Veranstaltung zum Tagungsmotto: Advanced Computing

H.-J. Bungartz und M. Mehl (Organisatoren)

{bungartz,mehl}@in.tum.de

Abstract

Where computers are still computing: At the Leibniz Supercomputing Centre in Garching, experts from science and industry demonstrate the manageability of current and future computing systems by means of Computer Science. Take part in presentations about topics such as HPC architectures, scientific computing, or visualisation, and directly experience what simulation nowadays is capable to achieve. Jump into the world of Advanced Computing and 'number crunchers' in tutorials and presentations. Gain new insights into one of the leading and most modern computing centres in guided tours - including supercomputing.

1 HPC System Architectures
C. Trinitis, Technische Universität München

2 Scalable Tools for HPC Systems and Grids
D. Kranzlmüller, Johannes Kepler Universität Linz

3 New Algorithmic Approaches for High Performance Computing
M. Bader, Technische Universität München

4 Parallele Visualisierungmethoden für große Datensätze
D. Weiskopf, Universität Stuttgart

Es werden Ziele und Anforderungen an Verfahren zur wissenschaftlichen Visualisierung großer Datenmengen diskutiert. Insbesondere wird die Frage der effizienten Darstellung, die für eine interaktive Visualisierung ausschlaggebend ist, beleuchtet. Hierfür werden generische Parallelisierungsstrategien für hybride Hardwarearchitekturen, wie beispielsweise Cluster-Computer mit mehreren GPU-Knoten, skizziert und exemplarisch für *sort-first*

und *sort-last* Ansätze zur direkten Volumenvisualisierung betrachtet. Zudem werden weitere Herausforderungen bei der Visualisierung großer Datensätze angerissen.

4.1 Einführung

Dieser Artikel beschäftigt sich mit Anforderungen und Zielen der wissenschaftlichen Visualisierung (engl. *scientific visualization*) sowie deren Umsetzung. Grundsätzlich zielt die wissenschaftliche Visualisierung auf Daten aus technischen, naturwissenschaftlichen und medizinischen Anwendungen [HJ05]. Ein wichtiges Problem ist die Visualisierung großer Datenmengen, welche eine besondere Herausforderung für die Visualisierungstechniken darstellen. Zentrale Punkte hierbei sind die Effizienz – eine Voraussetzung für eine interaktive Visualisierung – sowie zugehörige Parallelisierungsstrategien: Es werden sowohl klassische Parallelisierungsansätze mittels Bereichszerlegungen im Bild- oder Objektraum als auch aus dem Graphik- und Visualisierungsbereich stammende Ansätze der Parallelisierung auf Graphikprozessoren (GPUs, engl. *graphics processing units*) angesprochen.

4.2 Ziele und Anforderungen

Das übergeordnete Ziel der wissenschaftlichen Visualisierung ist es, die Benutzer von Visualisierungswerkzeugen beim Analysieren und Verstehen der Daten zu unterstützen. Da sowohl die vorliegenden Daten als auch der Visualisierungsprozess selbst von vielen Parametern abhängen, ist die Erzeugung einer fest vorgegebenen Visualisierung (z.B. eines einzelnen Bildes) meist nicht ausreichend. Stattdessen wird das Modell der interaktiven Exploration bevorzugt: In einem interaktiven Prozess können unterschiedliche Parameter untersucht und Visualisierungen aus unterschiedlichen Perspektiven und unter diversen Gesichtspunkten betrachtet werden, um dadurch einen vollständigen Eindruck von den Daten zu erhalten.

Eine technische Voraussetzung für Interaktion ist die ausreichend schnelle Erzeugung von Visualisierungsbildern. Je nach Anwendung werden Visualisierungszeiten von maximal 50 ms bis 200 ms als akzeptabel angesehen, d.h. Bildwiederholraten von 5 Hz bis 20 Hz. Ein Problem ist, dass die zu visualisierenden Datenmengen durch den wachsenden Umfang von Simulationsrechnungen und die immer bessere Auflösung von Scannern permanent größer und komplexer werden.

Die Herausforderungen sind folgende:

- Skalierbarkeit der Visualisierungsgeschwindigkeit für einen wachsenden Datenumfang. Insbesondere zeitabhängige Daten können im Bereich von mehreren Gigabyte bis zu Terabyte liegen. Die Herausforderung ist die Entwicklung von Verfahren für datenintensive Visualisierungsalgorithmen. Ein direkt verwandtes Problem ist die Skalierbarkeit des verfügbaren Speichers, damit diese Datenmengen verarbeitet werden können.

- Skalierbarkeit der Visualisierungsgeschwindigkeit für Displays mit hoher Auflösung. Beispielsweise verlangen Rückprojektionsleinwände (Powerwalls) Bilder mit 10 Megapixel oder mehr, was zu berechnungsintensiven Verfahren führt.
- Die Entwicklung von Visualisierungssoftware ist ein zunehmend komplexes Problem der Softwaretechnik. Die hybride Parallelisierung auf verschiedenen Hardware-Architektur-Ebenen (siehe Abschnitt 4.3) sowie der steigende Komplexitätsgrad von Visualisierungsverfahren führen zu Schwierigkeiten bei Entwicklung, Tests und Pflege großer Softwaresysteme.

Die in diesem Artikel diskutierte Strategie greift auf Parallelisierung zurück, um das Effizienzproblem der berechnungs- und datenintensiven Visualisierung zu lösen und um eine Skalierung der Rechengeschwindigkeit und des verfügbaren Speichers (bei Systemen mit verteiltem Speicher) zu erreichen. Im folgenden Abschnitt wird die Software-Entwicklung im Zusammenhang mit Programmiermodellen angerissen.

4.3 Hardware-Architekturen und Programmiermodelle

Bei Hardware-Architekturen geht der aktuelle Trend in Richtung steigender interner Parallelität. Ein typisches Beispiel ist die Entwicklung bei GPUs, die heute bis zu 128 parallele Verarbeitungseinheiten haben (Beispiel: NVIDIA GeForce 8800 Ultra). Ähnliches gilt für den Cell-Prozessor (Cell Broadband Engine) und für Mehrfachkern-CPUs. Für interne Parallelität spricht, dass die klassische Strategie der Steigerung der Taktraten an physikalisch-technische Grenzen stößt und Parallelisierung die einzig sinnvolle Alternative für Geschwindigkeitssteigerungen bietet. Ein verwandtes Problem ist die relativ geringe Bandbreite zum Hauptspeicher sowie über Netzwerkverbindungen, verglichen mit der Verarbeitungsgeschwindigkeit von Prozessoren. Insbesondere speicherintensive Verfahren sind von dieser Speicher-Wand (engl. *memory wall*) [WM95] betroffen.

Deshalb zielen effiziente Algorithmen sowohl auf Parallelisierung als auch auf Datenlokalität, um Speicherzugriffskosten zu verringern oder zu verstecken. Hierbei kann Parallelität auf verschiedenen Ebenen ausgenutzt werden: (1) Datenparallelität durch SIMD (engl. *single instruction, multiple data*), (2) Instruktionsparallelität durch gleichzeitiges Ausführen mehrerer skalarer Operationen, (3) Thread-Parallelität durch Multithreading auf eng gekoppelten Multikernrecheneinheiten, (4) Knoten-Parallelität durch Verteilung auf durch ein Netzwerk oder einen Bus verbundene Rechenknoten (z.B. in einem Rechen-Cluster oder auf mehreren GPUs innerhalb eines Rechners). Diese vier Ebenen führen zu einer hybriden Parallelisierung, die besondere Anforderungen an den Algorithmenentwurf und die -implementierung stellt.

Bedingt durch den Erfolg des GPU-Programmiermodells wird die Stromverarbeitung (engl. *stream processing*) immer beliebter [KRD+03]. Die Stromverarbeitung realisiert ein eingeschränktes, aber nützliches Modell der parallelen Verarbeitung. Hierbei sind Daten in Strömen (engl. *streams*) organisiert, und auf die Datenelemente werden Kernfunktionen (engl. *kernel functions*) angewendet. Die Kernfunktionen nutzen die Datenlokalität, indem

die meisten Zugriffe nur auf einen schnellen Speicher (z.B. lokale Register oder Stromregister in Form von Texturen) durchgeführt werden. Zudem kann innerhalb der Kernfunktionen Pipelining und Instruktionsparallelität ausgenutzt werden. Datenparallelität wird direkt durch die SIMD-Ausführung der Stromverarbeitung verwendet.

Für die Visualisierung spielt die Stromverarbeitungsarchitektur von GPUs momentan eine entscheidende Rolle. Eine entsprechende Programmierschnittstelle existiert für NVIDIA GPUs in Form von *CUDA* ("Compute Unified Device Architecture")[1]. Weitere Programmierumgebungen sind *Brook* bzw. *Brook+*, das beispielsweise für ATI-AMD GPUs eingesetzt wird[2]. Für eine weitergehende Hybridparallelisierung, die auch Knoten-Parallelität ausnutzt und Netzwerkkommunikation unterstützt, existiert die Erweiterung von *CUDA* auf *CUDASA* ("Compute Unified Device and Systems Architecture") [SMDE08]. Darüber hinaus können GPUs durch Graphikprogrammierschnittstellen wie *OpenGL* und *DirectX* angesprochen werden, die mittels computergraphischer Operationen eine Stromverarbeitung erlauben.

4.4 Allgemeine Parallelisierungsstrategien in der Visualisierung

Visualisierungstechniken haben schon frühzeitig das Stromverarbeitungsmodell von GPUs genutzt, um Berechnungen durch Parallelisierung zu beschleunigen. Die Hauptschwierigkeit besteht darin, Algorithmen für das eingeschränkte Programmiermodell der Stromverarbeitung zu entwickeln. Einen Überblick über entsprechende Verfahren für die Vektorfeld- und Volumenvisualisierung ist in [Wei06] zu finden.

Bei der Parallelisierung auf getrennten Knoten (z.B. in einem GPU-Cluster-Computer) führt das grundlegende Problem der Bereichszerlegung dazu, dass verteilt berechnete Bildanteile anschließend zu einem Gesamtbild zusammengefügt werden müssen. Entsprechende Ansätze lassen sich in drei Kategorien klassifizieren [MCEF94]: *sort-first*, *sort-middle* und *sort-last*. Im ersten Fall werden die graphischen Primitive (beispielsweise Volumenelemente oder Dreiecke) räumlich sortiert auf die Prozessoren verteilt; im zweiten Fall geschieht die Partitionierung nach der geometrischen Transformation und Beleuchtungsberechnung, aber noch vor der Rasterung der geometrischen Primitive auf dem Bildschirm; bei *sort-last* erfolgt die Sortierung erst nach der Rasterung. Da der Prozess aus geometrischer Transformation, Beleuchtungsberechnung und Rasterung meist gekapselt ist (beispielweise innerhalb des GPU-Stromprozessors), ist die *sort-middle* Strategie normalerweise nicht anwendbar. Im folgenden Abschnitt wird das Problem der Bereichszerlegung exemplarisch für die direkte Volumenvisualisierung diskutiert.

[1] `http://www.nvidia.com/object/cuda_home.html`
[2] `http://ati.amd.com/technology/streamcomputing`

4.5 Parallele Volumenvisualisierung

Die direkte Volumenvisualisierung zielt auf die Darstellung von 3D-Skalarfeldern und ist eine grundlegende Technik der wissenschaftlichen Visualisierung. Aufgrund der darzustellenden Datenmengen (meist über 1 GB pro Zeitschritt) ist die direkte Volumenvisualisierung ein typisches Beispiel für eine daten- und berechnungsintensive Anwendung, die traditionell durch GPUs beschleunigt wird [EHK+06]. Im Folgenden wird angenommen, dass die intrinsische GPU-Parallelität durch Darstellungsverfahren wie GPU-Raycasting (d.h. Verfolgung von Sehstrahlen) oder die texturbasierte Schichtdarstellung [EHK+06] für die Visualisierung eines Teilblocks des gesamten Datensatzes ausgenutzt wird.

Damit stellt sich die Frage, wie die Knoten-Parallelität in einem Cluster mit mehreren GPU-Knoten genutzt werden kann. Ein beliebter Ansatz beruht auf *sort-last*, bei dem die Volumendaten im 3D-Objektraum in Unterblöcke partitioniert werden [SMW+05]. Die Unterblöcke werden getrennt auf den einzelnen GPU-Knoten visualisiert und die so erzeugten Zwischenbilder anschließend tiefensortiert und durch Überlagerung (engl. *blending*) kombiniert (dieser Teil entspricht dem "Sortieren" in *sort-last*). Die Objektraumpartitionierung erlaubt bei statischen Volumendaten eine statische Verteilung dieser Daten, was eine ideale Skalierung des verfügbaren Speichers mit der Anzahl der GPU-Knoten ermöglicht. Andererseits bietet die feste Objektraumpartitionierung keine gute Lastverteilung, falls durch entsprechende Kameraparameter die Unterblöcke unterschiedliche Größen im Bildbereich aufweisen. Zudem sind der Transfer und die Überlagerung von Zwischenbildern stark von der Bildschirmauflösung abhängig. Alternativ liefert der *sort-first* Ansatz mit einer Partitionierung im Bildraum eine gute Skalierbarkeit bezüglich der Bildschirmauflösung. Normalerweise werden hierbei jedoch vollständige Kopien der Volumendaten auf allen GPU-Knoten gehalten, und eine Skalierbarkeit des Speichers ist nicht gegeben.

Durch eine adaptive Verteilung der Volumenunterblöcke auf die GPU-Knoten lässt sich der Speicher auch beim *sort-first* Ansatz skalieren [MWMS07]. Die Lastverteilung wird optimiert, indem für jedes Bild die blickpunktabhängige Last durch einen vereinfachten Darstellungsprozess abgeschätzt wird. Entsprechend der abgeschätzten Rechenzeiten wird der Bildbereich gemäß einer kd-Baum-Struktur aufgeteilt. Abbildung 1 zeigt die Bildraumpartitionierung und Zuordnung zu acht GPU-Knoten durch Farbkodierung; die Volumenunterblöcke sind durch ein Drahtgitter visualisiert. Da die Verteilung blickpunktabhängig ist, steigt der Kommunikationsaufwand zwischen den GPU-Knoten. Jedoch kann durch Caching-Strategien und Ausnutzung der zeitlichen Kohärenz der Kamerabewegung die Auswirkung der zusätzlichen Netzwerkkommunikation reduziert werden. Insbesondere bei zeitabhängigen Daten, die ohnehin einen permanenten Datentransfer benötigen, kann man den zusätzlichen Kommunikationsaufwand gegenüber dem Objektraumverfahren vernachlässigen. Damit stellt dieses *sort-first* Verfahren einen guten Kompromiss zwischen der Skalierbarkeit des Speichers und der Visualisierungsgeschwindigkeit auch für große Displays dar.

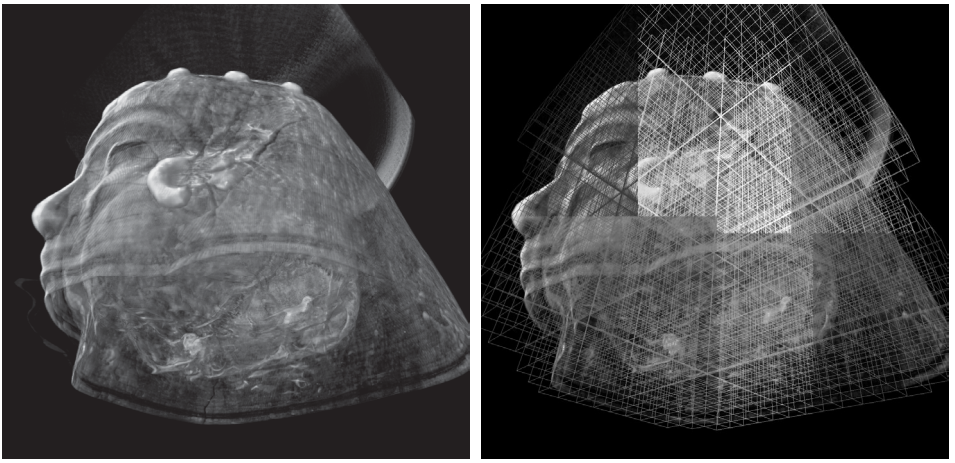

Abbildung 1: Beispiel für parallele Volumenvisualisierung. Links: Darstellung des CT-Scans einer Mumie. Rechts: Visualisierung der Bildraumpartitionierung (aus: Moloney et al. [MWMS07]; © Eurographics Association 2007, reproduced by kind permission of the Eurographics Association)

4.6 Ausblick

In diesem Artikel wurden Parallelisierungsstrategien für die Visualisierung großer Datensätze auf hybriden Systemen wie beispielsweise GPU-Clustern angerissen. Wie exemplarisch für die direkte Volumenvisualisierung gezeigt, stellen sich grundsätzlich die Fragen nach der angemessenen Partionierungsstrategie (z.B. Objekt- oder Bildraum) und der Parallelisierung auf hybriden Architekturen (z.B. innerhalb der GPU bzw. zwischen Cluster-Knoten). Die Volumenvisualisierung profitiert von der hohen intrinsischen Parallelität der Problemstellung (d.h. Unabhängigkeit der Sehstrahlen) und der relativ einfachen Partionierung der Daten in unabhängige Unterblöcke. Komplexere Verfahren der wissenschaftlichen Visualisierung wie beispielsweise Methoden für Vektorfelder führen zu einer weniger offensichtlichen Zerlegung (z.B. aufgrund der notwendigen Teilchenverfolgung entlang von Vektorfeldern) und stellen deshalb noch ungelöste Herausforderungen an die Parallelisierung.

Neben den in diesem Artikel angesprochenen Verteilungsansätzen spielen bei der Visualisierung großer Datensätze Aspekte wie ein effizienter Zugriff auf den Festplattenspeicher (für *out-of-core* Algorithmen), hierarchische und adaptive Darstellungsmethoden sowie Kompressionsverfahren eine wichtige Rolle. Bei Datenmengen im Bereich von Peta- und Exabytes, die in naher Zukunft Realität sein werden, wird die Datenhaltung und -verteilung eine immer größer werdende Rolle spielen; eine erfolgversprechende Lösungsstrategie ist die gemeinsame Simulation und Visualisierung auf demselben Rechner, um somit den Datentransfer zwischen Simulation und Visualisierung zu vermeiden. In diesem Zusammenhang wird das Problem der Entwicklung komplexer Simulations- und Visualisierungssoftware eine größer werdende Rolle spielen. Schließlich stellen sich grundsätzliche Fragen nach der halbautomatischen Analyse, um die darzustellenden Datenmengen für die Benutzer zu reduzieren, sowie der Berücksichtigung von Wahrnehmungsaspekten.

Danksagung

Der Mumien-Datensatz (Abbildung 1) wurde freundlicherweise von Brown & Herbranson Imaging (Stanford Radiology) und dem Rosicrucian Museum zur Verfügung gestellt. Der Autor dankt Magnus Strengert für inhaltliche Kommentare und Bettina A. Weiskopf für ihre Hilfe beim Korrekturlesen.

Literatur

[EHK+06] K. Engel, M. Hadwiger, J. M. Kniss, C. Rezk-Salama und D. Weiskopf. *Real-Time Volume Graphics*. A K Peters, 2006.

[HJ05] C. D. Hansen und C. R. Johnson, Hrsg. *The Visualization Handbook*. Elsevier, 2005.

[KRD+03] U. J. Kapasi, S. Rixner, W. J. Dally, B. Khailany, J. H. Ahn, P. Mattson und J. D. Owens. Programmable Stream Processors. *IEEE Computer*, 36(8):54–62, 2003.

[MCEF94] S. Molnar, M. Cox, David E. und H. Fuchs. A sorting classification of parallel rendering. *IEEE Computer Graph. Appl.*, 14(4):23–32, 1994.

[MWMS07] B. Moloney, D. Weiskopf, T. Möller und M. Strengert. Scalable sort-first parallel direct volume rendering with dynamic load balancing. In *EG Symp. Parallel Graph. Vis.*, S. 45–52, 2007.

[SMDE08] M. Strengert, C. Müller, C. Dachsbacher und T. Ertl. CUDASA: Compute unified device and systems architecture. In *EG Symp. Parallel Graph. Vis.*, S. 49–56, 2008.

[SMW+05] M. Strengert, M. Magallón, D. Weiskopf, S. Guthe und T. Ertl. Large volume visualization of compressed time-dependent datasets on GPU clusters. *Parallel Computing*, 31(2):205–219, 2005.

[Wei06] D. Weiskopf. *GPU-Based Interactive Visualization Techniques*. Springer, 2006.

[WM95] W. A. Wulf und S. A. McKee. Hitting the memory wall: implications of the obvious. *ACM SIGARCH Computer Architecture News*, 23(1):20–24, 1995.

5 Software Engineering for Computational Science and Engineering
R. Kendall, Carnegie Mellon University

6 Challenges in Computational Seismology using a HPC-Infrastructure
M. Käser, Ludwig-Maximilians-Universität München

We present current challenges encountered in computational seismology using modern HPC systems. The local time stepping algorithm of the solver SEISSOL leads to load balance and scaling problems due to the asynchronous evolution of the solution in elements with different time step lengths. New mesh partitioning strategies have to be developed as

an interdisciplinary effort to fully exploit the performance of powerful HPC facilities for the creation of highly valuable data sets from earthquake simulations.

6.1 Introduction

Computational Seismology has become an increasingly important discipline in geophysics in the last two decades. Especially, large-scale applications to solve seismic wave propagation problems including realistic material properties, complex geometries of the geological structures, and advanced approaches to handle dynamic rupture physics are aspired more than ever before. This is due to the requirement of highly accurate synthetic data sets of fully three-dimensional seismic wave fields and of the time series (seismograms) of ground motions typically observed at the Earth's surface. Such synthetic data sets are used in ground motion predictions for seismic hazard assessment or in inversion processes to find a geological subsurface model or the source parameters of an occurred earthquake. However, inversion techniques typically are based on an iterative process. In each iteration step, the simulated data are compared with real field observations, certain misfit criteria are evaluated, and the model parameters are modified accordingly. For each new model representation, a new numerical forward calculation has to be carried out, which usually is the most time consuming part of the iterative inversion process, until the misfits fall under an acceptable level.
Our current research considers only the forward calculation, i.e. the computationally most expensive part of the modeling, to produce the synthetic data sets. It is based on the numerical solution of the partial differential equations (PDE) describing the propagation of seismic waves through heterogeneous materials on different spatial scales. To this end, we intended to create a robust but also flexible numerical algorithm that can handle a large variety of seismological problems. First, the term *flexible* here means that we formulate the seismic wave equations mathematically as a linear hyperbolic system with different types of source terms. This formulation is derived from the theory of linear elasticity combining Hooke's law and Newton's equations of motion and leads to the so-called velocity-stress formulation of the seismic wave equations which allows for the incorporation of a variety of different material properties. Currently, we can handle acoustic fluids, elastic solids, viscoelastic solids to account for attenuation effects, anisotropic solids, and poroelastic material to include the effect of pore fluids on the propagation of seismic waves. Secondly, we understand the term *flexible* in the sense of geometrical complexity. The required discretization of the equations is based on the decomposition of the three-dimensional computational domain into tetrahedral or hexahedral elements. However, automatic mesh generation using cutting-edge third-party and usually commercial software shows that tetrahedral meshes are easier to obtain for very complex geometries. Furthermore, element deformations can typically be treated more easily for tetrahedral than for hexahedral elements. Nevertheless, hexahedral meshes often lead to a smaller number of elements for the same problem while retaining a similar accuracy. Therefore, the ultimate goal would be to combine both mesh types in order to use their particular advantages where possible. Finally, we use the term *flexible* with respect to our numerical scheme applied to solve

the PDE. We developed a Discontinuous Galerkin (DG) finite element method that uses a particular time integration technique based on the solution of Arbitrary accuracy DErivative Riemann problems (ADER). This way, we can formulate the numerical scheme to obtain solutions of arbitrarily high-order accuracy in space and time. Furthermore, the proposed ADER-DG scheme allows us to apply p-adaptation, where the degree of the approximation polynomial can be chosen locally for each element depending on the accuracy requirements on the numerical solution. Additionally, local time stepping (LTS) can be used, such that the explicit numerical scheme can update each element with its own optimal time step respecting a local Courant-Friedrich-Levy (CFL) stability criterion.
With respect to the efficient usage of High-Performance-Computing (HPC) systems, our workflow from model building over mesh generation to the final solution of the PDE poses several challenges. In the following, we will mainly address problems related to the formulation of the flexible ADER-DG algorithm with p-adaptation and LTS.

6.2 The ADER-DG Method in SEISSOL

The variety of seismological problems and their corresponding models with highly heterogeneous material properties, rheological peculiarities, and geometrical complexity strongly influences the structure and development of the implementation of the ADER-DG method to simulate seismic wave propagation. The code SEISSOL is implemented in FORTRAN 90 and uses MPI for communication between processors. The algorithm solves the hyperbolic PDE of the form

$$\frac{\partial Q_p}{\partial t} + A_{pq}\frac{\partial Q_q}{\partial x} + B_{pq}\frac{\partial Q_q}{\partial y} + C_{pq}\frac{\partial Q_q}{\partial z} = S_p, \tag{1}$$

where $Q_p = (\sigma_{xx}, \sigma_{yy}, \sigma_{zz}, \sigma_{xy}, \sigma_{yz}, \sigma_{xz}, u, v, w)^T$ is the vector of the unknown stresses and velocities, A_{pq}, B_{pq}, and C_{pq} the Jacobian matrices defined by the material properties, and S_p is the source term.

After subdividing the computational domain into elements indicated by $\mathcal{T}^{(i)}$, the degree N of the approximation polynomial inside each element can be chosen individually. Then a local time step length $\Delta t^{(i)}$ is determined by the stability criterion

$$\Delta t^{(i)} < \frac{1}{2N+1} \cdot \frac{l_{min}^{(i)}}{s_{max}^{(i)}}, \tag{2}$$

where $l_{min}^{(i)}$ is the diameter of the insphere of element $\mathcal{T}^{(i)}$ and $s_{max}^{(i)}$ is the maximum wave speed arising in the element.
Omitting the details of the full numerical scheme [DKT07], we explain the LTS algorithm by visualizing its fundamental concept by a one-dimensional example in Fig. 2. Small elements, e.g. $\mathcal{T}^{(5)}$, have a small time step and therefore have to updated more frequently than a large element, e.g. $\mathcal{T}^{(1)}$. In fact, each element can only be updated if it fulfills the

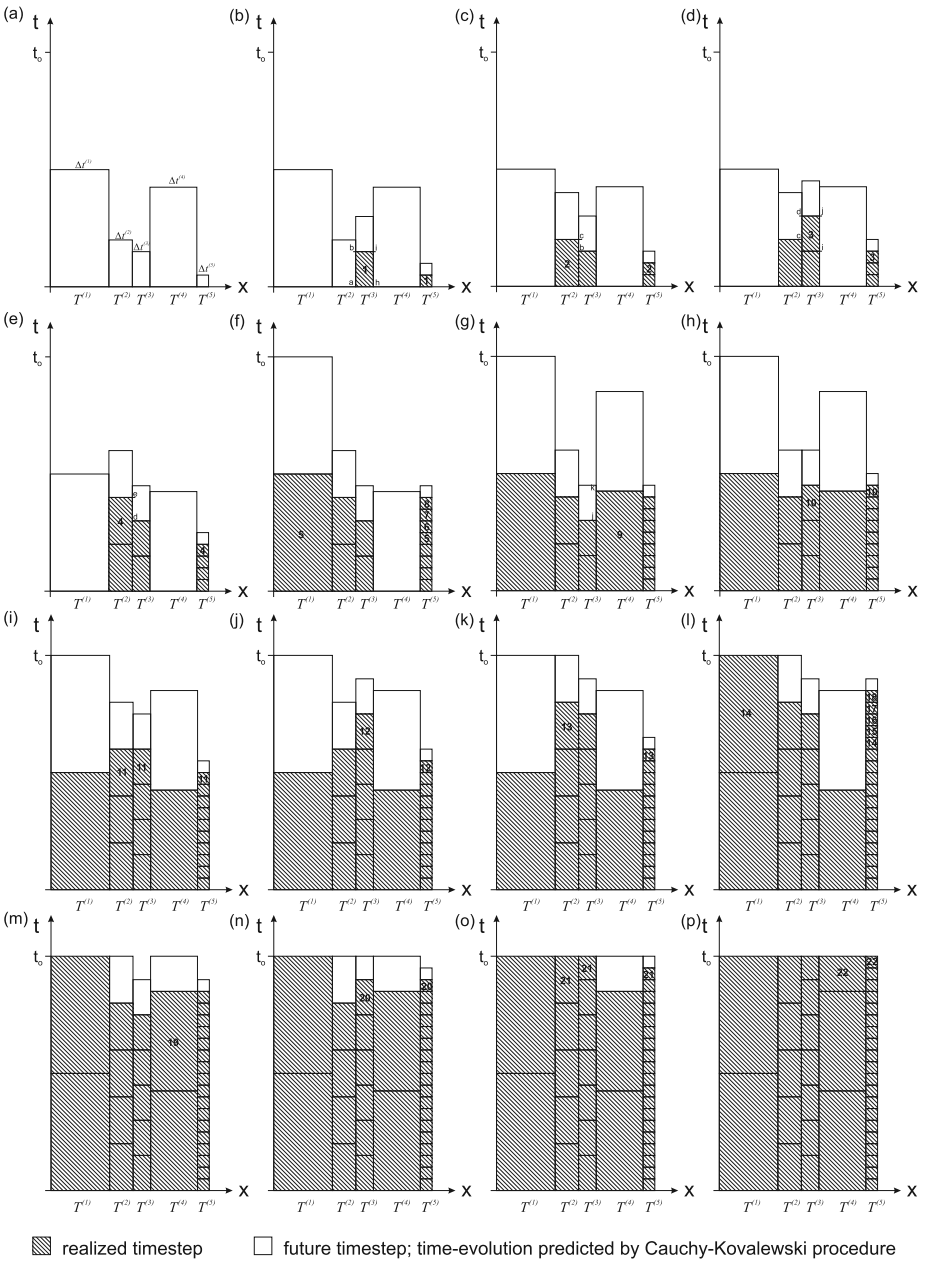

Abbildung 2: The local time stepping concept in one dimension. Space-time domains are shown for a number of consecutive time cycles to illustrate the asynchronous updates of five elements of different sizes. In each cycle elements are locally advanced to their next time level.

update criterion

$$t^{(i)} + \Delta t^{(i)} \leq \min\left(t^{(k_j)} + \Delta t^{(k_j)}\right) \qquad \forall\, k_j \tag{3}$$

with respect to all the direct neighboring element $T^{(k_j)}$. If an element can be updated, the numerical fluxes between two elements $T^{(i)}$ and $T^{(k_j)}$ have to be computed in the time interval

$$[t_1; t_2] = \left[\max\left(t^{(i)}, t^{(k_j)}\right); \min\left(t^{(i)} + \Delta t^{(i)}, t^{(k_j)} + \Delta t^{(k_j)}\right)\right]. \tag{4}$$

This is shown e.g. for the first update in Fig. 2(b) for element $\mathcal{T}^{(3)}$ for the time intervals $[a, b]$ and $[h, i]$. . Therefore, the total number of updates using local time stepping is reduced significantly compared to global time stepping. However, due to the asynchronous computations in time the parallel code can produce problems in scalability as further discussed in Section 6.4. The varying cost per element due to p-adaptation additionally complicates the situation.

6.3 Cache Memory Usage

A key issue to exploit the full performance of current HPC hardware is the optimal usage of cache memory. The core of the SEISSOL code consists of millions of small ($\sim 50 \times 50$) and mainly sparse matrix-matrix multiplications requiring only very local data from each element and its direct neighbors. As jumps in address space on modern computer architectures might cause cache misses, the access time can increase significantly. Therefore, Bader & Zenger [BZ06] designed algorithms that inherently benefit from underlying cache hierarchies to efficiently perform matrix multiplications without the need of address arithmetic. Their approach is based on the use of space filling curves (Peano curves) to order the matrix elements with respect to optimal data locality. An open question is, if their approach can be used with respect to unstructured tetrahedral meshes and non-square sparse matrices. The comparison with highly optimized routines, e.g. from the Intel Math Kernel Library, will give important information about the code's performance in terms of flops.

6.4 Load Balance and Scaling

The parallelization of SEISSOL is based on the partition of the computational mesh and poses new challenges, as the LTS algorithm introduces the issue of synchronization.

Mesh Partitioning
We use the software package METIS [KK98] to partition our finite element meshes. Applying a particular weighting strategy, we can produce mesh partitions of equal computational cost considering the individual cost of each element, e.g. caused by different rheologies or approximation orders (p-adaptation). However, using the LTS algorithm, different

elements update asynchronously at different times. Large elements typically update less frequently than smaller ones (see Fig 2). As a consequence, they have to wait until their

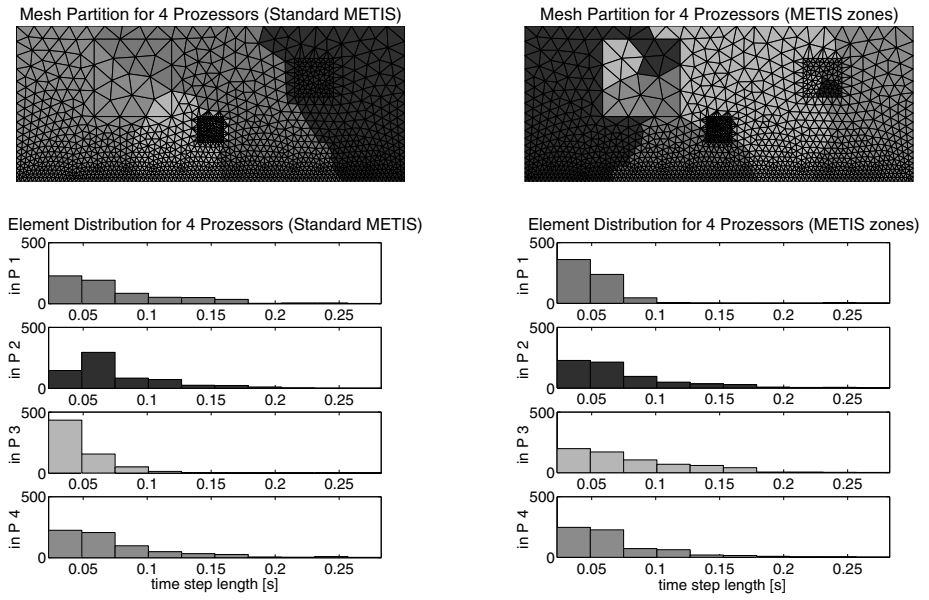

Abbildung 3: Mesh partitions for four processors (indicated by different colors) created by METIS with an equal number of elements. (a) No balanced distribution of elements (i.e. time step) sizes leading to asynchronous computations. (b) Element distribution using a zonal partitioning strategy leading to improved synchronization and load balancing.

next time step fulfills the update criterion (3). In the case that one partition contains more small elements than some other it also involves more element updates. Therefore, not only the computational cost and total number of updates inside a partition have to be considered but also an equal distribution of time step lengths, i.e. the number of elements with different time steps should be the same on each processor.

For the LTS algorithm of SEISSOL, a good scalability can be achieved only if: (i) good load balance is assured, (ii) communication between processors is kept minimal, and (iii) synchronization is ensured. Partitions done by METIS satisfy only constraints (i) and (ii), but not (iii). Thus, additional constraints are necessary for an improved parallel LTS performance. The example in Figure 6.4 shows the difference of partitioning strategies for the same problem with heterogeneous element sizes. We measure the element distribution by assigning them according to their time step lengths to ten different bins. Typically, a better load balance due to an improved element distribution can be achieved at the cost of increased communication. The variation in CPU-time requirements of each partition during different time cycles additionally complicates the problem. Therefore, the optimization of the load balance, synchronization, and scaling for thousands of processors remains a challenging task.

Space Filling Curves
A new approach to improve the load balance and scaling issue might again be based on space filling curves (SFC). A mesh can be re-ordered using the Hilbert SFC, whose shape maintains communication at a minimum [RK08]. However, for locally very fine meshes, the generation of the Hilbert SFC is a demanding task. The mesh partitioning then begins at one end of the SFC collecting elements along the curve until their time step lengths sum up to a number that corresponds to one partition. This process is repeated until the whole mesh is subdivided into the number of desired partitions. Obviously, this way the partitions will no longer have the same number of elements but they will be compact and have a comparable load. In general, an average improvement of 15-20% was observed. However, the time evolution of the load balancing still poses problems. In fact, tracing the time for each time iteration (cycle) of all processors during a simulation generates a spectrum-like graph [RK08]. The narrower this band, the better the load balance is. However, the more processors are used, the more crucial such load balancing issues become.

6.5 Conclusion

We raised the issue of using modern HPC systems efficiently and pointed out some of the major challenges concerning the load balance and scaling properties of the local time stepping approach of SEISSOL, a flexible solver for wave propagation problems in computational seismology. We want to emphasize that, in our opinion, only an interdisciplinary collaboration of geophysicists, applied numerical mathematicians, and computer scientists will help to overcome these current problems encountered when trying to exploit the full performance of powerful HPC systems.

Acknowledgments: The authors thank the DFG supporting our work through the Emmy Noether-Program (KA 2281/2-1) and the LRZ for the help within the DEISA Extreme Computing Initiative of the DEISA Consortium (www.deisa.eu), co-funded through EU FP6 projects RI-508830 and RI-031513.

Literatur

[BZ06] M. Bader and C. Zenger. Cache oblivious matrix multiplication using an element ordering based on a Peano curve. *Linear Algebra and Its Applications*, 417(2-3):301–313, 2006.

[DKT07] M. Dumbser, M. Käser, and E. Toro. An Arbitrary High Order Discontinuous Galerkin Method for Elastic Waves on Unstructured Meshes V: Local Time Stepping and p-Adaptivity. *Geophys. J. Int.*, 171(2):695–717, 2007.

[KK98] G. Karypis and V. Kumar. Multilevel k-way partitioning scheme for irregular graphs. *J. Parallel Distrib. Comput.*, 48:96–129, 1998.

[RK08] O. Rivera and M. Käser. Toward Optimal Load Balance in Parallel Programs for Geophysical Simulations. *inSiDE*, 6(1):46–49, 2008.

7 Demonstration of HPC Grid Applications at LRZ
H.-M. Adorf, A.C. Frank, I. Saverchenko, I.L. Muntean

7.1 Introduction

The term ”Grid Computingöriginated in the early 1990s to define a paradigm for making computer power as easily accessible as the electric power grid. The Grid is a distributed system that enables the sharing, selection, and aggregation of geographically distributed resources dynamically at runtime depending on their availability, capability, performance, cost, and quality-of-service requirements.

What distinguishes the Grid from typical cluster systems is that grids consist of loosely coupled, heterogeneous, and geographically dispersed computer resources. While a computing grid may be dedicated to a specialized science field, it is often constructed with the aid of software libraries and middleware serving a more general purpose. These components help to hide complexity that is often faced by the users of high performance computing systems. Grid middleware provides, for instance, a single point of contact and a common interface, both command line and graphical, to a variety of resources. Another important feature of the Grid is the single sign-on mechanism that enables a user to authenticate once and gain access to resources of multiple software systems.

Today references to several hundreds of Grid projects can be found in the Internet. A part of them are research projects that are carried out by single universities or research institutions. Although there also many national and international Grids some of them spanning across different continents. Grids come in different flavors, concentrating on either compute power, data federation, etc. The Leibniz Supercomputing Centre is mostly focused on Grid in HPC and is currently involved in several national and international grid projects including D-Grid and DEISA.

In the following, three demo applications are described which illustrate various aspects of the Grid. All of them are focused on executing numerical simulations, yet each demo uses a unique set of tools and technologies and presents a specific Grid usage scenario.

7.2 Management of CFD Simulations on the Grid

A drift ratchet, also known as Brownian motor, is used in particle separation devices at the microscale. Experiments show that under certain conditions devices based on drift ratchets lead to the separation of macroparticles by size. Nevertheless, the physics behind this separation process is not yet completely understood. The aim of our simulations is to identify and quantify different sources that lead to the drift ratchet effect inside one single pore of a separation device.

The computational challenges bundled with the fluid-particle interaction scenario are: 3D Navier-Stokes solver for the flow motion, multi-scale and multi-physics simulation, long simulation time, various pore chamber geometries and particle dimensions, etc. A para-

meter study has to be carried out, where every single simulation takes several weeks on a modern supercomputer. To allow efficient control and management of the simulations the GridSFEA framework is used.

GridSFEA is a lightweight framework focused on supporting various simulation scenarios executed on the Grid. It provides several features of interest to a broad group of end-users including management of simulation results, checkpoint-based migration of simulations, preview and transparent access to the results. The framework is based on the Globus Toolkit 4 middleware and employs such technologies as GridFTP, RFT, WS GRAM, etc.

In the demonstration is shown how a Grid user can benefit from the GridSFEA framework while performing such tasks as job migration and management of simulation results. For running the simulations D-Grid and DEISA resources are used.

7.3 Using the Grid Infrastructure for Galaxy Collision Simulations

Large galaxies like our own Milky Way often collide with neighboring galaxies. Most of the time a large galaxy hits a smaller ”dwarf”galaxy, resulting in the smaller galaxy being absorbed by the larger one. However, once in awhile, a large galaxy may encounter another large galaxy. Depending on the circumstances the result may be a merger of these galaxies into a single elliptical galaxy.

Galaxy collisions take much longer than other collisions we are used to. A merger of two large galaxies may take as long as 1 billion years to occur! Thus, we are unable to observe with a telescope how a collision happens. The best we can do is to take many observations, each of which represents a snapshot of a different collision-in-progress.

Computer simulations are a good way to see how galaxy collisions might evolve, and this is the topic of our demo. Two slowly rotating galaxies are sent on a collision course towards each other. A number of properties control the ultimate fate of the galaxies such as their relative mass, their relative speed, and whether they are on a head-on collision course. During a simulation, at regular intervals 3D snapshots of the evolving collision are saved and later assembled to a movie. Since a high-resolution simulation takes a long time, we demonstrate a real-time production of a low-resolution simulation using the Grid.

A given 3D snapshot, corresponding to a certain state of the evolving galaxy collision, may be viewed from different angles. We simulate a moving camera which travels along a given trajectory around the 3D model of the colliding galaxies. At regular stops 2D pictures are taken, and when these are assembled, a movie results which shows how the galaxy collision looks from different perspectives. The movie production is fast enough to be carried out on the Grid in real-time during the demo.

Various features of the AstroGrid-D project infrastructure and Globus Toolkit 4 middleware are presented during the demo. The execution of the jobs is being organized by the Planck Process Coordinator (ProC) scientific workflow engine.

7.4 Remote Interactive Visualization of Simulations on the Grid

Running an interactive numerical simulation is a challenge since resources a user normally has access to sometimes do not provide necessary computing power. Computing on the Grid that offers a greater variety of resources is therefore often easier and faster, however, the executing host is a priori unknown to the user. This introduces a problem since the user requires this information to be able to establish a connection between the execution and visualization hosts. As a part of this demo we present a viable solution to this problem.

Another challenge is the amount of data generated by a numerical simulation that often consists of many gigabytes. To visualize these data sets, lots of RAM, powerful graphics cards and expensive software licenses are needed. The solution is a dedicated visualization server that takes 3D graphics commands, renders 2D images and sends these back as jpeg. In this demo a freely available software program NAMD is used in order to run the simulation on the Grid. Data produced is visualized on a high end graphic workstation via VMD and the final images are displayed on a remote terminal using VirtualGL.

NAMD is a parallel molecular dynamics code designed for high-performance simulation of large biomolecular systems. The code shows strong scaling and can be run on thousands of processors. VMD is a molecular visualization program for displaying, animating, and analyzing large biomolecular systems using 3-D graphics and built-in scripting. VMD provides a powerful means of visualizing these simulations, and using the Interactive Molecular Dynamics (IMD) interface can even enable real time simulation steering. Any molecular dynamics simulation that runs in NAMD can be used for IMD.

The numerical simulation is run on the Grid and directs its output to a remote visualization station, which features a set of high performance graphics card for high speed rendering of 3D scenes. The created 2D images are then compressed and sent over the network to the user. Mouse events generated by the user are sent back to the execution host which reacts accordingly.

8 IBM Presentation
K. Gottschalk, IBM

9 Sun Microsystems – Peta-Scale I/O with Lustre
R. Rambau, Sun Microsystems

Suns Lustre file system first went into production in Spring 2003 on the Multiprogrammatic Capability Resource (MCR) cluster at Lawrence Livermore National Laboratory (LLNL). The MCR cluster was one of the largest clusters at that time with 1100 Linux compute nodes as Lustre clients. Since then, the Lustre file system has been deployed on even larger systems, notably the Sandia Red Storm deployment with approximately $25,000$ clients, and the Oak Ridge National Laboratory (ORNL) Jaguar system, which is of similar scale and runs Lustre technology both with client-node Linux (CNL) and

with Catamount, and then recently on the TACC Ranger cluster with its now 580 TF peak performance. A number of other Lustre system deployments feature many thousands of clients. The servers used with these configurations vary considerably with some clusters using fast heavyweight servers and others using a great many lightweight servers. The scale of these clusters poses serious challenges to any I/O system. This talk discusses discoveries related to cluster scaling made over the years. It describes implemented features, work currently in progress, and problems that remain unresolved. Topics covered include scalable I/O, locking policies and algorithms to cope with scale, implications for recovery, and other scalability issues.

10 Cray System Architectures for Petaflop Computing
W. Oed, Cray

Whilst the TOP500 list has little to do with actual application performance, it nonetheless reflects architectural and technological trends. The largest performance gains come from increased parallelism on practically all levels.

Within a processor, there are multiple functional units as well as multiple cores. In order to reach Petaflop scale, well over $100,000$ processor cores need to be employed in such a system. And, indeed, the average number of processors (or processor cores) employed in the ten most powerful systems of the TOP 500 already is close to $100,000$ processors.

This trend is expected to continue for the next couple of years. Thus, the biggest challenge is making this computational power available to a wider spectrum of 'real' applications such as fluid dynamics or complex weather models.

Contrary to the simple Linpack benchmark, for applications to scale to thousands of processors, aspects like communication, load imbalance, and I/O come into play. In part, this means tuning applications accordingly, but hardware and system software have to operate in a reliable and consistent way. Any kind of disturbances, commonly referred to as 'jitter' are inhibiting scalability.

This presentation will cover the design criteria of the current Cray XT5 scalable supercomputer and provide an outlook to future Cray MPP architectures. This includes processor performance, interconnection network, I/O-subsystem, power and cooling as well as system and application software.

11 UG – A Flexible Software Tool for PDEs
A. Naegel, G. Wittum, Universität Heidelberg

Numerical simulation provides important tools for the understanding of systems in science and engineering. It has therefore become one of the major topics in Computational Science. To promote modelling and simulation of complex problems new strategies are needed allowing for the solution of large, complex model systems. Crucial issues for such strategies

are reliability, efficiency, robustness, usability, and versatility.

After discussing the needs of large-scale simulation, we point out basic simulation strategies such as adaptivity, parallelism, and multigrid solvers. To allow adaptive, parallel computations, the load balancing problem for dynamically changing grids has to be solved efficiently by fast heuristics. These strategies are combined in the simulation system UG ('Unstructured Grids') being presented in the following.

In the second part of the presentation we show the performance and efficiency of this approach in various applications. In particular large scale parallel computations of density-driven groundwater flow as well as some non-standard problems from biotechnology and medicine are discussed in more detail. We present results for drug diffusion through human skin and signalling mechanisms in the neurosciences.

12 Exploiting Multicore Processors – a Solvable Problem?

J. Weidendorfer, Technische Universität München

13 Software-Optimisation

G. Wellein, Regionales Rechenzentrum Erlangen