*, Studienarbeit, Department of Computer Science 5 (Pattern recognition), Friedrich-Alexander-Universität Erlangen-Nürnberg, Germany, 2001.

[57] ———, *Akkurate Behandlung von Singularitäten bei partiellen Differentialgleichungen*, Master's thesis, Department of Computer Science 10 (System Simulation), Friedrich-Alexander-Universität Erlangen-Nürnberg, Germany, 2003.

[58] ———, *Der künstliche Autodidakt: Informationsbedarf einer Wissensbasis*, Master's thesis, Betriebswirtschaftslehre, Operation Research, Fernuniversität Hagen, Germany, 2003.

[59] ———, *A Multigrid Framework for Variational Approaches in Medical Image Processing and Computer Vision*, Verlag Dr. Hut, München, 2008.

[60] ———, *Multigrid HowTo: A simple Multigrid solver in C++ in less than 200 lines of code*, Tech. Report 08-3, Department of Computer Science 10 (System Simulation), Friedrich-Alexander-Universität Erlangen-Nürnberg, Germany, 2008.

[61] H. Köstler, C. Möller, and F. Deserno, *Performance Results for Optical Flow on an Opteron Cluster Using a Parallel 2D/3D Multigrid Solver*, Tech. Report 06-5, Department of Computer Science 10 (System Simulation), Friedrich-Alexander-Universität Erlangen-Nürnberg, Germany, 2006.

[62] H. Köstler, C. Popa, and U. Rüde, *Algebraic multigrid for general inconsistent linear systems: The correction step*, Tech. Report 06-4, Department of Computer Science 10 (System Simulation), Friedrich-Alexander-Universität Erlangen-Nürnberg, Germany, 2006.

[63] H. Köstler, M. Stürmer, Ch. Freundl, and U. Rüde, *PDE based Video Compression in Real-Time*, Tech. Report 07-11, Department of Computer Science 10 (System Simulation), Friedrich-Alexander-Universität Erlangen-Nürnberg, Germany, 2007.

[64] S. Kuckuk and H. Köstler, *A Framework for Interactive Physical Simulations on Remote HPC Clusters*, Tech. Report CS-2013-6, Universitätsbibliothek der Universität Erlangen-Nürnberg, Universitätsstr. 4, 91054 Erlangen, 2013.

[65] P. Münch and H. Köstler, *Videocoding using a variational approach for decompression*, Tech. Report 07-1, Department of Computer Science 10 (System Simulation), Friedrich-Alexander-Universität Erlangen-Nürnberg, Germany, 2007.

[66] T. Preclik and H. Köstler, *Multigrid HowTo: An Open Source Algebraic Multigrid Solver in C++*, Tech. Report 09-2, Department of Computer Science 10 (System Simulation), Friedrich-Alexander-Universität Erlangen-Nürnberg, Germany, 2009.

[67] L. Renker and H. Köstler, *Multikulturalität in Studiengängen - Chancen und Herausforderungen für die Teamarbeit*, Tech. Report CS-2012-3, Universitätsbibliothek der Universität Erlangen-Nürnberg, Universitätsstr. 4, 91054 Erlangen, 2012.

Supervised Theses (since 2010)

[68] S. Alassi, *Estimating Blood Flow Based on 2D Angiographic Image Sequences*, Master's thesis, Department of Computer Science 10 (System Simulation), Friedrich-Alexander-Universität Erlangen-Nürnberg, Germany, 2012, Betr. Kowarschik, Pohl, Köstler.

[69] H. Attar, *Simulation of heat-induced elastic deformation of cylindrical-shaped bodies*, Master's thesis, Department of Computer Science 10 (System Simulation), Friedrich-Alexander-Universität Erlangen-Nürnberg, Germany, 2010, Betr. Pickl, Köstler, Rüde.

[70] M. Bauer, *Special Finite Elements for Dipole Modelling*, Master's thesis, Department of Computer Science 10 (System Simulation), Friedrich-Alexander-Universität Erlangen-Nürnberg, Germany, 2012, Betr. Köstler, Rüde.

[71] M. Grießinger, *3D Bidomain Equation for Muscle Fibers*, Master's thesis, Department of Computer Science 10 (System Simulation), Friedrich-Alexander-Universität Erlangen-Nürnberg, Germany, 2011, Betr. Köstler.

[72] M. Hofmann, *Parallelisation of Swimmer Models for the Simulation of Swarms of Bacteria in the Physics Engine pe*, Master's thesis, Department of Computer Science 10 (System Simulation), Friedrich-Alexander-Universität Erlangen-Nürnberg, Germany, 2013, Betr. Pickl, Preclik, Köstler.

[73] T. Kluge, *Development and GPU-based implementation of a complex anisotropic nonlinear diffusion filter for 3D CTA image preprocessing*, Master's thesis, Department of Computer Science 5 (Pattern recognition), Friedrich-Alexander-Universität Erlangen-Nürnberg, Germany, 2012, Betr. Hornegger, Kollorz, Köstler, Bernhardt.

[74] S. Kuckuk, *Visualization and Interactivity for Physics Engines in Real Time*, Master's thesis, Department of Computer Science 10 (System Simulation), Friedrich-Alexander-Universität Erlangen-Nürnberg, Germany, 2013, Betr. Köstler.

[75] M. Loch, *Segmentation of skeletal muscle fibers*, Master's thesis, Department of Computer Science 10 (System Simulation), Friedrich-Alexander-Universität Erlangen-Nürnberg, Germany, 2010, Betr. Köstler.

[76] P. Manstetten, *Real-time Simulation of Heat Conduction in Rolls with Internal Cooling*, Master's thesis, Department of Computer Science 10 (System Simulation), Friedrich-Alexander-Universität Erlangen-Nürnberg, Germany, 2012, Betr. Stürmer, Köstler.

[77] S. Tatavarty, *Accelerating Image Registration on GPUs*, Master's thesis, Department of Computer Science 10 (System Simulation), Friedrich-Alexander-Universität Erlangen-Nürnberg, Germany, 2010, Betr. Köstler.

[78] Z. Wang, *GPU implementation of Free Surface Lattice Boltzmann code*, Studienarbeit, Department of Computer Science 10 (System Simulation), Friedrich-Alexander-Universität Erlangen-Nürnberg, Germany, 2011, Betr. Donath, Köstler, Rüde.

Bibliography

[79] K. Asanovic, R. Bodik, J. Demmel, T. Keaveny, K. Keutzer, J. Kubiatowicz, N. Morgan, D. Patterson, K. Sen, J. Wawrzynek, D. Wessel, and K. Yelick, *A view of the parallel computing landscape*, Commun. ACM **52** (2009), no. 10, 56–67.

[80] C. Baillie, J. McWilliams, J. Weiss, and I. Yavneh, *Implementation and Performance of a Grand Challenge 3d Quasi-Geostrophic Multi-Grid code on the Cray T3D and IBM SP2.*, Proc. Supercomputing, 1995.

[81] D. Barkai and A. Brandt, *Vectorized multigrid Poisson solver for the CDC Cyber 205*, Applied Mathematics and Computation **13** (1983), no. 3, 215–227.

[82] D. Bartuschat, D. Ritter, and U. Rüde, *Parallel multigrid for electrokinetic simulation in particle-fluid flows*, High Performance Computing and Simulation (HPCS), 2012 International Conference on, IEEE, 2012, pp. 374–380.

[83] P. Bastian, K. Birken, K. Johannsen, S. Lang, V. Reichenberger, Ch. Wieners, G. Wittum, and Ch. Wrobel, *A parallel software-platform for solving problems of partial differential equations using unstructured grids and adaptive multigrid methods*, High Performance Computing in Science and Engineering, Springer, 1999, pp. 326–339.

[84] B. Bergen, *Hierarchical hybrid grids: Data structures and core algorithms for efficient finite element simulations on supercomputers*, SCS Publishing House eV, 2006.

[85] B. Bergen and F. Hülsemann, *Hierarchical hybrid grids: A framework for efficient multigrid on high performance architectures* , Tech. Report 03-5, Department of Computer Science 10 (System Simulation), Friedrich-Alexander-Universität Erlangen-Nürnberg, Germany, 2003.

[86] S. Bogner, *Simulation of Floating Objects in Free-Surface Flow*, Master's thesis, Department of Computer Science 10 (System Simulation), Friedrich-Alexander-Universität Erlangen-Nürnberg, Germany, 2009.

[87] S. Bogner and U. Rüde, *Simulation of floating bodies with the lattice Boltzmann method*, Computers & Mathematics with Applications **65** (2013), no. 6, 901–913.

[88] M. Born and R. Oppenheimer, *Zur Quantentheorie der Molekeln*, Annalen der Physik **389** (1927), no. 20, 457–484.

[89] E. Briggs, D. Sullivan, and J. Bernholc, *Real-space multigrid-based approach to large-scale electronic structure calculations*, Physical Review B **54** (1996), no. 20, 14362.

[90] W. Briggs, V. Henson, and S. McCormick, *A multigrid tutorial*, 2nd ed., Society for Industrial and Applied Mathematics (SIAM), Philadelphia, PA, USA, 2000.

[91] A. Bruhn, Th. Pock, and X. Tai, *Efficient Algorithms for Global Optimisation Methods in Computer Vision (Dagstuhl Seminar 11471)*, Dagstuhl Reports **1** (2012), no. 11, 66–90.

[92] C. Burstedde, L. Wilcox, and O. Ghattas, *p4est: Scalable algorithms for parallel adaptive mesh refinement on forests of octrees*, SIAM Journal on Scientific Computing **33** (2011), no. 3, 1103–1133.

[93] R. Car and M. Parrinello, *Unified approach for molecular dynamics and density-functional theory*, Physical Review Letters **55** (1985), 2471–2474.

[94] D. Ceperley and B. Alder, *Ground state of the electron gas by a stochastic method*, Physical Review Letters **45** (1980), 566–569.

[95] S. Chen and G. Doolen, *Lattice boltzmann method for fluid flows*, Annual review of fluid mechanics **30** (1998), no. 1, 329–364.

[96] O. Cohen, L. Kronik, and A. Brandt, *Locally Refined Multigrid Solution of the All-Electron Kohn–Sham Equation*, Journal of Chemical Theory and Computation **9** (2013), no. 11, 4744–4760.

[97] K. Deb, A. Pratap, S. Agarwal, and T. Meyarivan, *A fast and elitist multiobjective genetic algorithm: NSGA-II*, Evolutionary Computation, IEEE Transactions on **6** (2002), no. 2, 182–197.

[98] P. Dirac, *A new notation for quantum mechanics*, Proceedings of the Cambridge Philosophical Society, vol. 35, Cambridge Univ Press, 1939, pp. 416–418.

[99] S. Donath, *Wetting models for a parallel high-performance free surface lattice boltzmann method*, Verlag Dr. Hut, München, 2011.

[100] S. Donath, C. Feichtinger, T. Pohl, J. Götz, and U. Rüde, *Localized Parallel Algorithm for Bubble Coalescence in Free Surface Lattice-Boltzmann Method*, Lecture Notes in Computer Science, Euro-Par 2009, vol. 5704, Springer, 2009, pp. 735–746.

[101] R. Falgout, V. Henson, J. Jones, and U. Yang, *Boomer AMG: A parallel implementation of algebraic multigrid*, Tech. Report UCRL-MI-133583, Lawrence Livermore National Laboratory, 1999.

[102] C. Feichtinger, *Design and Performance Evaluation of a Software Framework for Multi-Physics Simulations on Heterogeneous Supercomputers*, Verlag Dr. Hut, München, 2012.

[103] C. Feichtinger, J. Götz, S. Donath, K. Iglberger, and U. Rüde, *Concepts of waLBerla Prototype 0.1*, Tech. Report 07-10, Department of Computer Science 10 (System Simulation), Friedrich-Alexander-Universität Erlangen-Nürnberg, Germany, 2007.

[104] L. Garshol, *BNF and EBNF: What are they and how do they work*, acedida pela última vez em **16** (2003).

[105] D. Ghosh, *DSLs in action*, Manning Publications Co., 2010.

[106] G. Gilboa, N. Sochen, and Y. Zeevi, *Image enhancement and denoising by complex diffusion processes*, IEEE Transactions on Pattern Analysis and Machine Intelligence **26** (2004), no. 8, 1020–1036.

[107] B. Gmeiner, *Design and Analysis of Hierarchical Hybrid Multigrid Methods for Peta-Scale Systems and Beyond*, Verlag Dr. Hut, München, 2013.

[108] B. Gmeiner and U. Rüde, *Peta-scale hierarchical hybrid multigrid using hybrid parallelization*, Large-Scale Scientific Computing, Lecture Notes in Computer Science, Springer Verlag, 2013, accepted.

[109] J. Götz, *Massively parallel direct numerical simulation of particulate flows*, Verlag Dr. Hut, München, 2012.

[110] J. Götz, S. Donath, C. Feichtinger, K. Iglberger, and U. Rüde, *Concepts of waLBerla Prototype 0.0*, Tech. Report 07–4, Department of Computer Science 10 (System Simulation), Friedrich-Alexander-Universität Erlangen-Nürnberg, Germany, 2007.

[111] J. Götz, K. Iglberger, C. Feichtinger, S. Donath, and U. Rüde, *Coupling Multibody Dynamics and Computational Fluid Dynamics on 8192 Processor Cores*, Parallel Computing **36** (2010), no. 2-3, 142–151.

[112] J. Götz, K. Iglberger, M. Stürmer, and U. Rüde, *Direct Numerical Simulation of Particulate Flows on 294912 Processor Cores*, 2010 ACM/IEEE International Conference for High Performance Computing, Networking, Storage and Analysis, IEEE, 2010, pp. 1–11.

[113] D. Haspel, *Simulation of Clotting Processes using Non-Newtonian Blood Models and the Lattice Boltzmann Method*, Master's thesis, Department of Computer Science 10 (System Simulation), Friedrich-Alexander-Universität Erlangen-Nürnberg, Germany, 2009.

[114] M. Heisig, *Efficient generation of Mehrstellenverfahren for elliptic PDEs*, Bachelor thesis, Department of Computer Science 10 (System Simulation), Friedrich-Alexander-Universität Erlangen-Nürnberg, Germany, 2013.

[115] H. Hellmann, *A new approximation method in the problem of many electrons*, The Journal of Chemical Physics **3** (1935), no. 1, 61–61.

[116] P. Hohenberg and W. Kohn, *Inhomogeneous electron gas*, Physical review **136** (1964), B864–B871.

[117] O. Honigman and Y. Zeevi, *Enhancement of Textured Images Using Complex Diffusion Incorporating Schroedinger's Potential*, 2006 IEEE International Conference on Acoustics, Speech and Signal Processing, 2006. ICASSP 2006 Proceedings, vol. 2, 2006.

[118] K. Iglberger, *Software design of a massively parallel rigid body framework*, Verlag Dr. Hut, München, 2010.

[119] K. Iglberger, J. Götz, S. Donath, C. Feichtinger, and U. Rüde, *Large Scale Simulations of Realistic Flow Problems*, inSiDE **7** (2009), no. 2, 40–45.

[120] K. Iglberger and U. Rüde, *The pe Rigid Multi-Body Physics Engine*, Tech. Report 09-9, Department of Computer Science 10 (System Simulation), Friedrich-Alexander-Universität Erlangen-Nürnberg, Germany, 2009.

[121] K. Iglberger and U. Rüde, *Massively parallel granular flow simulations with non-spherical particles*, Computer Science Research and Development **25** (2010), no. 1-2, 105–113.

[122] W. Kohn and L. Sham, *Self-consistent equations including exchange and correlation effects*, Physical Review **140** (1965), A1133–A1138.

[123] S. Kronawitter and Ch. Lengauer, *Optimization of two jacobi smoother kernels by domain-specific program transformation*, 9th International Conference on High-Performance and Embedded Architectures and Compilers (HiPEAC) (2013), accepted.

[124] O. McBryan, P. Frederickson, J. Lindenand, A. Schüller, K. Solchenbach, K. Stüben, C. Thole, and U. Trottenberg, *Multigrid methods on parallel computers – a survey of recent developments*, IMPACT of Computing in Science and Engineering **3** (1991), no. 1, 1–75.

[125] B. Nestler, H. Garcke, and B. Stinner, *Multicomponent alloy solidification: Phase-field modeling and simulations*, Physical Review E **71** (2005), no. 4, 041609.

[126] P. Neumann, *Numerical Simulation of Nanoparticles in Brownian Motion using the Lattice Boltzmann Method*, Master's thesis, Department of Computer Science 10 (System Simulation), Friedrich-Alexander-Universität Erlangen-Nürnberg, Germany, 2008.

[127] M. Odersky, L. Spoon, and B. Venners, *Programming in scala: a comprehensive step-by-step guide*, Artima Inc, 2008.

[128] J. Perdew and A. Zunger, *Self-interaction correction to density-functional approximations for many-electron systems*, Physical Review B **23** (1981), 5048–5079.

[129] K. Pickl, J. Götz, K. Iglberger, J. Pande, K. Mecke, A.-S. Smith, and U. Rüde, *All good things come in threes – Three beads learn to swim with lattice Boltzmann and a rigid body solver*, Journal of Computational Science **3** (2012), no. 5, 374–387.

[130] U. Rüde and Ch. Zenger, *A Workbench for Multigrid Methods*, Tech. Report TUM-I8607, Institut für Informatik der Technischen Universität München, 1986.

[131] J. Ryckaert, G. Ciccotti, and H. Berendsen, *Numerical integration of the cartesian equations of motion of a system with constraints: molecular dynamics of n-alkanes*, Journal of Computational Physics **23** (1977), no. 3, 327–341.

[132] E. Schrödinger, *An undulatory theory of the mechanics of atoms and molecules*, Physical Review **28** (1926), no. 6, 1049.

[133] N. Siegmund, S. Kolesnikov, Ch. Kästner, S. Apel, D. Batory, M. Rosenmüller, and G. Saake, *Predicting performance via automated feature-interaction detection*, Proceedings of the 2012 International Conference on Software Engineering, IEEE Press, 2012, pp. 167–177.

[134] K. Solchenbach, C. Thole, and U. Trottenberg, *Parallel multigrid methods: implementation on suprenum-like architectures and applications*, Supercomputing, Springer, 1988, pp. 28–42.

[135] T. Takaki, T. Shimokawabe, M. Ohno, A. Yamanaka, and T. Aoki, *Unexpected selection of growing dendrites by very-large-scale phase-field simulation*, Journal of Crystal Growth **382** (2013), 21–25.

[136] U. Trottenberg, C. Oosterlee, and A. Schüller, *Multigrid*, Academic Press, San Diego, CA, USA, 2001.

[137] A. Vondrous, M. Selzer, J. Hötzer, and B. Nestler, *Parallel computing for phase-field models*, International Journal of High Performance Computing Applications (2013).

[138] R. Wienands and W. Joppich, *Practical Fourier analysis for multigrid methods*, Numerical Insights, vol. 5, Chapmann and Hall/CRC Press, Boca Raton, Florida, USA, 2005.

[139] R. Wienands and I. Yavneh, *Collocation coarse approximation in multigrid*, SIAM Journal on Scientific Computing **31** (2009), no. 5, 3643–3660.

[140] K. Yelick, L. Semenzato, G. Pike, C. Miyamoto, B. Liblit, A. Krishnamurthy, P. Hilfinger, S. Graham, D. Gay, and Ph. Colella, *Titanium: A high-performance java dialect*, Concurrency Practice and Experience **10** (1998), no. 11-13, 825–836.

Part III.

Publikationen