This is a preprint of an article accepted for publication in the Int. J. Computational Science and Engineering

Co-Processor Acceleration of an Unmodified Parallel Solid Mechanics Code with FeastGPU

Dominik Göddeke*, Hilmar Wobker
Institute of Applied Mathematics, Dortmund University of Technology, Germany
E-mail: dominik.goeddeke@math.tu-dortmund.de
*Corresponding author

Robert Strzodka
Max Planck Center, Max Planck Institut Informatik, Germany

Jamaludin Mohd-Yusof, Patrick McCormick
Computer, Computational and Statistical Sciences Division
Los Alamos National Laboratory, USA

Stefan Turek
Institute of Applied Mathematics, Dortmund University of Technology, Germany

Abstract: FEAST is a hardware-oriented MPI based Finite Element solver toolkit. With the extension FEASTGPU the authors have previously demonstrated that significant speed-ups in the solution of the scalar Poisson problem can be achieved by the addition of GPUs as scientific co-processors to a commodity based cluster. In this paper we put the more general claim to the test: Applications based on FEAST, that ran only on CPUs so far, can be successfully accelerated on a co-processor enhanced cluster without any code modifications. The chosen solid mechanics code has higher accuracy requirements and a more diverse CPU/co-processor interaction than the Poisson example, and is thus better suited to assess the practicability of our acceleration approach. We present accuracy experiments, a scalability test and acceleration results for different elastic objects under load. In particular, we demonstrate in detail that the single precision execution of the co-processor does not affect the final accuracy. We establish how the local acceleration gains of factors 5.5 to 9.0 translate into 1.6- to 2.6-fold total speed-up. Subsequent analysis reveals which measures will increase these factors further.

Keywords: heterogeneous computing; parallel scientific computing; computational solid mechanics; GPUs; co-processor integration; multigrid solvers; domain decomposition

Reference to this paper should be made as follows: Göddeke et al. (2008) 'Co-Processor Acceleration of an Unmodified Parallel Solid Mechanics Code with FEASTGPU', Int. J. Computational Science and Engineering, Vol. x, Nos. a/b/c, pp. 1–16.

Biographical notes: Dominik Göddeke and Hilmar Wobker are PhD students, working on advanced computer architectures, computational solid mechanics and HPC for FEM. Robert Strzodka received his PhD from the University of Duisburg-Essen in 2004 and leads the group Integrative Scientific Computing at the Max Planck Center. Jamaludin Mohd-Yusof received his PhD from Cornell University (Aerospace Engineering) in 1996. His research at LANL includes fluid dynamics applications and advanced computer architectures. Patrick McCormick is a Project Leader at LANL, focusing on advanced computer architectures and scientific visualisation. Stefan Turek holds a PhD (1991) and a Habilitation (1998) in Numerical Mathematics from the University of Heidelberg and leads the Institute of Applied Mathematics in Dortmund.

1 INTRODUCTION

While high-end HPC systems continue to be specialised solutions, price/performance and power/performance considerations lead to an increasing interest in HPC systems built from commodity components. In fact, such clusters have been dominating the TOP500 list of supercomputers in the number of deployed installations for several years (Meuer et al., 2007).

Meanwhile, because thermal restrictions have put an end to frequency scaling, codes no longer automatically run faster with each new commodity hardware generation. Now, parallelism and specialisation are considered as the most important design principles to achieve better performance. Soon, CPUs will have tens of parallel cores; future massively parallel chip designs will probably be heterogeneous with general and specialised cores and non-uniform memory access (NUMA) to local storage/caches on the chip.

Currently available specialised co-processors are forerunners of this development and a good testbed for future systems. The GRAPE series clusters (Genomic Sciences Center, RIKEN, 2006), the upgrade of the TSUBAME cluster with the ClearSpeed accelerator boards (Tokyo Institute of Technology, 2006; ClearSpeed Technology, Inc., 2006), or the Maxwell FPGA supercomputer (FPGA High Performance Computing Alliance, 2007) have demonstrated the power efficiency of this approach.

Multimedia processors such as the Cell BE processor or graphics processor units (GPUs) are also considered as potent co-processors for commodity clusters. The absolute power consumption of the corresponding boards is high, because, in contrast to the GRAPE or ClearSpeed boards, they are optimised for high bandwidth data movement, which is responsible for most of the power dissipation. However, the power consumption relative to the magnitude of the data throughput is low, so that these boards do improve the power/performance ratio of a system for data intensive applications such as the Finite Element simulations considered in this paper.

Unfortunately, different co-processors are controlled by different languages and integrated into the system with different APIs. In practice, application programmers are not willing to deal with the resulting complications, and co-processor hardware can only be deployed in the market, if standard high level compilers for the architecture are available or if hardware accelerated libraries for common sub-problems are provided.

1.1 Main Hypothesis

Obviously, not all applications match the specialisation of a given co-processor, but in many cases hardware acceleration can be exploited without fundamental restructuring and reimplementation, which is prohibitively expensive for established codes. In particular, we believe that each co-processor should have at least one parallel language to efficiently utilise its resources, and some associated runtime environment enabling the access to the co-processor from the main code. However, the particular choice of the co-processor and language are of subordinated relevance because productivity reasons limit the amount of code that may be reimplemented for acceleration. Most important are the abstraction from the particular co-processor hardware (such that changes of co-processor and parallel language become manageable) and a global computation scheme that can concentrate resource utilisation in independent fine-grained parallel chunks. Consequently, the main task does not lie in the meticulous tuning of the co-processor code as the hardware will soon be outdated anyway, but rather in the abstraction and global management that remain in use over several hardware generations.

1.2 Contribution

We previously suggested an approach – tailored to the solution of PDE problems – which integrates co-processors not on the kernel level, but as local solvers for local subproblems in a global, parallel solver scheme (Göddeke et al., 2007a). This concentrates sufficient fine-grained parallelism in separate tasks and minimises the overhead of repeated co-processor configuration and data transfer through the relatively narrow PCIe/PCI-X bus. The abstraction layer of the suggested *minimally invasive* integration encapsulates heterogeneities of the system on the node level, so that MPI sees a globally homogeneous system, while the local heterogeneity within the node interacts cleverly with the local solver components.

We assessed the basic applicability of this approach for the scalar Poisson problem, using GPUs as co-processors. They are attractive because of very good price/performance ratios, fairly easy management, and very high memory bandwidth. We encapsulated the hardware specifics of GPUs such that the general application sees only a generic co-processor with certain parallel functionality. Therefore, the focus of the project does not lie in new ways of GPU programming, but rather in algorithm and software design for heterogeneous co-processor enhanced clusters.

In this paper, we use an extended version of this hardware-aware solver toolkit and demonstrate that even a fully developed non-scalar application code can be significantly accelerated, without any code changes to either the application or the previously written accelerator code. The application specific solver based on these components has a more complex data-flow and more diverse CPU/co-processor interaction than the Poisson problem. This allows us to perform a detailed, realistic assessment of the accuracy and speed of co-processor acceleration of unmodified code within this concept. In particular, we quantify the strong scalability effects within the nodes, caused by the addition of parallel co-processors. Our application domain in this paper is Computational Solid Mechanics (CSM), but the approach is widely applicable, for example to the important class of saddlepoint problems arising in Computational Fluid Dynamics (CFD).

1.3 Related Work

We have surveyed co-processor integration in commodity clusters in more detail in Section 1 of a previous publication (Göddeke et al., 2007b).

Erez et al. (2007) present a general framework and evaluation scheme for irregular scientific applications (such as Finite Element computations) on stream processors and architectures like ClearSpeed, Cell BE and Merrimac. Sequoia (Fatahalian et al., 2006) presents a general framework for portable data parallel programming of systems with deep, possibly heterogeneous, memory hierarchies, which can also be applied to a GPU-cluster. Our work distribution is similar in spirit, but more specific to PDE problems and more diverse on the different memory levels.

GPU-enhanced systems have traditionally been deployed for parallel rendering (Humphreys et al., 2002; van der Schaaf et al., 2006) and scientific visualisation (Kirchner et al., 2003; Nirnimesh et al., 2007). One of the largest examples is 'gauss', a 256 node cluster installed by GraphStream at Lawrence Livermore National Laboratory (GraphStream, Inc., 2006). Several parallel non-graphics applications have been ported to GPU clusters. Fan et al. (2004) present a parallel GPU implementation of flow simulation using the Lattice Boltzmann model. Their implementation is 4.6 times faster than an SSE-optimised CPU implementation, and in contrast to FEM, LBM typically does not suffer from reduced precision. Stanford's Folding@Home distributed computing project has deployed a dual-GPU 16 node cluster achieving speedups of 40 over highly tuned SSE kernels (Owens et al., 2008). Recently, GPGPU researchers have started to investigate the benefits of dedicated GPU-based HPC solutions like NVIDIA's Tesla (2008) or AMD's FireStream (2008) technology, but published results usually do not exceed four GPUs (see for example the VMD code by Stone et al. in the survey paper by Owens et al. (2008)).

An introduction to GPU computing and programming aspects is clearly beyond the scope of this paper. For more details, we refer to excellent surveys of techniques, applications and concepts, and to the GPGPU community website (Owens et al., 2008, 2007; GPGPU, 2004–2008).

1.4 Paper Overview

In Section 2 the theoretical background of solid mechanics in the context of this paper is presented, while our mathematical and computational solution strategy is described in Section 3. In Section 4 we revisit our minimally invasive approach to integrate GPUs as co-processors in the overall solution process without changes to the application code. We present our results in three different categories: In Section 5 we show that the restriction of the GPU to single precision arithmetic does not affect the accuracy of the computed results in any way, as a consequence of our solver design. Weak scalability is demonstrated on up to 64 nodes and half a billion unknowns in Section 6. We study the performance of the accelerated solver scheme in Section 7 in view of absolute timings and the strong scalability effects introduced by the co-processor. Section 8 summarises the paper and briefly outlines future work.

For an accurate notation of bandwidth transfer rates given in metric units (e.g. 8 GB/s) and memory capacity given in binary units (e.g. 8 GiB) we use the International standard IEC60027-2 in this paper: G= 10^9, Gi= 2^{30} and similar for Ki, Mi.

2 COMPUTATIONAL SOLID MECHANICS

In Computational Solid Mechanics (CSM) the deformation of solid bodies under external loads is examined. We consider a two-dimensional body covering a domain $\bar{\Omega} = \Omega \cup \partial\Omega$, where Ω is a bounded, open set with boundary $\Gamma = \partial\Omega$. The boundary is split into two parts: the Dirichlet part Γ_D where displacements are prescribed and the Neumann part Γ_N where surface forces can be applied ($\Gamma_\mathrm{D} \cap \Gamma_\mathrm{N} = \emptyset$). Furthermore the body can be exposed to volumetric forces, e. g. gravity. We treat the simple, but nevertheless fundamental, model problem of elastic, compressible material under static loading, assuming small deformations. We use a formulation where the displacements $\boldsymbol{u}(\boldsymbol{x}) = \big(u_1(\boldsymbol{x}), u_2(\boldsymbol{x})\big)^\mathsf{T}$ of a material point $\boldsymbol{x} \in \bar{\Omega}$ are the only unknowns in the equation. The strains can be defined by the linearised strain tensor $\varepsilon_{ij} = \frac{1}{2}\left(\frac{\partial u_i}{\partial x_j} + \frac{\partial u_j}{\partial x_i}\right), \quad i,j = 1,2$, describing the linearised kinematic relation between displacements and strains. The material properties are reflected by the constitutive law, which determines a relation between the strains and the stresses. We use Hooke's law for isotropic elastic material, $\boldsymbol{\sigma} = 2\mu\boldsymbol{\varepsilon} + \lambda\,\mathrm{tr}(\boldsymbol{\varepsilon})\boldsymbol{I}$, where $\boldsymbol{\sigma}$ denotes the symmetric stress tensor and μ and λ are the so-called Lamé constants, which are connected to the Young modulus E and the Poisson ratio ν as follows:

$$\mu = \frac{E}{2(1+\nu)}, \qquad \lambda = \frac{E\nu}{(1+\nu)(1-2\nu)} \tag{1}$$

The basic physical equations for problems of solid mechanics are determined by equilibrium conditions. For a body in equilibrium, the inner forces (stresses) and the outer forces (external loads $\boldsymbol{f}$) are balanced:

$$-\,\mathbf{div}\,\boldsymbol{\sigma} = \boldsymbol{f}, \qquad \boldsymbol{x} \in \Omega.$$

Using Hooke's law to replace the stress tensor, the problem of linearised elasticity can be expressed in terms of the following elliptic boundary value problem, called the Lamé equation:

$$-2\mu\,\mathbf{div}\,\boldsymbol{\varepsilon}(\boldsymbol{u}) - \lambda\,\mathbf{grad}\,\mathrm{div}\,\boldsymbol{u} = \boldsymbol{f}, \qquad \boldsymbol{x} \in \Omega \tag{2a}$$

$$\boldsymbol{u} = \boldsymbol{g}, \qquad \boldsymbol{x} \in \Gamma_\mathrm{D} \tag{2b}$$

$$\boldsymbol{\sigma}(\boldsymbol{u}) \cdot \boldsymbol{n} = \boldsymbol{t}, \qquad \boldsymbol{x} \in \Gamma_\mathrm{N} \tag{2c}$$

Here, $\boldsymbol{g}$ are prescribed displacements on Γ_D, and $\boldsymbol{t}$ are given surface forces on Γ_N with outer normal $\boldsymbol{n}$. For details on the elasticity problem, see for example Braess (2001).

3 SOLUTION STRATEGY

To solve the elasticity problem, we use FEASTSOLID, an application built on top of FEAST, our toolkit providing Finite Element discretisations and corresponding optimised parallel multigrid solvers for PDE problems. In FEAST, the discretisation is closely coupled with the domain decomposition for the parallel solution: The computational domain $\bar{\Omega}$ is covered with a collection of quadrilateral subdomains $\bar{\Omega}_i$. The subdomains form an unstructured coarse mesh (cf. Figure 8 in Section 7), and are hierarchically refined so that the resulting mesh is used for the discretisation with Finite Elements. Refinement is performed such as to preserve a logical tensorproduct structure of the mesh cells within each subdomain. Consequently, FEAST maintains a clear separation of globally unstructured and locally structured data. This approach has many advantageous properties which we outline in this Section and Section 4. For more details on FEAST, we refer to Turek et al. (2003) and Becker (2007).

3.1 Parallel Multigrid Solvers in FEAST

For the problems we are concerned with in the (wider) context of this paper, multigrid methods are obligatory from a numerical point of view. When parallelising multigrid methods, numerical robustness, numerical efficiency and (weak) scalability are often contradictory properties: A strong recursive coupling between the subdomains, for instance by the direct parallelisation of ILU-like smoothers, is advantageous for the numerical efficiency of the multigrid solver. However, such a coupling increases the communication and synchronisation requirements significantly and is therefore bound to scale badly. To alleviate this high communication overhead, the recursion is usually relaxed to the application of *local smoothers* that act on each subdomain independently. The contributions of the separate subdomains are combined in an additive manner only after the smoother has been applied to all subdomains, without any data exchange during the smoothing. The disadvantage of such a (in terms of domain decomposition) *block-Jacobi* coupling is that typical local smoothers are usually not powerful enough to treat, for example, local anisotropies. Consequently, the numerical efficiency of the multigrid solver is dramatically reduced (Smith et al., 1996; Turek et al., 2003).

To address these contradictory needs, FEAST employs a generalised multigrid domain decomposition concept. The basic idea is to apply a *global* multigrid algorithm which is smoothed in an additive manner by *local* multigrids acting on each subdomain independently. In the nomenclature of the previous paragraph, this means that the application of a local smoother translates to performing few iterations – in the experiments in this paper even only one iteration – of a local multigrid solver, and we can use the terms *local smoother* and *local multigrid* synonymously. This cascaded multigrid scheme is very robust as local irregularities are 'hidden' from the outer solver, the global multigrid provides strong global coupling (as it acts on all levels of refinement), and it exhibits good scalability by design. Obviously, this cascaded multigrid scheme is prototypical in the sense that it can only show its full strength for reasonably large local problem sizes and ill-conditioned systems (Becker, 2007).

Global Computations

Instead of keeping all data in one general, homogeneous data structure, FEAST stores only local FE matrices and vectors, corresponding to the subdomains. Global matrix-vector operations are performed by a series of local operations on matrices representing the restriction of the 'virtual' global matrix on each subdomain. These operations are *directly* followed by exchanging information via MPI over the boundaries of neighbouring subdomains, which can be implemented asynchronously without any global barrier primitives. There is only an implicit subdomain overlap, the domain decomposition is implemented via special boundary conditions in the local matrices (Becker, 2007). Several subdomains are typically grouped into one MPI process, exchanging data via shared memory.

To solve the coarse grid problems of the global multigrid scheme, we use a tuned direct LU decomposition solver from the UMFPACK library (Davis, 2004) which is executed on the master process while the compute processes are idle.

Local Computations

Finite Element codes are known to be limited in performance by memory latency in case of many random memory accesses, and memory bandwidth otherwise, rather than by raw compute performance. This is in general known as the *memory wall problem*. FEAST tries to alleviate this problem by exploiting the logical tensorproduct structure of the subdomains. Independently on each grid level, we enumerate the degrees of freedom in a line-wise fashion such that the local FE matrices corresponding to scalar equations exhibit a band structure with fixed band offsets. Instead of storing the matrices in a CSR-like format which implies indirect memory access in the computation of a matrix-vector multiplication, we can store each band (with appropriate offsets) individually, and perform matrix-vector multiplication with direct access to memory only. The asymptotic performance gain of this approach is a factor of two (one memory access per matrix entry instead of two), and usually higher in practice as block memory transfers and techniques for spatial and temporal locality can be employed instead of excessive pointer chasing and irregular memory access patterns. The explicit knowledge of the matrix structure is analogously exploited not only for parallel linear algebra operations, but also, for instance, in the design of highly tuned, very powerful smoothing operators (Becker, 2007).

The logical tensorproduct structure of the underlying mesh has an additional important advantage: Grid transfer operations during the local multigrid can be expressed as

matrices with constant coefficients (Turek, 1999), and we can directly use the corresponding stencil values without the need to store and access a matrix expressing an arbitrary transfer function. This significantly increases performance, as grid transfer operations are reduced to efficient vector scaling, reduction and expansion operations.

The small local coarse grid problems are solved by performing few iterations of a preconditioned conjugate gradient algorithm.

In summary, Figure 1 illustrates a typical solver in FEAST. The notation 'local multigrid (V 4+4, S, CG)' denotes a multigrid solver on a single subdomain, configured to perform a V cycle with 4 pre- and postsmoothing steps with the smoothing operator $S \in \{\text{Jacobi, Gauss-Seidel, ILU}, \ldots\}$, using a conjugate gradient algorithm on the coarsest grid. To improve solver robustness, the global multigrid solver is used as a preconditioner to a Krylov subspace solver such as BiCGStab which executes on the global fine grid. As a preconditioner, the global multigrid performs exactly one iteration without convergence control.

Everything up to the local multigrid executes on the CPUs in double precision. The computational precision of the local multigrid may vary depending on the architecture.

global BiCGStab
preconditioned by
  global multigrid (V 1+1)
  additively smoothed by
    for all $\bar{\Omega}_i$: **local multigrid** (V 4+4, S, CG)
  coarse grid solver: LU decomposition

Figure 1: Illustration of the family of cascaded multigrid solver schemes in FEAST. The accelerable parts of the algorithm (cf. Section 4) are highlighted.

We finally emphasise that the entire concept – comprising domain decomposition, solver strategies and data structures – is independent of the spatial dimension of the underlying problem. Implementation of 3D support is tedious and time-consuming, but does not pose any principal difficulties.

3.2 Scalar and Vector-Valued Problems

The guiding idea to treating vector-valued problems with FEAST is to rely on the modular, reliable and highly optimised scalar local multigrid solvers on each subdomain, in order to formulate robust schemes for a wide range of applications, rather than using the best suited numerical scheme for each application and go through the optimisation and debugging process over and over again. Vector-valued PDEs as they arise for instance in solid mechanics (CSM) and fluid dynamics (CFD) can be rearranged and discretised in such a way that the resulting discrete systems of equations consist of blocks that correspond to scalar problems (for the CSM case see beginning of Section 3.3). Due to this special block-structure, all operations required to solve the systems can be implemented as a series of operations for scalar systems (in particular matrix-vector operations, dot products and grid transfer operations in multigrid), taking advantage of the highly tuned linear algebra components in FEAST. To apply a scalar local multigrid solver, the set of unknowns corresponding to a global scalar equation is restricted to the subset of unknowns that correspond to the specific subdomain.

To illustrate the approach, consider a matrix-vector multiplication $\mathbf{y} = \mathbf{A}\mathbf{x}$ with the exemplary block structure:

$$\begin{pmatrix}\mathbf{y}_1\\ \mathbf{y}_2\end{pmatrix} = \begin{pmatrix}\mathbf{A}_{11} & \mathbf{A}_{12}\\ \mathbf{A}_{21} & \mathbf{A}_{22}\end{pmatrix}\begin{pmatrix}\mathbf{x}_1\\ \mathbf{x}_2\end{pmatrix}$$

As explained above, the multiplication is performed as a series of operations on the local FE matrices per subdomain $\bar{\Omega}_i$, denoted by superscript $(\cdot)^{(i)}$. The global scalar operators, corresponding to the blocks in the matrix, are treated individually:

For $j = 1, 2$, do

1. For all $\bar{\Omega}_i$, compute $\mathbf{y}_j^{(i)} = \mathbf{A}_{j1}^{(i)}\mathbf{x}_1^{(i)}$.
2. For all $\bar{\Omega}_i$, compute $\mathbf{y}_j^{(i)} = \mathbf{y}_j^{(i)} + \mathbf{A}_{j2}^{(i)}\mathbf{x}_2^{(i)}$.
3. Communicate entries in $\mathbf{y}_j$ corresponding to the boundaries of neighbouring subdomains.

3.3 Solving the Elasticity Problem

In order to solve vector-valued linearised elasticity problems with the application FEASTSOLID using the FEAST intrinsics outlined in the previous paragraphs, it is essential to order the resulting degrees of freedom corresponding to the spatial directions, a technique called separate displacement ordering (Axelsson, 1999). In the 2D case where the unknowns $\boldsymbol{u} = (u_1, u_2)^\mathsf{T}$ correspond to displacements in x and y-direction, rearranging the left hand side of equation (2a) yields:

$$-\begin{pmatrix}(2\mu+\lambda)\partial_{xx} + \mu\partial_{yy} & (\mu+\lambda)\partial_{xy}\\ (\mu+\lambda)\partial_{yx} & \mu\partial_{xx} + (2\mu+\lambda)\partial_{yy}\end{pmatrix}\begin{pmatrix}u_1\\ u_2\end{pmatrix} = \begin{pmatrix}f_1\\ f_2\end{pmatrix} \quad (3)$$

We approximate the domain $\bar{\Omega}$ by a collection of several subdomains $\bar{\Omega}_i$, each of which is refined to a logical tensorproduct structure as described in Section 3.1. We consider the weak formulation of equation (3) and apply a Finite Element discretisation with conforming bilinear elements of the Q_1 space. The vectors and matrices resulting from the discretisation process are denoted with upright bold letters, such that the resulting linear equation system can be written as $\mathbf{K}\mathbf{u} = \mathbf{f}$. Corresponding to representation (3) of the continuous equation, the discrete system has the following block structure,

$$\begin{pmatrix}\mathbf{K}_{11} & \mathbf{K}_{12}\\ \mathbf{K}_{21} & \mathbf{K}_{22}\end{pmatrix}\begin{pmatrix}\mathbf{u}_1\\ \mathbf{u}_2\end{pmatrix} = \begin{pmatrix}\mathbf{f}_1\\ \mathbf{f}_2\end{pmatrix}, \quad (4)$$

where $\mathbf{f} = (\mathbf{f}_1, \mathbf{f}_2)^T$ is the vector of external loads and $\mathbf{u} = (\mathbf{u}_1, \mathbf{u}_2)^T$ the (unknown) coefficient vector of the FE solution. The matrices $\mathbf{K}_{11}$ and $\mathbf{K}_{22}$ of this block-structured system correspond to scalar elliptic operators (cf. Equation (3)). It is important to note that (in the domain decomposition sense) this also holds for the restriction of the equation to each subdomain $\bar{\Omega}_i$, denoted by $\mathbf{K}_{jj}^{(i)}\mathbf{u}_j^{(i)} = \mathbf{f}_j^{(i)}$, $j = 1, 2$. Consequently, due to the local generalised tensor-product structure, FEAST's tuned solvers can be applied on each subdomain, and as the system as a whole is block-structured, the general solver (see Figure 1) is applicable. Note, that in contrast to our previous work with the Poisson problem (Göddeke et al., 2007a), the scalar elliptic operators appearing in equation (3) are *anisotropic*. The degree of anisotropy a_{op} depends on the material parameters (see Equation (1)) and is given by

$$a_{\text{op}} = \frac{2\mu + \lambda}{\mu} = \frac{2 - 2\nu}{1 - 2\nu}. \tag{5}$$

We illustrate the details of the solution process with a basic iteration scheme, a preconditioned defect correction method:

$$\mathbf{u}^{k+1} = \mathbf{u}^k + \omega\tilde{\mathbf{K}}_{\text{B}}^{-1}(\mathbf{f} - \mathbf{K}\mathbf{u}^k) \tag{6}$$

This iteration scheme acts on the global system (4) and thus couples the two sets of unknowns $\mathbf{u}_1$ and $\mathbf{u}_2$. The *block-preconditioner* $\tilde{\mathbf{K}}_{\text{B}}$ explicitly exploits the block structure of the matrix $\mathbf{K}$. We use a *block-Gauss-Seidel* preconditioner $\tilde{\mathbf{K}}_{\text{BGS}}$ in this paper (see below). One iteration of the global defect correction scheme consists of the following three steps:

1. Compute the global defect (cf. Section 3.2):

$$\begin{pmatrix}\mathbf{d}_1\\ \mathbf{d}_2\end{pmatrix} = \begin{pmatrix}\mathbf{K}_{11} & \mathbf{K}_{12}\\ \mathbf{K}_{21} & \mathbf{K}_{22}\end{pmatrix}\begin{pmatrix}\mathbf{u}_1^k\\ \mathbf{u}_2^k\end{pmatrix} - \begin{pmatrix}\mathbf{f}_1\\ \mathbf{f}_2\end{pmatrix}$$

2. Apply the block-preconditioner

$$\tilde{\mathbf{K}}_{\text{BGS}} := \begin{pmatrix}\mathbf{K}_{11} & \mathbf{0}\\ \mathbf{K}_{21} & \mathbf{K}_{22}\end{pmatrix}$$

   by approximately solving the system $\tilde{\mathbf{K}}_{\text{BGS}}\mathbf{c} = \mathbf{d}$. This is performed by two scalar solves per subdomain and one global (scalar) matrix-vector multiplication:

   (a) For each subdomain $\bar{\Omega}_i$, solve $\mathbf{K}_{11}^{(i)}\mathbf{c}_1^{(i)} = \mathbf{d}_1^{(i)}$.

   (b) Update RHS: $\mathbf{d}_2 = \mathbf{d}_2 - \mathbf{K}_{21}\mathbf{c}_1$.

   (c) For each subdomain $\bar{\Omega}_i$, solve $\mathbf{K}_{22}^{(i)}\mathbf{c}_2^{(i)} = \mathbf{d}_2^{(i)}$.

3. Update the global solution with the (eventually damped) correction vector: $\mathbf{u}^{k+1} = \mathbf{u}^k + \omega\mathbf{c}$

Instead of the illustrative defect correction scheme outlined above, our full solver is a multigrid iteration in a (V 1+1) configuration. The procedure is identical: During the restriction phase, global defects are smoothed by the block-Gauss-Seidel approach, and during the prolongation phase, correction vectors are treated analogously. Figure 2 summarises the entire scheme. Note the similarity to the general template solver in Figure 1, and that this specialised solution scheme is entirely constructed from FEAST intrinsics.

global BiCGStab
preconditioned by
  global multigrid (V 1+1)
  additively smoothed (block-Gauss-Seidel) by
    for all $\bar{\Omega}_i$: solve $\mathbf{K}_{11}^{(i)}\mathbf{c}_1^{(i)} = \mathbf{d}_1^{(i)}$ by
      local multigrid (V 4+4, Jacobi, CG)
    update RHS: $\mathbf{d}_2 = \mathbf{d}_2 - \mathbf{K}_{21}\mathbf{c}_1$
    for all $\bar{\Omega}_i$: solve $\mathbf{K}_{22}^{(i)}\mathbf{c}_2^{(i)} = \mathbf{d}_2^{(i)}$ by
      local multigrid (V 4+4, Jacobi, CG)
  coarse grid solver: LU-decomposition

Figure 2: Our solution scheme for the elasticity equations. The accelerable parts of the algorithm (cf. Section 4) are highlighted.

4 CO-PROCESSOR INTEGRATION

4.1 Minimally Invasive Integration

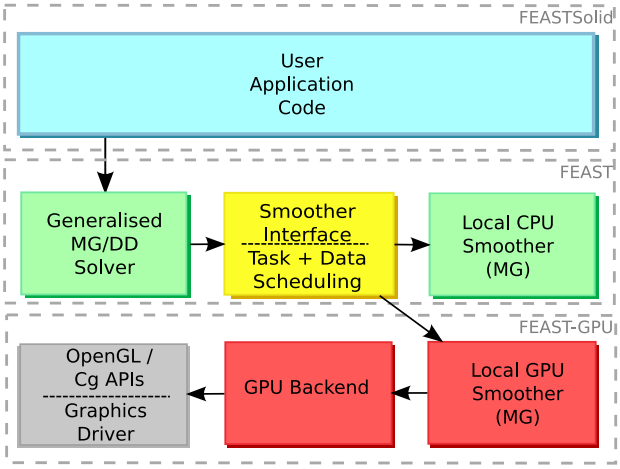

Figure 3: Interaction between FEASTSOLID, FEAST and FEASTGPU.

Figure 3 illustrates how the elasticity application FEASTSOLID, the core toolbox FEAST and the GPU accelerated library FEASTGPU interact. The control flow of the global recursive multigrid solvers (see Figure 2) is realised via one central interface, which is responsible for scheduling both tasks and data. FEASTGPU adds GPU support, by replacing the local multigrid solvers (see Figure 2) that act on the individual subdomains, with a GPU implementation. Then, the GPU serves as a *local smoother* to the outer multigrid (cf. Section 3.1). The GPU smoother implements the same interface as the existing local CPU smoothers, consequently, comparatively

few changes in FEAST's solver infrastructure were required for their integration, e.g. changes to the task scheduler and the parser for parameter files. Moreover, this integration is completely independent of the type of accelerator we use, and we will explore other types of hardware in future work. For more details on this minimally invasive hardware integration into established code we refer to Göddeke et al. (2007a).

Assigning the GPU the role of a local smoother means that there is a considerable amount of work on the GPU, before data must be transferred back to the host to be subsequently communicated to other processes via MPI. Such a setup is very advantageous for the application of hardware accelerators in general, because the data typically has to travel through a bandwidth bottleneck (1-4 GB/s PCIe compared to more than 80 GB/s video memory) to the accelerator, and this overhead can only be amortised by a faster execution time if there is enough local work. In particular, this bottleneck makes it impossible to accelerate the local portions of global operations, e. g. defect calculations. The impact of reduced precision on the GPU is also minimised by this approach, as it can be interpreted as a mixed precision iterative refinement scheme, which is well-suited for the kind of problems we are dealing with (Göddeke et al., 2007c). We analyse the achieved accuracy in detail in Section 5.

4.2 Process Scheduling

In a homogeneous cluster environment like in our case where each node has exactly the same (potentially internally heterogeneous) specification, the assignment of MPI processes to the nodes is easy to manage. In the heterogeneous case where the configuration of the individual nodes differs, we are faced with a complicated multidimensional dynamic scheduling problem which we have not addressed yet. For instance, jobs can be differently sized depending on the specification of the nodes, the accumulated transfer and compute performance of the graphics cards compared to the CPUs etc. For the tests in this paper, we use static partitions and only modify FEAST's scheduler to be able to distribute jobs based on hard-coded rules.

An example of such a hard-coded rule is the dynamic rescheduling of small problems (less than 2000 unknowns), for which the configuration overhead would be very high, from the co-processor back to the CPU, which executes them much faster from its cache. For more technical details on this rule and other tradeoffs, we refer to our previous publication (Göddeke et al., 2007a).

4.3 FeastGPU - The GPU Accelerated Library

While the changes to FEAST that enable the co-processor integration are hardware independent, the co-processor library itself is hardware specific and has to deal with the peculiarities of the programming languages and development tools. In case of the GPU being used as a scientific co-processor in the cluster, FEASTGPU implements the data transfer and the multigrid computation.

The data transfer from the CPU to the GPU and vice versa seems trivial at first sight. However, we found this part to be most important to achieve good performance. In particular it is crucial to avoid redundant copy operations. For the vector data, we perform the format conversion from double to single precision on the fly during the data transfer of the local right hand side vector (the input to the smoother); and from single to double precision during the accumulation of the smoothing result from the GPU with the previous iterate on the CPU. This treatment minimises the overhead and in all our experiments we achieved best performance with this approach. For the matrix data, we model the GPU memory as a large L3 cache with either automatic or manual prefetching. For graphics cards with a sufficient amount of video memory ($\geq$512 MiB), we copy all data into driver-controlled memory in a preprocessing step and rely on the graphics driver to page data in and out of video memory as required. For older hardware with limited video memory ($\leq$128 MiB), we copy all matrix data associated with one subdomain manually to the graphics card, immediately before using it. Once the data is in video memory, one iteration of the multigrid solver can execute on the GPU without further costly transfers to or from the main memory.

The implementation of the multigrid scheme relies on various operators (matrix-vector multiplications, grid transfers, coarse grid solvers, etc.) collected in the GPU backend. We use the graphics-specific APIs OpenGL and Cg for the concrete implementation of these operators. In case of a basic V-cycle with a simple Jacobi smoother this is not a difficult task on DirectX9c capable GPUs, for details see papers by Bolz et al. (2003) and Goodnight et al. (2003). In our implementation we can also use F- and W-cycles in the multigrid scheme, but this is a feature of the control flow that executes on the CPU and not the GPU. It is a greater challenge to implement complex local smoothers, like ADI-TRIGS that is used in Section 5.3 on the CPU, because of the sequential dependencies in the computations. The new feature of local user-controlled storage in NVIDIA's G80 architecture accessible through the CUDA programming environment (NVIDIA Corporation, 2007), will help in resolving such dependencies in parallel. But before designing more optimised solutions for a particular GPU-generation, we want to further evaluate the minimally invasive hardware integration into existing code on a higher level.

5 ACCURACY STUDIES

On the CPU, we use double precision exclusively. An important benefit of our solver concept and corresponding integration of hardware acceleration is that the restriction of the GPU to single precision has no effect on the final accuracy and the convergence of the global solver. To verify this claim, we perform three different numerical experiments and increase the condition number of the global system

and the local systems. As we want to run the experiments for the largest problem sizes possible, we use an outdated cluster named DQ, of which 64 nodes are still in operation. Unfortunately, this is the only cluster with enough nodes available to us at the time of writing. Each node contains one NVIDIA Quadro FX 1400 GPU with only 128 MiB of video memory; these boards are three generations old. As newer hardware generations provide better compliance with the IEEE 754 single precision standard, the accuracy analysis remains valid for better GPUs.

GPUs that support double precision in hardware are already becoming available (AMD Inc., 2008). However, by combining the numerical experiments in this Section with the performance analysis in Section 7.2, we demonstrate that the restriction to single precision is actually a performance advantage. In the long term, we expect single precision to be 4x faster for compute-intensive applications (transistor count) and 2x faster for data-intensive applications (bandwidth).

Unless otherwise indicated, in all tests we configure the solver scheme (cf. Figure 2) to reduce the initial residuals by 6 digits, the global multigrid performs one pre- and postsmoothing step in a V cycle, and the inner multigrid uses a V cycle with four smoothing steps. As explained in Section 4.1, we accelerate the plain CPU solver with GPUs by replacing the scalar, local multigrid solvers with their GPU counterparts, otherwise, the parallel solution scheme remains unchanged. We statically schedule 4 subdomains per cluster node, and refine each subdomain 7-10 times. A refinement level of L yields $2(2^L+1)^2$ DOF (degrees of freedom) per subdomain, so the maximum problem size in the experiments in this Section is 512 Mi DOF for refinement level $L = 10$.

5.1 Analytic Reference Solution

This numerical test uses a unitsquare domain, covered by 16, 64 or 256 subdomains which are refined 7 to 10 times. We define a parabola and a sinusoidal function for the x and y displacements, respectively, and use these functions and the elasticity equation (2) to prescribe a right hand side for the global system, so that we know the exact analytical solution. This allows us to compare the L_2 errors (integral norm of the difference between computed FE function and analytic solution), which according to FEM theory are reduced by a factor of 4 (h^2) in each refinement step (Braess, 2001).

Figure 4 illustrates the main results. We first note that all configurations require exactly four iterations of the global solver. Most importantly, the differences between CPU and GPU runs are in the noise, independent of the level of refinement or the size of the problem. In fact, in the figure the corresponding CPU and GPU results are plotted directly on top of each other. We nevertheless prefer illustrating these results with a figure instead of a table, because it greatly simplifies the presentation: Looking at the graphs vertically, we see the global error reduction by a factor of 4 with increasing level of refinement L. The

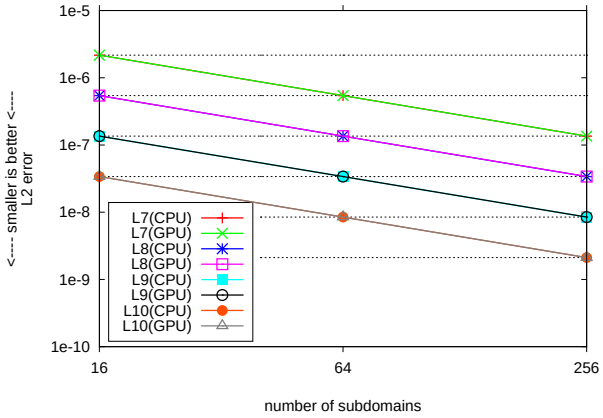

Figure 4: Error reduction of the elasticity solver in the L_2 norm.

independence of the subdomain distribution and the level of refinement can be seen horizontally, for instance 16 subdomains on level 8 are equivalent to 64 level-7 subdomains.

5.2 Global Ill-conditioning: Cantilever Beam Configuration

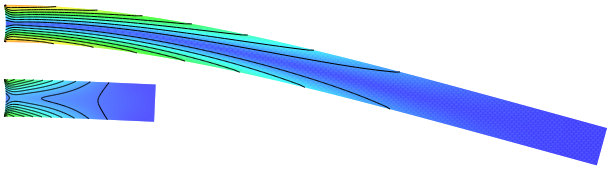

Figure 5: Computed displacements and von Mises stresses for the `BEAM` configuration with anisotropy 1:16 (top) and 1:4 (bottom).

The cantilever beam test (`BEAM`) is a standard benchmark configuration in CSM, and is known to be difficult to solve numerically (Braess, 2001; Ovtchinnikov and Xanthis, 1998). A long, thin beam is horizontally attached at one end and the gravity force pulls it uniformly in the y-direction (see Figure 5). We partition the geometry in such a way that the number of compute nodes is proportional to the length of the beam, and test two configurations: one consisting of 8×2 square subdomains (distributed to 4 nodes), the other of 32×2 square subdomains (16 nodes), resulting in a global domain anisotropy of 4:1 and 16:1, respectively, and a maximum problem size of 32 Mi and 128 Mi DOF, respectively. This high degree of domain anisotropy in conjunction with the large ratio between free Neumann boundary and fixed Dirichlet boundary (only the narrow side at one end is fixed) and the high level of refinement results in a very ill-conditioned global system (Ovtchinnikov and Xanthis, 1998; Axelsson, 1999). To illustrate this, we first use a simple unpreconditioned conjugate gradient method whose iteration count grows with

aniso04 refinement L	Iterations CPU	Iterations GPU	Volume CPU	Volume GPU	y-Displacement CPU	y-Displacement GPU
8	4	4	1.6087641E-3	1.6087641E-3	-2.8083499E-3	-2.8083499E-3
9	4	4	1.6087641E-3	1.6087641E-3	-2.8083628E-3	-2.8083628E-3
10	4.5	4.5	1.6087641E-3	1.6087641E-3	-2.8083667E-3	-2.8083667E-3
aniso16						
8	6	6	6.7176398E-3	6.7176398E-3	-6.6216232E-2	-6.6216232E-2
9	**6**	**5.5**	6.7176427E-3	6.7176427E-3	-6.6216551E-2	-6.6216552E-2
10	5.5	5.5	6.7176516E-3	6.7176516E-3	-6.6217501E-2	-6.6217502E-2

Table 1: Iterations and computed results for the BEAM configuration with anisotropy of 1:4 (top) and 1:16 (bottom). Differences are highlighted in bold face.

the condition number of the system and is thus a suitable indicator thereof.

For small problem sizes we solve the two beam configurations described above, as well as a third variant – a very short 'beam' with global 'anisotropy' of 1:1. The latter configuration is used exclusively in this test to emphasise the dependency on the global anisotropy. Figure 6 shows the iteration numbers of the conjugate gradient solver for varying degrees of freedom. Reading the graphs vertically (for a fixed number of DOF), shows a significant rise of iteration numbers due to the increasing degree of anisotropy (note that the y-axis is scaled logarithmically). For the isotropic and mildly anisotropic beam we can clearly observe that one grid refinement doubles the number of iterations, which is the expected behaviour of CG. For the strongly anisotropic beam, however, this is no longer true on the highest refinement levels, where the precise iteration counts are 8389, 21104 and 47522, showing a factor which is clearly greater than 2.

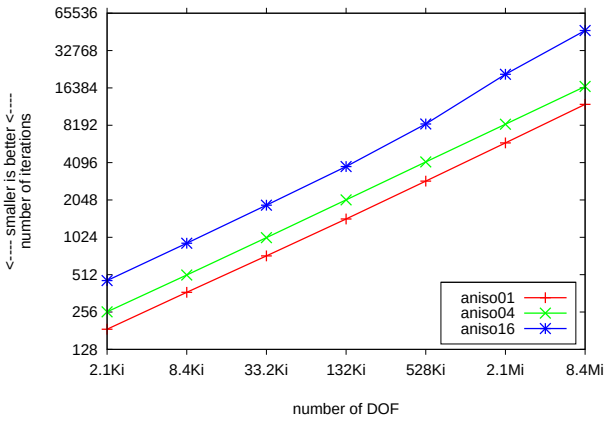

Figure 6: Illustration of the ill-conditioning: BEAM configuration with a simple CG solver. Note: Logscale on y-axis.

Table 1 contains the results we achieved with our multigrid solver for the cantilever beam configuration. The fractional iteration count is a consequence of the global BiCGStab solver permitting an 'early exit' after the first of the two applications of the preconditioner (an entire parallel multigrid iteration), if the scheme has already converged. As there exists no analytic reference solution in this case, we use the displacement (in y-direction) of the midpoint of the free side of the deformed beam and its volume as features of the solutions to compare the computed CPU and GPU results, for increasing levels of refinement L and corresponding problem sizes.

We see no accuracy difference except for floating point noise between the CPU and the GPU configurations, although the CPU configuration uses double precision everywhere and the GPU configuration solves the local multigrid in single precision. This is a very important advantage of our minimally invasive co-processor acceleration and corresponding solver concept: As the global anisotropies and ill-conditioning are entirely hidden from the local multigrids (which see locally isotropic problems on their respective subdomains), the potentially negative impact caused by the limited precision of the GPUs does not come into play.

Overall, we see the expected rise of iteration numbers when elongating the beam and thus increasing the system's condition number. But apart from some granularity effects that are inevitable within our solver concept, we see good level independence and, in particular, identical convergence of the CPU and the GPU variants.

5.3 Local Anisotropies: Towards Incompressible Material

While the previous subsection examined how our solver concept performs for ill-conditioned system matrices due to global domain anisotropies, we now analyse the impact of local anisotropies. For this test we use a standard benchmark configuration in CSM which is often used for testing Finite Element formulations in the context of finite deformations (Reese et al., 1999): A rectangular block is vertically compressed by a surface load. To induce local anisotropies, we increase the Poisson ratio ν of the material, which according to Equation (5) changes the anisotropy of the elliptic operators in Equation (3). Physically, this means that the material becomes more incompressible. Since we use a pure displacement FE formulation, the value of ν must be bounded away from 0.5, otherwise the effect of *volume locking* would hinder convergence to the correct solution (Braess, 2001). Equation (1) shows that the critical parameter λ tends to infinity for $\nu \to 0.5$. As we only have a simple Jacobi smoother available on the GPU, we have to increase the number of Jacobi smoothing steps of the inner solver (the outer solver's configuration remains unchanged) to ensure convergence. All

L	V-4+4 Jacobi				V-8+8 Jacobi				V-2+2 ADI-TRIGS			
nu=0.40	Iters.		Time/Iters.		Iters.		Time/Iters.		Iters.		Time/Iters.	
$a_{\text{op}} = 6$	CPU	GPU	CPU	GPU	CPU	GPU	CPU	GPU	CPU	GPU	CPU	GPU
8	4	4	3.3	4.9	3.5	3.5	4.5	5.7	3.5	n/a	3.6	n/a
9	4	4	11.1	11.2	3.5	3.5	15.9	13.4	3.5	n/a	12.0	n/a
10	4	4	48.2	41.2	3.5	3.5	69.7	52.2	3.5	n/a	49.3	n/a
nu=0.45	Iters.		Time/Iters.		Iters.		Time/Iters.		Iters.		Time/Iters.	
$a_{\text{op}} = 11$	CPU	GPU	CPU	GPU	CPU	GPU	CPU	GPU	CPU	GPU	CPU	GPU
8	4.5	4.5	3.2	4.9	4.5	4.5	4.3	5.5	4.5	n/a	3.4	n/a
9	5	5	11.0	11.1	4.5	4.5	15.6	13.2	4	n/a	11.8	n/a
10	5	5	48.1	41.1	4.5	4.5	69.3	52.1	4	n/a	49.4	n/a
nu=0.48	Iters.		Time/Iters.		Iters.		Time/Iters.		Iters.		Time/Iters.	
$a_{\text{op}} = 26$	CPU	GPU	CPU	GPU	CPU	GPU	CPU	GPU	CPU	GPU	CPU	GPU
8	7.5	7.5	3.0	4.6	6.5	6.5	4.1	5.5	6.5	n/a	3.1	n/a
9	7.5	7.5	10.8	11.0	6.5	6.5	15.4	13.5	6.5	n/a	11.5	n/a
10	7	7	47.9	42.1	6.5	6.5	69.2	52.4	6.5	n/a	49.0	n/a

Table 2: Convergence behaviour of the CPU and the GPU solver with increasing degree of operator anisotropy. To concentrate on general tendencies, the timings are normalised by the number of iterations.

experiments are again performed on 64 nodes of the DQ cluster, yielding a maximum problem size of 512 Mi DOF for the highest refinement level ($L = 10$). As this cluster is outdated and its GPUs come from an even older technology generation than its CPUs, only changes in relative timings (CPU/GPU) are relevant.

For clarity, Table 2 does not contain accuracy results, but we have confirmed that in this series of tests we get identical results (up to numerical noise) for the CPU and the GPU just as in the previous two experiments.

Several important observations can be made from the data listed in Table 2. First, as the increasing value of ν affects the complete system (3), the number of solver iterations rises accordingly. Second, the number of iterations required to solve the system is reduced by increased local smoothing (the anisotropy of the elliptic operators is hidden better from the outer solver), and occasional granularity effects disappear. Finally, the accelerated solver behaves identically to the unaccelerated one, in other words, even if there are effects due to the GPU's reduced precision, they are completely encapsulated from the outer solver and do not influence its convergence. Looking at the (normalised) timings in detail, we see that on the CPU, performing twice as many smoothing steps in the inner multigrid results in a 40-50% increase in runtime for the highest level of refinement, while on the GPU, only 20-25% more time is required. There are two reasons for this behaviour: On the CPU, the operations for high levels of refinement are performed completely out of cache, while on the GPU, full bandwidth is only available to the application for large input sizes, as the overhead costs associated with launching compute kernels is less dominant. Second, as the amount of video memory on these outdated GPUs is barely large enough to hold all matrix data associated with one subdomain, the high cost of paging data in and out of memory is amortised much better when more operations are performed on the same data.

To verify that our test results are not influenced by the weakness of the Jacobi smoother, we perform all tests again with a very powerful alternating directions tridiagonal Gauss-Seidel smoother (ADI-TRIGS, see Becker (2007)), for which two smoothing steps (one in each direction) suffice. This powerful smoother is currently only available on the CPU. Table 2 shows the expected results: The type of the inner smoother has no effect on the outer convergence behaviour, as long as the inner smoother is strong enough to resolve the local operator anisotropy. This justifies our 'emulation' of a stronger smoother on the GPU by performing eight smoothing steps with the Jacobi. Since in general not all local problems can be resolved with more smoothing steps of the Jacobi, the GPU will also need to be enhanced with more powerful smoothers in the future (cf. Section 4.3).

We finally note that a similar type of anisotropy occurs in fluid dynamics simulations, where it is often necessary to resolve boundary layers more accurately than the inside of the flow domain. The resulting high element aspect ratios typically influence the condition number of the system matrix comparably to anisotropies in the elliptic operators.

6 WEAK SCALABILITY

In this Section, we analyse weak scalability of our GPU-enhanced solver in comparison to the unaccelerated case. The configurations used in these tests comprise two variations of the standard benchmark test case in CSM (see Section 5.3), modified so that each subdomain remains square when doubling the number of subdomains. We increase the number of nodes (and hence, DOFs) from 4, 8, 16, 32 to 64 (32 Mi to 512 Mi, $L = 10$). As we do not have access to enough nodes with modern GPUs, we had to execute the runs on the outdated DQ cluster described in the previous Section.

Figure 7 demonstrates good weak scalability of our approach for both the accelerated and the unaccelerated solver. The relatively poor performance gain of the GPUs is attributed to the outdated GPUs in the DQ cluster, in particular, their small amount of local memory can only hold the data associated with a single subdomain, and consequently, the entire matrix data is paged out from video memory for each subdomain.

As mentioned in Section 3.1, the parallel computation is completely decoupled, data is exchanged asynchronously

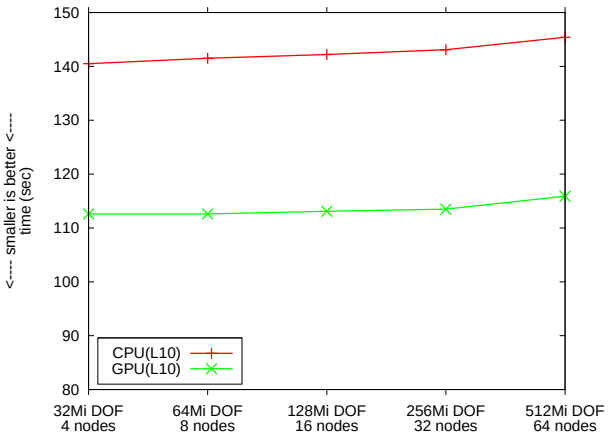

Figure 7: Weak scalability

only between neighbouring subdomains. The only exception is the solution of the global coarse grid problem of the data-parallel multigrid solver, which we perform on the master node while the compute nodes are idle. The size of this coarse grid problem depends only on the number of subdomains (to be more precise, on the number of DOF implied by the coarse grid formed by the collection of subdomains) and is in particular independent of the level of refinement and the number of parallel processes. Due to the robustness of our solver, comparatively few global iterations suffice. In accordance with similar approaches (see, for example, Bergen et al. (2005)), we can safely conclude that the global coarse grid solver, which is the only sequential part of the otherwise parallel execution, is not the bottleneck in terms of weak scalability. As a concrete example, the solution of the global coarse grid problems in the scalability tests in Figure 7 contributes at most 3% to the total runtime.

As previous experiments with the prototypical scalar Poisson equation on up to 160 nodes resulted in equally good scalability (Göddeke et al., 2007b), combining these results we may argue that our heterogeneous solution approach for the more demanding application discussed in this paper would also scale very well beyond 100 GPU nodes.

7 PERFORMANCE STUDIES

We use 16 nodes of an advanced cluster – USC – to compare the performance of the accelerated solver with the unaccelerated case. Table 3 lists the relevant hardware details.

We employ four configurations that are prototypical for practical applications. Figure 8 shows the coarse grids, the prescribed boundary conditions and the partitioning for the parallel execution of each configuration. The `BLOCK` configuration (Figure 8 (a)), as introduced in Section 5.3, is a standard test case in CSM, a block of material is vertically compressed by a surface load. The `PIPE` configuration (Figure 8 (b)) represents a circular cross-section of a pipe clamped in a bench vise. It is realised by loading two opposite parts of the outer boundary by surface forces. With the `CRACK` configuration (Figure 8 (c)) we simulate an industrial test environment for assessing material properties. A workpiece with a slit is torn apart by some device attached to the two holes. In this configuration the deformation is induced by prescribed horizontal displacements at the inner boundary of the holes, while the holes are fixed in the vertical direction. For the latter two configurations we exploit symmetries and consider only sections of the real geometries. Finally, the `STEELFRAME` configuration (Figure 8 (d)) models a section of a steel frame, which is fixed at both ends and asymmetrically loaded from above.

Node	Graphics Card
AMD Opteron Santa Rosa dual-core, 1.8 GHz 2 MiB L2 cache 800 W power supply	NVIDIA Quadro FX5600 600 MHz, max. 171 W
8 GiB DDR2 667 **12.8 GB/s bandwidth**	1.5 GiB GDDR3 **76.8 GB/s bandwidth**
4x DDR InfiniBand 1.6 GB/s peak (0.8–1.2 GB/s benchmarked)	PCIe bus 4 GB/s peak (0.8–1.5 GB/s benchmarked)

Table 3: Hardware configuration of each node in the USC cluster.

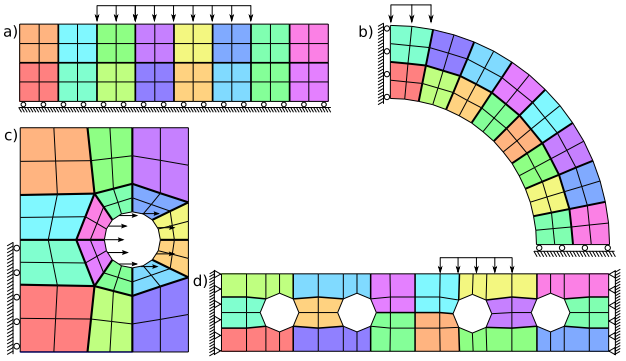

Figure 8: Coarse grids, boundary conditions and static partition into subdomains for the configurations (a) `BLOCK`, (b) `PIPE`, (c) `CRACK` and (d) `STEELFRAME`.

In all tests we configure the solver scheme (cf. Figure 2) to reduce the initial residuals by 6 digits, the global multigrid performs one pre- and postsmoothing step in a V cycle, and the inner multigrid uses a V cycle with four smoothing steps. We consider only the refinement level $L = 10$ (128 Mi DOF), and statically assign four subdomains per node, such as to balance the amount of video memory on the USC cluster with the total amount of memory available per node. The four subdomains per node are either collected in one MPI process (called `single`), leaving the second core of the Opteron CPUs idle, or distributed to two MPI processes per node, each of which comprises two subdomains (called `dual`). As we only have one GPU per node, we perform GPU tests in a `single` configuration. We could also run one large GPU process and one small

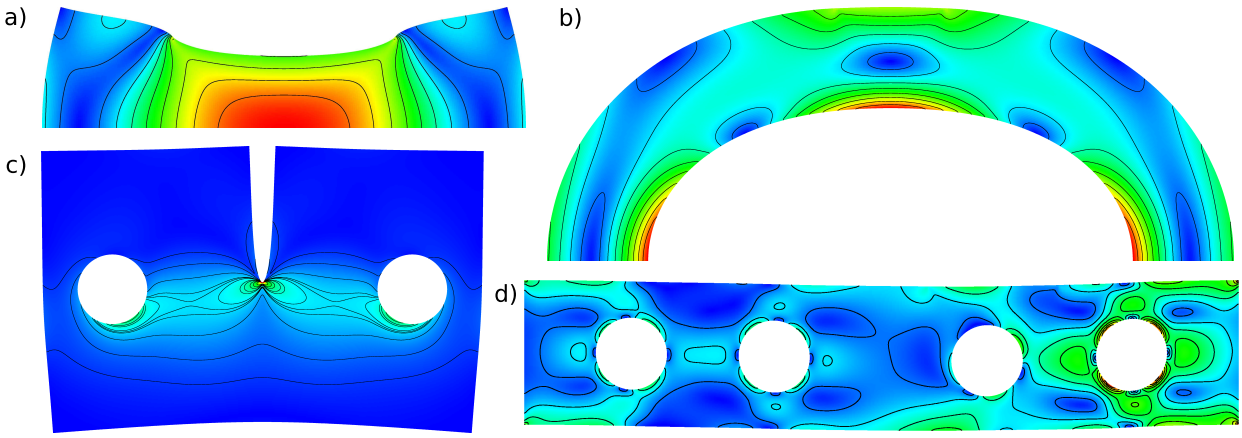

Figure 9: Computed displacements and von Mises stress for the configurations used in the speed-up tests.

CPU process per node in parallel, but previous experiences show that this is only feasible with a more advanced scheduler (Göddeke et al., 2007a). As explained in Section 4.1, we accelerate the plain CPU solver with GPUs by replacing the scalar, local multigrid solvers with their GPU counterparts. Otherwise, the parallel solution scheme remains unchanged. All computations on the CPUs are executed in double precision.

7.1 Absolute Performance

Figure 9 shows the computed deformations of the four geometries and the von Mises stresses, which are an important measure for predicting material failure in an object under load. Both the accelerated and the unaccelerated solver compute identical results, according to a comparison of the displacement of selected reference points and the volume of the deformed bodies.

Figure 10 depicts time measurements for the four configurations. We measure the absolute time to solution for the three scheduling schemes `CPU-single`, `CPU-dual` and `GPU-single`. Consistently for all four configurations, the accelerated solver is roughly 2.6 times faster than the unaccelerated solver using only one MPI process per node, and the speed-up reaches a factor of 1.6 if we schedule two half-sized MPI processes per node. The Opterons in the USC cluster have a very efficient memory subsystem, so that the `dual` configuration runs 1.6 times faster than the `single` configuration. But these absolute timings do not tell the whole story, and favour the CPU in the `CPU-dual` vs. `GPU-single` comparison. We investigate this effect further in the following subsection.

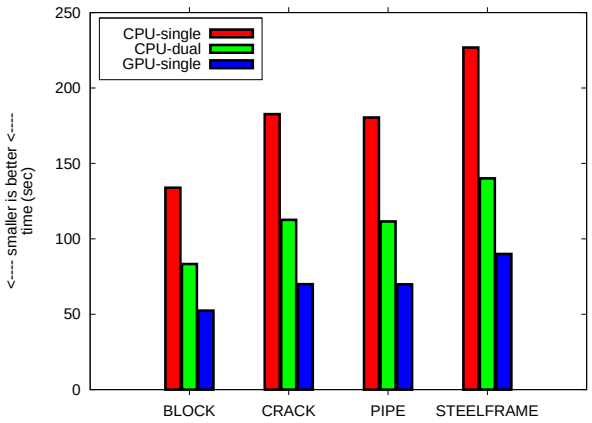

Figure 10: Execution times for the elasticity solver with and without GPU acceleration.

7.2 Performance Analysis

One important benefit of our *minimally invasive* integration of hardware acceleration is that the heterogeneity is encapsulated within the nodes, and the coarse-grained parallelism on the MPI level remains unchanged. Therefore, the correct model to analyse the speed-up achieved by the GPU is *strong scalability* within the node, as the addition of GPUs increases the available compute resources. Consequently, the portions of the application dealing with the coarse-grained parallelism constitute the 'fixed' part of the solution process, as they are not accelerated. In other words, the fraction of the execution time that can be accelerated limits the achievable speed-up.

As the measured acceleration factors are consistent for all four test configurations in the previous section, we perform the analysis only with the `BLOCK` configuration. To separate the accelerable from the unaccelerable portions of the local solution process, we instrumented the code with a special timer T_{local} that measures the local multigrid solvers on each subdomain independently. T_{local} does not include any stalls due to communication, but it does include all data transfers between host and co-processor. Table 4 additionally lists the total time to solution T_{total} and the difference $C := T_{\text{total}} - T_{\text{local}}$, i. e.the time spent in the outer solver, including MPI communication. This data allows us to calculate the fraction $R_{\text{acc}} := T_{\text{local}}/T_{\text{total}}$ of the accelerable part of the code. We see that for the larger subdomains ($L = 9, L = 10$) approximately 66% of the overall time can be accelerated with fine-grained parallelism. Consequently, the estimated maximally achievable

L	T_{total} single	T_{total} dual	T_{total} GPU	T_{local} single	T_{local} dual	T_{local} GPU	$C := T_{\text{total}} - T_{\text{local}}$ single	C dual	C GPU	$R_{\text{acc}} := T_{\text{local}}/T_{\text{total}}$ single	R_{acc} dual	$S_{\text{local}} := T^{\text{CPU}}_{\text{local}}/T^{\text{GPU}}_{\text{local}}$ single	S_{local} dual	S_{total} single	S_{total} dual
8	8.3	5.3	6.8	5.1	3.1	3.5	3.2	2.2	3.3	61%	58%	1.5	0.9	1.3	0.9
9	33.8	20.7	16.9	22.4	13.5	5.5	11.4	7.2	11.4	66%	65%	4.1	2.5	2.0	1.6
10	133.9	83.3	52.4	90.2	55.8	10.0	43.7	27.5	42.4	67%	67%	9.0	5.6	2.5	2.2

Table 4: For subdomains of different size ($L = 8, 9, 10$) and different configurations (`CPU-single`, `CPU-dual`, `GPU`) the table lists the total and local execution time, their difference, the accelerable fraction, and local and total GPU speed-ups.

total speed-up is $1/(1 - 2/3) = 3$.

The comparison of the local CPU and GPU solution time gives us the local GPU speed-up $S_{\text{local}} := T^{\text{CPU}}_{\text{local}}/T^{\text{GPU}}_{\text{local}}$. In the context of Finite Element simulations these numbers are impressive: For large subdomains ($L = 10$) we achieve a GPU speed-up factor 9.0 against a single core and a factor 5.6 against two cores. These factors decrease quickly for smaller subdomains, because of the overhead of co-processor configuration and data transport through the narrow (in comparison to the bandwidth on the co-processor board) PCIe bus.

Given the local GPU speed-up factor S_{local} and the accelerable fraction R_{acc} we can deduce the total GPU speed-up again as

$$S_{\text{total}} := \frac{1}{(1 - R_{\text{acc}}) + (R_{\text{acc}}/S_{\text{local}})}$$

Note the similarity of the formula to Amdahl's Law. Figure 11 illustrates the relation between R_{acc}, S_{local} and S_{total}.

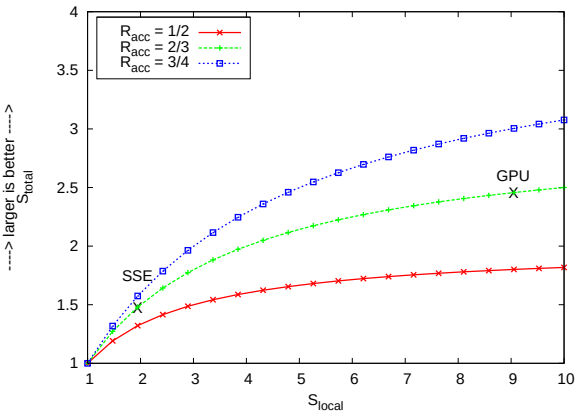

Figure 11: Relation between S_{local} and S_{total} for three different accelerable fractions R_{acc}. The labels `X` mark the estimated speed-up of a mixed-precision SSE implementation on the CPU, and the measured speed-up by our GPU-enhanced solver.

For the application presented in this paper on the highest level of refinement ($L = 10$) with the accelerable fraction $R_{\text{acc}} = 2/3$, we obtain the total GPU speed-ups of 2.5 for single and 2.2 for dual-core (3.0 is the estimated maximum). The first number differs only slightly from the measured factor 2.6 in the previous section, because the unaccelerable portions differ only slightly ($C = 43.7$ to $C = 42.4$). But the estimated total GPU speed-up against the dual-core of 2.2 differs greatly from the measured factor 1.6. We attribute this to the larger difference in the C numbers (27.5 against 42.4): The dual-core version has the unfair advantage of a much faster execution of the global solver. For a fair comparison the GPU solver would have to utilise all the resources within the node instead of leaving one core idle. Consequently, once we design an advanced scheduler and a hybrid communication model that incorporates parallelism among the resources within the node, this problem will no longer be present: All processes will benefit in the same way from interleaved communication and computation.

This analysis allows us to estimate the effectiveness of our approach for other hardware configurations. To illustrate this, we assess the maximum achievable performance gain for a mixed precision solver running entirely on the CPU, by using single precision in all local multigrid solvers, analogously to the GPU implementation. As the performance of the solver is mostly bound by the memory bandwidth and latency (see Section 3.1) we optimistically assume single precision calculations to be performed twice as fast as double precision, at least for a fully SSE-optimised implementation. Substituting the values $R_{\text{acc}} = 2/3$ and $S_{\text{local}} = 2$ in the above formula results in an estimated ideal acceleration of $S_{\text{total}} = 1.5$, which is still far away from the estimated maximum of 3.0. In particular, the gradient of the curve relating S_{local} to the achievable acceleration S_{total} (see Figure 11) is steep at $S_{\text{local}} = 2$, such that further acceleration by GPUs significantly improves the achievable speed-up. The speed-up factors of the GPU for $L = 10$, however, are already very close to the theoretical maximum. Consequently, the gradient of the curve in Figure 11 is small and further local acceleration would give only small returns. Instead, we will have to concentrate on the increase of the accelerable fraction R_{acc}. Figure 11 also illustrates – for three different values of R_{acc} – that the performance difference between the mixed precision CPU approach and GPU acceleration grows rapidly for increasing R_{acc}.

Note, that 2/3 is already a good value when operating within a minimally invasive co-processor integration without any modification of the user code. Since the increase of R_{acc} has a much bigger impact now, we will aim at further improvement in future work. For instance, stronger smoothing operators require more time and consequently improve both numerical robustness and the accelerable fraction R_{acc} at the same time.

8 CONCLUSIONS AND FUTURE WORK

FEAST reduces complex, vector-valued problems to blockwise scalar operations. This allows to optimise data structures, linear algebra operators, multigrid smoothers and even entire local solvers *once* on the intrinsic FEAST kernel level. These components are then used to design robust schemes that perform well out of the box, without much further application-specific performance tuning. The reduction of different problems onto the same global solver scheme also allows a minimally invasive integration of the extension FEASTGPU, which thus adds GPU acceleration to all applications based on FEAST.

Using the unmodified application FEASTSOLID, we have conducted a thorough evaluation of this hardware acceleration approach with respect to accuracy and performance. Because of the hierarchical solver concept, the final accuracy of the results does not suffer from the local computations in single precision on the GPUs, even in case of very ill-conditioned problems. Therefore, despite emerging double precision co-processors, our approach favours the local single precision computation because it offers performance advantages without accuracy limitations. New extensions to FEAST will enable the use of different types of co-processors in the future.

Our performance analysis reveals that the local computations are accelerated by a factor of 9 against a single-core CPU. This acceleration applies to 2/3 of the total solution time and thus translates into a 2.5-fold performance improvement. Future work will seek to increase the accelerable part, so that the local acceleration translates into larger overall gains and the unaccelerable part does not dominate the solution process.

We can conclude that it is possible to significantly accelerate unmodified applications with existing co-processor technology, if the underlying solver concept follows hardware-oriented design principles and the local subproblems for the co-processor are sufficiently large.

ACKNOWLEDGMENT

We thank Christian Becker, Sven Buijssen, Matthias Grajewski and Thomas Rohkämper for co-developing FEAST and its tool chain. Thanks to Michael Köster, Hugh Greenberg and John Patchett for help in debugging and tracking down various software and hardware issues. Also thanks to NVIDIA and AMD for donating hardware that was used in developing the serial version of the GPU backend. Finally, thanks to the anonymous reviewers for many useful suggestions on improving the paper.

This research has been partly supported by a Max Planck Center for Visual Computing and Communication fellowship, by the German Research Foundation (DFG) in Project TU102/22-1 and Research Unit 493 and by the U.S. Department of Energy's Office of Advanced Scientific Computing Research.

REFERENCES

AMD Inc. AMD FireStream stream processor. `http://ati.amd.com/products/streamprocessor/specs.html`, 2008.

Owe Axelsson. On iterative solvers in structural mechanics; separate displacement orderings and mixed variable methods. *Mathematics and Computers in Simulations*, 40:11–30, 1999.

Christian Becker. *Strategien und Methoden zur Ausnutzung der High-Performance-Computing-Ressourcen moderner Rechnerarchitekturen für Finite Element Simulationen und ihre Realisierung in FEAST (Finite Element Analysis & Solution Tools)*. PhD thesis, Universität Dortmund, Logos Verlag, Berlin, 2007.

Benjamin Bergen, Frank Hülsemann, and Ulrich Rüde. Is 1.7×10^{10} unknowns the largest finite element system that can be solved today? In *SC '05: Proceedings of the 2005 ACM/IEEE conference on Supercomputing*, 2005.

Jeff Bolz, Ian Farmer, Eitan Grinspun, and Peter Schröder. Sparse matrix solvers on the GPU: Conjugate gradients and multigrid. *ACM Transactions on Graphics (TOG)*, 22(3):917–924, 2003.

Dietrich Braess. *Finite Elements – Theory, fast solvers and applications in solid mechanics*. Cambridge University Press, 2nd edition, 2001.

ClearSpeed Technology, Inc. ClearSpeed Advance Accelerator Boards. `www.clearspeed.com/products/cs_advance/`, 2006.

Timothy A. Davis. A column pre-ordering strategy for the unsymmetric-pattern multifrontal method. *ACM Transactions on Mathematical Software*, 30(2):165–195, 2004.

Mattan Erez, Jung Ho Ahn, Jayanth Gummaraju, Mendel Rosenblum, and William J. Dally. Executing irregular scientific applications on stream architectures. In *ICS '07: Proceedings of the 21st annual international conference on Supercomputing*, pages 93–104, 2007.

Zhe Fan, Feng Qiu, Arie Kaufman, and Suzanne Yoakum-Stover. GPU cluster for high performance computing. In *SC '04: Proceedings of the 2004 ACM/IEEE conference on Supercomputing*, November 2004.

Kayvon Fatahalian, Timothy Knight, Mike Houston, Mattan Erez, Daniel R. Horn, Larkhoon Leem, Ji Young Park, Manman Ren, Alex Aiken, Willian J. Dally, and Pat Hanrahan. Sequoia: Programming the memory hierarchy. In *SC '06: Proceedings of the 2006 ACM/IEEE conference on Supercomputing*, 2006.

FPGA High Performance Computing Alliance. The FPGA supercomputer "Maxwell". `http://www.fhpca.org/press_energy.html`, 2007.

Genomic Sciences Center, RIKEN. The GRAPE series of processors. http://mdgrape.gsc.riken.jp/, http://www.metrix.co.jp/grapeseries.html, http://www.riken.jp/engn/r-world/info/release/press/2006/060619/index.html, 2006.

Dominik Göddeke, Robert Strzodka, Jamal Mohd-Yusof, Patrick McCormick, Hilmar Wobker, Christian Becker, and Stefan Turek. Using GPUs to improve multigrid solver performance on a cluster. *accepted for publication in the International Journal of Computational Science and Engineering*, Special Issue on Implementational Aspects in Scientific Computing, 2007a.

Dominik Göddeke, Robert Strzodka, Jamaludin Mohd-Yusof, Patrick McCormick, Sven H.M. Buijssen, Matthias Grajewski, and Stefan Turek. Exploring weak scalability for FEM calculations on a GPU-enhanced cluster. *Parallel Computing*, 33(10–11):685–699, 2007b.

Dominik Göddeke, Robert Strzodka, and Stefan Turek. Performance and accuracy of hardware-oriented native-, emulated- and mixed-precision solvers in FEM simulations. *International Journal of Parallel, Emergent and Distributed Systems*, 22(4):221–256, 2007c.

Nolan Goodnight, Cliff Woolley, Gregory Lewin, David Luebke, and Greg Humphreys. A multigrid solver for boundary value problems using programmable graphics hardware. In *Graphics Hardware 2003*, pages 102–111, July 2003.

GPGPU. General-purpose computation using graphics hardware. http://www.gpgpu.org, 2004–2008.

GraphStream, Inc. GraphStream scalable computing platform (SCP). http://www.graphstream.com/, 2006.

Greg Humphreys, Mike Houston, Ren Ng, Randall Frank, Sean Ahern, Peter Kirchner, and Jim Klosowski. Chromium: A stream-processing framework for interactive rendering on clusters. *ACM Transactions on Graphics (TOG)*, 21(3):693–702, July 2002.

Peter D. Kirchner, James T. Klosowski, Peter Hochschild, and Richard Swetz. Scalable visualization using a network-attached video framebuffer. *Computers & Graphics*, 27(5):669–680, October 2003.

Hans Meuer, Erich Strohmaier, Jack J. Dongarra, and Horst D. Simon. Top500 supercomputer sites. http://www.top500.org/, 2007.

Nirnimesh Nirnimesh, Pawan Harish, and P. J. Narayanan. Garuda: A scalable tiled display wall using commodity PCs. *IEEE Transactions on Visualization and Computer Graphics*, 13(5):864–877, September/October 2007.

NVIDIA Corporation. NVIDIA CUDA Compute Unified Device Architecture Programming Guide. http://www.nvidia.com/cuda, January 2007.

NVIDIA Corporation. NVIDIA Tesla GPU computing solutions for HPC. www.nvidia.com/page/hpc.html, 2008.

Evgueni E. Ovtchinnikov and Leonidas S. Xanthis. Iterative subspace correction methods for thin elastic structures and Korn's type inequality in subspaces. *Proceedings: Mathematical, Physical and Engineering Sciences*, 454(1976):2023–2039, 1998.

John D. Owens, David Luebke, Naga Govindaraju, Mark Harris, Jens Krüger, Aaron E. Lefohn, and Timothy J. Purcell. A survey of general-purpose computation on graphics hardware. *Computer Graphics Forum*, 26(1): 80–113, 2007.

John D. Owens, Mike Houston, David Luebke, Simon Green, John E. Stone, and James C. Phillips. GPU computing. *Proceedings of the IEEE*, 96(5):879–899, May 2008.

Stefanie Reese, Martin Küssner, and Batmanathan Dayanand Reddy. A new stabilization technique for finite elements in finite elasticity. *International Journal for Numerical Methods in Engineering*, 44:1617–1652, 1999.

Barry F. Smith, Petter E. Bjørstad, and William D. Gropp. *Domain Decomposition: Parallel Multilevel Methods for Elliptic Partial Differential Equations*. Cambridge University Press, 1996.

Tokyo Institute of Technology. Global scientific information and computing center. http://www.gsic.titech.ac.jp/, 2006.

Stefan Turek. *Efficient Solvers for Incompressible Flow Problems: An Algorithmic and Computational Approach*. Springer, Berlin, 1999.

Stefan Turek, Christian Becker, and Susanne Kilian. Hardware–oriented numerics and concepts for PDE software. *Future Generation Computer Systems*, 22(1-2): 217–238, 2003.

Tom van der Schaaf, Michal Koutek, and Henri Bal. Parallel particle rendering: A performance comparison between Chromium and Aura. In *6th Eurographics Symposium on Parallel Graphics and Visualization*, pages 137–144, May 2006.