GPU Acceleration of an Unmodified Parallel Finite Element Navier–Stokes Solver

Dominik Göddeke, Sven H.M. Buijssen, Hilmar Wobker and Stefan Turek
Angewandte Mathematik und Numerik, TU Dortmund, Germany
dominik.goeddeke, sven.buijssen, hilmar.wobker, stefan.turek@math.tu-dortmund.de

ABSTRACT

We have previously suggested a minimally invasive approach to include hardware accelerators into an existing large-scale parallel finite element PDE solver toolkit, and implemented it into our software FEAST. Our concept has the important advantage that applications built on top of FEAST benefit from the acceleration immediately, without changes to application code. In this paper we explore the limitations of our approach by accelerating a Navier-Stokes solver. This nonlinear saddle point problem is much more involved than our previous tests, and does not exhibit an equally favourable acceleration potential: Not all computational work is concentrated inside the linear solver. Nonetheless, we are able to achieve speedups of more than a factor of two on a small GPU-enhanced cluster. We conclude with a discussion how our concept can be altered to further improve acceleration.

KEYWORDS: Large Scale Scientific Computing, Parallelization of Simulation, Fine-Grain Parallelism and Architectures

1. INTRODUCTION

Computational science and numerical simulation are in the midst of a revolution, caused by the fundamental paradigm shift of the underlying hardware towards parallelism and heterogeneity. Due to power and heat considerations, chip manufacturers now scale the number of cores per chip rather than clock frequencies. At the same time, memory bandwidth and latency continue to improve at a much slower rate than (peak) compute performance, further inhibited by pin limits. This well-known memory wall problem is worsened by multicore architectures: The available bandwidth typically scales with the number of sockets per compute node and not with the number of cores per chip. Graphics processor units (GPUs) on the other hand provide a much higher bandwidth than commodity CPU designs, and their architecture and programming model is representative of future manycore architectures.

1.1. Hardware-Oriented Numerics

The memory wall problem is particularly critical in the numerical simulation of physical phenomena described by partial differential equations (PDEs), such as the Navier–Stokes equations governing fluid flow. In the finite element method (FEM) and similarly for finite differences and finite volumes, the discretisation of the PDEs leads to large, sparse systems of equations, and linear algebra operations such as matrix-vector multiplication exhibit an arithmetic intensity (ratio of floating point operations per memory access) of 1:1 or less, while peak processor performance is only attained for ratios of 10:1 or higher. In addition, the discrete problems are typically much too large to be solved on a single computer, and parallel solution schemes are necessary. We are convinced that in this situation, significant performance gains can only be achieved by 'hardware-oriented numerics'. This concept comprises much more than a highly-tuned implementation involving optimal data structures and maximising data reuse to exploit (cache) memory hierarchies. Here, we only illustrate the broad ideas, and refer to previous work for more details [22, 23].

For ill-conditioned problems depending on the mesh width, multigrid methods are obligatory due to their asymptotic optimality: Even for a simple Poisson problem, a multigrid solver with the worst possible choice of smoother, data structure and numbering scheme for the unknowns executes faster than a Krylov subspace solver with a very powerful elementary preconditioner like ILU [15]. On the other hand, as long as the factorisation overhead can be amortised over several right hand sides, the (serial) direct solver in the UMFPACK library [5] outperforms multigrid for up to 20–60,000 unknowns, depending on the hardware. Serial smoothers with 'optimal' numerical properties are strongly recursive, preventing a scalable implementation in parallel. Instead, methods are employed that are potentially less efficient in terms of numerics (i. e., convergence rates), but that are much better suited for the communication characteristics of the parallel computer. In general, hardware trends enforce research into novel numerical methodology that in turn exploits the available hardware in a better way.

Obviously, the total time to solution is the metric most relevant to the user. However, we believe that extracting eventually 10% more performance for one particular problem on one particular computer system after maybe an entire month of meticulous tuning is of minor importance. The robustness of the numerical discretisation and solution schemes across a wide range of problem classes, problem instances and architectures is in most cases much more relevant. This last aspect explicitly implies that numerical schemes and their implementations should be future-proof for a reasonable timespan, instead of being superseded with the availability of the next hardware generation. The fundamental idea of 'hardware-oriented numerics' is thus to balance all these conflicting goals.

1.2. GPU Computing

Over the past several years, graphics processors (GPUs) have rapidly gained interest as a viable architecture for data-parallel general purpose computations. In fact, GPUs are now widely considered as forerunners of future manycore architectures, comprising many thin cores with limited functionality rather than few fat ones. Peak performance of GPUs exceeds that of CPUs by at least an order of magnitude, a single high-end GPU is capable of delivering more than one TFLOP/s and a sustained bandwidth of 160 GB/s to off-chip memory. This high level of performance is achieved by keeping thousands of threads 'in flight' simultaneously, supported by a hardware scheduler that very efficiently suspends threads stalled for memory transactions. It is important to note that this extremely high memory bandwidth can in fact be achieved in real applications. We refer to a recent article by Fatahalian and Houston for an overview of the throughput-oriented GPU architecture [8].

Despite all advances and achievements, it must be kept in mind that GPUs are co-processors in the traditional sense. Several GPUs within one compute node have to be manually coordinated on the CPU; and for parallel computations on clusters, the CPU retains full control of the interconnect, data has to be manually moved from device memory to host memory prior to be sent over the network, and vice versa.

From a software perspective, programming was cumbersome and often 'hacky' in the early years of GPGPU, as algorithms had to be cast in graphics terms. The advent of NVIDIA's CUDA and AMD's STREAM platforms in late 2006 has resulted in a major surge of interest, culminating in the recent standardisation of OpenCL. The survey articles by Owens et al. [17,18] provide an excellent overview of the field in terms of applications and techniques, and we refer to the community website GPGPU.ORG and the vendors' websites for details and sample applications on the actual implementation of scientific algorithms on graphics hardware.

1.3. Paper Contribution and Overview

We have previously suggested, in the spirit of the general ideas and goals of hardware-oriented numerics outlined above, an approach to include GPUs into a large-scale parallel finite element based PDE toolkit without having to change a single line of application code [10]. The latter benefit is very relevant in practice, in particular in the engineering context where the balancing of implementational effort and prospective performance gains is more important than the best achievable speedup through a full re-implementation of a particular code. We briefly summarise our suggested approach and previous tests for linear problems in Sections 2 and 3.

Many PDE problems in practice are nonlinear. Focussing entirely on accelerating (portions of) linear solvers is still justified, because the approximate solution of linear subproblems often constitutes the most time-consuming task in the solution process, especially for saddle point problems arising, e. g., in CFD. However, this focus naturally limits the achievable speedups, as the system matrices for the linearised problems have to be assembled in each nonlinear sweep. In this paper, we explore the limitations of our approach by applying it to the acceleration of a Navier–Stokes solver for small Reynolds numbers, computing a stationary laminar flow (see Section 4). We refrain from investigating non-stationary flows for two reasons: First, the result in terms of speedups would be qualitatively comparable and the time-stepping loop only increases the runtime of our experiments. Second, the stationary approach significantly worsens the condition number of the matrices, which can be critical as our GPUs only support single precision floating point arithmetics. Hence, by confining ourselves to the stationary case, we actually examine the more difficult configuration. Sections 5 and 6 discuss the results of our evaluation.

1.4. Related Work

Keyes and Colella et al. [4, 13] survey trends towards terascale computing for a wide range of applications, including Finite Element software. They conclude that only a combination of techniques from computer architecture, software engineering, numerical modeling and numerical analysis will enable a satisfactory scale-out on the application level. This is exactly what we refer to as hardware-oriented numerics. Fan et al. were among the first to accelerate a Lattice Boltzmann CFD solver on a GPU cluster [7]. Navier-Stokes simulations on the GPU are abundant, but tend to use Stam's approach which is known to be only visually accurate [20]. Elsen et al. present complex, engineering-level simulations based on the Euler equations on a single GPU [6]. Closest to our work is probably the work by Phillips et al. on accelerating an existing Fortran code-base for solving the Eu-

ler equations on a GPU cluster [19]. Common to all these publications on using GPUs for fluid dynamics simulations is that single precision is sufficient in the respective metric which is used to evaluate the accuracy of the results. In case of the Navier-Stokes equations, however, the elliptic character resulting from the incompressibility condition leads to Poisson-like operators. We have previously demonstrated that double precision is necessary in this case [11].

2. FEAST – HARDWARE-ORIENTED NUMERICS

FEAST (*Finite Element Analysis and Solution Toolkit*) is our next-generation parallel finite element and solver toolkit for the solution of PDE problems on HPC systems, actively developed at TU Dortmund. It implements many concepts of hardware-oriented numerics to maximise single-processor performance, and to achieve near-perfect weak and strong scalability. In this section, we briefly summarise the most important ones in the scope of this paper. For more information on FEAST, we refer to previous publications [22, 23].

2.1. Globally Unstructured, Locally Structured Grids

FEAST covers the computational domain with a collection of quadrilateral subdomains. These patches form an unstructured coarse mesh and each subdomain is refined in a regular, generalised tensor product fashion. The unstructured coarse mesh retains flexibility in resolving geometric details, while the tensor product property of the local meshes entails a linewise numbering of the unknowns which is exploited in optimised linear algebra components. The guiding idea of this separation between structured and unstructured data is that matrices stemming from the FE discretisation on fully adaptive, unstructured grids typically lead to a significant performance penalty for linear algebra components such as sparse matrix vector multiply, see Köster et al. [15] for illustrative examples in the scope of this paper. On the other hand, the linewise numbering in FEAST leads to banded matrices which permits matrix-vector multiplication and other multigrid solver components such as the application of smoothers to be implemented with direct memory accesses only, significantly reducing the bandwidth requirements and increasing performance: In view of the memory wall problem, this approach allows for implementations exhibiting very good spatial and temporal locality, executing close to peak bandwidth and throughput with near-optimal cache hit rates [2]. Anisotropic refinement within each subdomain, hanging nodes on subdomain boundaries and r-adaptivity via mesh deformation can be realised without affecting the tensor product property.

2.2. Parallel Multilevel Solvers

SCARC (*Scalable Recursive Clustering*), the solver concept at the core of FEAST, generalises techniques from *multilevel domain decomposition* and *parallel multigrid*; combining their respective advantages into a very robust, and (numerically *and* computationally) efficient parallel solution scheme for (scalar) elliptic PDEs. Matrices and vectors exist only locally, based on minimally overlapping subdomains, which means that, essentially, only data associated with vertices lying on subdomain boundaries have to be shared via communication [14]. Between the different mesh resolutions, the coupling is multiplicative, as in classical multigrid. Within one hierarchy level, the coupling is additive, i. e. the minimally overlapping subdomains are treated simultaneously and independently of each other. Global coarse grid problems are solved with UMFPACK [5] on a master node.

Instead of the blockwise application of elementary local smoothers, the SCARC scheme employs full multigrid solvers acting locally on the individual subdomains to 'hide' local irregularities as much as possible from the outer solver. The local solvers are typically configured to, e. g., gain one digit, and can fully exploit the underlying tensor product property of the local problems. SCARC can be interpreted as a 'generalised domain decomposition multigrid method'; we use the terminology of *two-layer* SCARC solvers.

The resulting hierarchical solvers are very robust, exhibiting very good weak and strong scaling. In previously published work, we demonstrated—for the maximum number of resources available to us at that time—perfect weak scalability for the Poisson problem on up to 320 Xeon processors [9], and excellent strong scaling for applications from linearised elasticity and incompressible flow for an experiment that subsequently quadrupled the resources up to a maximum of 128 CPUs [23].

2.3. Scalar and Vector-Valued Problems

The original implementation of SCARC supports scalar systems only. Vector-valued systems, such as the Navier–Stokes equations, are treated so that as much of the existing linear algebra and solver infrastructure can be reused, in particular optimisations for various architectures. To achieve this, the unknowns are numbered equation-wise, leading to block-structured matrices. In this paper, blocks correspond to the velocities in x and y direction, and to pressure. All operations on the block system, from linear algebra to entire multigrid solvers, are implemented as sequences of scalar operations corresponding to the blocks. Block systems can be restricted to individual subdomains, resulting in a generalised SCARC solver for vector-valued problems. We illustrate this approach in more detail in Section 4.2.

3. MINIMALLY INVASIVE GPU INTEGRATION

We previously suggested an approach of a *minimally invasive* co-processor integration, tailored to the solution of PDE problems on heterogeneous commodity based HPC clusters. GPUs are particularly well suited hardware accelerators for FE codes, because they provide an order of magnitude more bandwidth to off-chip memory. In addition, this bandwidth increases at a much faster rate than in commodity CPU designs, a consequence of the latency-hiding massively multithreaded hardware architecture. In view of our concept of hardware-oriented numerics, GPUs are thus a viable target architecture, the gains in performance and their future prospects outweigh the initial effort of re-implementing parts of existing codes.

The core idea of our approach is to replace *only* the local multigrid solvers within a two-layer ScaRC scheme by GPU-accelerated counterparts. This means that there is a considerable amount of arithmetic work on the device, which is important as GPUs communicate with the rest of the node (CPU, main memory, interconnect) over a bandwidth bottleneck. This approach is not limited to GPUs alone, different backends, e. g., for the Cell processor, can be added with minimal effort as soon as the implementation of the multigrid solver on the co-processor is available. Our current implementation only supports GPUs though. In particular, applications built on top of FEAST can benefit from the GPU acceleration of the underlying solvers without any changes to application code, speedups are enabled by changing a target parameter inside some configuration file. As these devices execute much faster in single than in double precision, or only support single precision such as the GPUs we use for this paper, the entire scheme constitutes a mixed precision solver.

The concept has been thoroughly evaluated in a series of three journal articles. The first one presents the concept, and evaluates it in various performance and energy related metrics [10]. The second paper confirms that the excellent scalability of the CPU-based ScaRC solvers is retained on a 160 node hybrid CPU-GPU cluster [9]. Both publications are restricted to the Poisson equation as an important model problem. The last paper evaluates the concept for more challenging simulations in linearised elasticity [12]. In all tests so far, we always achieve the same accuracy in the results, despite the restriction of the local solvers to single precision.

However, all these experiments are limited to linear problems. In this paper we extend our evaluation to the Navier–Stokes equations, a nonlinear saddle point problem which requires a more complicated solution scheme exhibiting a smaller acceleration potential.

It is also worth noting that we migrated from an OpenGL-based implementation of the GPU code, used in the previous publications, to NVIDIA CUDA. This had two benefits: First, performance increased by roughly 10% because of using hardware features (in particular, the parallel data cache) not exposed to the graphics API. Second, the CUDA backend of our co-processor extension to FEAST is roughly half as complex (measured in lines of code) as the OpenGL version, an important benefit in terms of debugging and maintainability.

4. NAVIER–STOKES SOLVERS IN FEAST

The *Navier–Stokes* equations are derived under certain assumptions and simplifications from the general conservation laws for mass, momentum and energy in continuum mechanics. They govern the flow of incompressible, isothermal, homogeneous and isotropic Newtonian fluids and gases. The *kinematic viscosity* ν, the ratio between inertial and viscous forces, is assumed to be constant. It is inversely proportional to the *Reynolds number* of the fluid; more precisely, the relation is $Re = UL/\nu$ where U is the characteristic fluid velocity and L the characteristic length. Low Reynolds numbers correspond to highly viscous fluids, small spatial dimensions or low flow speeds.

The stationary Navier–Stokes equations read in their dimensionless form:

$$\begin{aligned} -\frac{1}{Re}\Delta \boldsymbol{u} + (\boldsymbol{u}\cdot\mathbf{grad})\boldsymbol{u} + \mathbf{grad}\, p &= \boldsymbol{f} \text{ in } \Omega \\ \operatorname{div} \boldsymbol{u} &= 0 \text{ in } \Omega \\ \boldsymbol{u} &= \boldsymbol{u}_D \text{ on } \Gamma_{\mathrm{D}} \\ -\frac{1}{Re}(\boldsymbol{n}\cdot\mathbf{grad})\boldsymbol{u} + p\boldsymbol{n} &= \boldsymbol{t} \text{ on } \Gamma_{\mathrm{N}} \end{aligned} \tag{1}$$

Here, $\boldsymbol{u}$ denotes velocity, p pressure, $\boldsymbol{f}$ body forces, $\Omega \subseteq \mathbb{R}^2$ the domain with outer normal vector $\boldsymbol{n}$, and Γ_{D} and Γ_{N} the boundary parts with, respectively, Dirichlet and natural boundary conditions (i. e., inflow, outflow and adhesion/slip conditions). A common simplification are the *Stokes* equations for very low Reynolds numbers. In this case, the viscous term dominates and the nonlinear transport term $(\boldsymbol{u}\cdot\mathbf{grad})\boldsymbol{u}$ can be omitted, leading to a linear problem.

4.1. Discretisation

We consider the weak formulation of equation system (1) and apply a Finite Element discretisation with conforming bilinear elements of the Q_1 space for both $\boldsymbol{u}$ and p. This Galerkin approach exhibits instabilities which stem from dominating convection and from the violation of the discrete inf-sup or LBB-condition [3]. For stability on arbitrary meshes, pressure-stabilisation (PSPG) and streamline-upwind stabilisation (SUPG) is applied choosing the mesh-dependent pa-

rameters in accordance with Apel et al. [1]. The resulting discrete nonlinear equation system reads

$$\begin{pmatrix} \mathbf{A}_{11} & \mathbf{A}_{12} & \mathbf{B}_1 \\ \mathbf{A}_{21} & \mathbf{A}_{22} & \mathbf{B}_2 \\ \mathbf{B}_1^\mathsf{T} & \mathbf{B}_2^\mathsf{T} & \mathbf{C} \end{pmatrix} \begin{pmatrix} \mathbf{u}_1 \\ \mathbf{u}_2 \\ \mathbf{p} \end{pmatrix} = \begin{pmatrix} \mathbf{f}_1 \\ \mathbf{f}_2 \\ \mathbf{g} \end{pmatrix},$$

with

$$\begin{aligned} \mathbf{A}_{11} &:= \frac{1}{Re}\mathbf{L}_{11} + \mathbf{N}_{11}(\mathbf{u}) + \tilde{\mathbf{C}}_{11} \\ \mathbf{A}_{12} &:= \tilde{\mathbf{C}}_{12} \\ \mathbf{A}_{21} &:= \tilde{\mathbf{C}}_{21} \\ \mathbf{A}_{22} &:= \frac{1}{Re}\mathbf{L}_{22} + \mathbf{N}_{22}(\mathbf{u}) + \tilde{\mathbf{C}}_{22}, \end{aligned}$$

in short $\mathbf{K}\mathbf{v} = \mathbf{h}$ where the matrices $\mathbf{L}_{ii}$ correspond to the Laplacian operator and $\mathbf{N}_{ii}(\mathbf{u})$ to the convection operator. $\mathbf{B}$ and $\mathbf{B}^\mathsf{T}$ are discrete analogues of the gradient and divergence operators while $\mathbf{C}$ and $\tilde{\mathbf{C}}_{ij}$ stem from the discretisation of the PSPG and SUPG stabilisation terms, respectively. For the case of an isotropic mesh, we notice the following: $\mathbf{C}$ is identical to a discretisation of the pressure Poisson operator, scaled with the mesh size h^2.

4.2. Solution Algorithm

The nonlinear problem is reduced to a sequence of linear problems by applying a fixed point defect correction method

$$\mathbf{v}^{k+1} = \mathbf{v}^k + \omega\tilde{\mathbf{K}}_{\text{B}}^{-1}(\mathbf{h} - \mathbf{K}\mathbf{v}^k) \qquad k = 1, \dots$$

where the application of $\tilde{\mathbf{K}}_{\text{B}}$ can be identified with the solution of linearised subproblems with the nonlinear residual as right hand side and using the solution $\mathbf{u}$ from the previous nonlinear iteration for assembly of the $\mathbf{N}_{ii}(\mathbf{u})$ part of $\mathbf{K}$. These still vector-valued linearised subproblems are subsequently tackled with help of a pressure Schur complement approach. We illustrate it with the following basic iteration, but prefer a Krylov subspace solver such as BiCGStab for increased numerical efficiency:

$$\begin{pmatrix} \mathbf{u}^{n+1} \\ \mathbf{p}^{n+1} \end{pmatrix} = \begin{pmatrix} \mathbf{u}^{n} \\ \mathbf{p}^{n} \end{pmatrix} + \mathbf{K}_{\text{S}}^{-1} \left[\begin{pmatrix} \mathbf{f} \\ \mathbf{g} \end{pmatrix} - \begin{pmatrix} \mathbf{A} & \mathbf{B} \\ \mathbf{B}^\mathsf{T} & \mathbf{C} \end{pmatrix} \begin{pmatrix} \mathbf{u}^{n} \\ \mathbf{p}^{n} \end{pmatrix} \right] \tag{2}$$

$\mathbf{A}$ is a block-structured matrix consisting of the linearised matrices $\mathbf{A}_{ij}$, $\mathbf{B}^\mathsf{T}$ is defined as $(\mathbf{B}_1^\mathsf{T}, \mathbf{B}_2^\mathsf{T})$ and the vectors $\mathbf{u}^n$ and $\mathbf{f}$ as the iterates of the solution $(\mathbf{u}_1, \mathbf{u}_2)^\mathsf{T}$ and right hand side $(\mathbf{f}_1, \mathbf{f}_2)^\mathsf{T}$, respectively. The preconditioner $\mathbf{K}_{\text{S}}$ is defined as the lower block triangular matrix:

$$\mathbf{K}_{\text{S}} := \begin{pmatrix} \mathbf{A} & 0 \\ \mathbf{B}^\mathsf{T} & -\mathbf{S} \end{pmatrix}$$

involving the pressure Schur complement matrix

$$\mathbf{S} := \mathbf{B}^\mathsf{T}\mathbf{A}^{-1}\mathbf{B} - \mathbf{C}.$$

It is easy to prove that the square of the iteration matrix of the preconditioned system

$$\hat{\mathbf{K}} := \mathbf{I} - \mathbf{K}_{\text{S}}^{-1} \begin{pmatrix} \mathbf{A} & \mathbf{B} \\ \mathbf{B}^\mathsf{T} & \mathbf{C} \end{pmatrix}$$

vanishes [16], which is equivalent to the associated Krylov space, $\text{span}\{\mathbf{r}, \hat{\mathbf{K}}\mathbf{r}, \hat{\mathbf{K}}^2\mathbf{r}, \hat{\mathbf{K}}^3\mathbf{r}, \dots\}$, having dimension 2. This implies that—with exact arithmetics—any Krylov subspace solver would terminate in at most two iterations with the solution to the linear system arising in system (2), using the the preconditioner $\mathbf{K}_{\text{S}}$. Few iterations with a 'good' approximate $\tilde{\mathbf{K}}_{\text{S}}$ of $\mathbf{K}_{\text{S}}$ hence suffice to solve system (2).

In summary, the basic iteration (2) entails the following steps:

1. Compute the global defect

$$(\mathbf{d}_1, \mathbf{d}_2, \mathbf{d}_3)^\mathsf{T} = \begin{pmatrix} \mathbf{f} \\ \mathbf{g} \end{pmatrix} - \begin{pmatrix} \mathbf{A} & \mathbf{B} \\ \mathbf{B}^\mathsf{T} & \mathbf{C} \end{pmatrix} \begin{pmatrix} \mathbf{u}_n \\ \mathbf{p}_n \end{pmatrix}$$

2. Apply the block preconditioner $\tilde{\mathbf{K}}_{\text{S}}$ by approximately solving

$$\mathbf{A}(\mathbf{c}_1, \mathbf{c}_2)^\mathsf{T} = (\mathbf{d}_1, \mathbf{d}_2)^\mathsf{T} \tag{3}$$

and

$$\mathbf{S}\mathbf{c}_3 = -\mathbf{d}_3 + \mathbf{B}_1^\mathsf{T}\mathbf{c}_1 + \mathbf{B}_2^\mathsf{T}\mathbf{c}_2 \tag{4}$$

3. Update the global solution with the (damped) correction vector:

$$(\mathbf{u}^{n+1}, \mathbf{p}^{n+1})^\mathsf{T} = (\mathbf{u}^n, \mathbf{p}^n)^\mathsf{T} + \omega(\mathbf{c}_1, \mathbf{c}_2, \mathbf{c}_3)^\mathsf{T}$$

The second step is the most expensive one and requires closer examination. Note that the diagonal block matrices $\mathbf{A}_{ii}$ correspond to scalar operators due to the momentum equations, so FEAST's tuned scalar solvers can be applied. Our solution scheme is a two-layer SCARC solver, generalised to treat the vector-valued block system (3). The outer layer comprises a global multigrid solver configured to perform a V cycle with 1 pre- and no postsmoothing step (more smoothing only results in longer total runtimes). As UMFPACK outperforms multigrid methods for small problem sizes, the multigrid stops traversing the grid hierarchy at the first refinement level with less than 20,000 unknowns. The global multigrid solver is additively smoothed by local multigrid issued on every subdomain and run either on the CPU or GPU. This inner layer performs a V cycle with 4 pre- and postsmoothing steps using a Jacobi smoother. Its coarse grid solver is a conjugate gradient method, as a direct solver has not been implementated yet on the GPU. The number of unknowns for which this multigrid is truncated in favour of the coarse grid solver is 200. Both global and local multigrid stop as soon as the respective initial residual is reduced by one digit.

To treat the $\mathbf{S}$ block efficiently, an appropriate preconditioner for the pressure Schur complement is required. It has been

fixed point iteration
solving linearised subproblems with
global BiCGStab (reduce initial residual by 1 digit)
preconditioned by applying $\tilde{\mathbf{K}}_S$ via
1) approx. solve $\mathbf{A}(\mathbf{c}_1, \mathbf{c}_2)^\mathsf{T} = (\mathbf{d}_1, \mathbf{d}_2)^\mathsf{T}$ with
global MG (V 1+0), additively smoothed by
for all $\bar{\Omega}^{(i)}$: solve $\mathbf{A}_{11}^{(i)}\mathbf{c}_1^{(i)} = \mathbf{d}_1^{(i)}$ with
local MG (V 4+4, Jacobi, CG)
for all $\bar{\Omega}^{(i)}$: solve $\mathbf{A}_{22}^{(i)}\mathbf{c}_2^{(i)} = \mathbf{d}_2^{(i)}$ with
local MG (V 4+4, Jacobi, CG)
2) update RHS: $\mathbf{d}_3 = -\mathbf{d}_3 + \mathbf{B}^\mathsf{T}(\mathbf{c}_1, \mathbf{c}_2)^\mathsf{T}$
3) scale $\mathbf{c}_3 = (\mathbf{M}_p^\mathsf{L})^{-1}\mathbf{d}_3$

Figure 1: Our Navier–Stokes Solution Scheme

shown that the pressure mass matrix $\mathbf{M}_p$ is a good preconditioner for the diffusive part of $\mathbf{S}$ [21]. The use of a lumped mass matrix $\mathbf{M}_p^\mathsf{L}$ reduces solving equation (4) to scaling the right hand side with the inverse of this diagonal matrix. As the convection part is neglected it is clear that this choice of the preconditioner is only favourable for stationary Navier–Stokes problems at low Reynolds numbers. Figure 1 summarises the solver scheme, the accelerable parts of the algorithm are highlighted.

5. RESULTS

5.1. Test Procedure

The addition of GPUs to an existing cluster increases the compute resources. Therefore, the correct model to analyse performance is strong scalability within each node. In other words, the time spent in portions of the entire scheme that cannot be accelerated limits the achievable speedup. To quantify this effect, we instrument the code with timers measuring only the local multigrid solvers, as this is the only part of the solver that is accelerated by the GPUs so far (see Figure 1). This allows us to distinguish between the total speedup and the local speedup achieved by the GPUs alone. We execute all tests on our small four-node GPU-enhanced cluster. Table 1 summarises the hardware details of a single node, note that the GPUs stem from the same hardware generation as the remaining cluster components. This small amount of compute nodes unfortunately prevents performing meaningful scalability tests. We know from previous work that FEAST scales very well, and we are confident that these results transfer to our Navier–Stokes solver [9, 12].
Our primary test problem (see Figure 2) is a driven cavity simulation on a unit-square domain—a standard benchmark for CFD codes. The domain is partitioned into eight subdomains, each of which is refined regularly L times, leading to global problem sizes of 395 523, 1 577 475, 6 300 675 and 25 184 259 degrees of freedom for $L = 7, \ldots, 10$. The corresponding local, scalar sub-problems comprise 16 641, 66 049, 263 169 and 1 050 625 degrees of freedom. The global problem sizes are chosen to fill the available memory of the four nodes, and we statically schedule two subdomains per node.

Table 1: Hardware Details of our 4-Node GPU Cluster

CPU	AMD Opteron X2 Santa Rosa
	1.8 GHz, 1 MB L2 cache per core
CPU memory	8 GB DDR2 667
	6.4 GB/s bandwidth
GPU	NVIDIA GeForce 8800 GTX
	16 multiprocessors (128 'cores'), 1.35 GHz
GPU memory	768 MB GDDR3
	86.4 GB/s bandwidth
Interconnect	DDR Infiniband

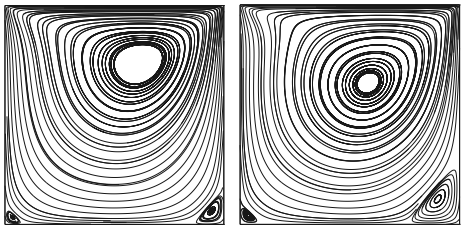

Figure 2: Driven Cavity Simulation at Re=100 and 250

5.2. Analysis of the Linear Solver

Our first experiment focusses on the linear solver alone, so we consider the (linear) Stokes problem instead of performing a full Navier–Stokes simulation. We equip the code with timers as explained above, and measure the following quantities of interest:

- $T_{\text{lin}}^{\text{C,G}}$ – time to solve the linear problem, superscripts C and G denote the original CPU and the GPU-accelerated solver respectively
- $T_{\text{loc}}^{\text{C,G}}$ – time spent in the local scalar multigrid solvers
- $R_{\text{acc}} = T_{\text{local}}^{\text{C}}/T_{\text{lin}}^{\text{C}}$ – accelerable fraction of the linear solver
- $S_{\text{max}} = 1/(1 - R_{\text{acc}})$ – maximal theoretical speedup
- $S_{\text{loc}} = T_{\text{loc}}^{\text{C}}/T_{\text{loc}}^{\text{G}}$ – local speedup by the GPU
- $S_{\text{tot}} = T_{\text{lin}}^{\text{C}}/T_{\text{lin}}^{\text{G}}$ – total speedup of the entire parallel linear solver

Table 2 (top) presents the results, using one core of each CPU. Several important observations can be made: For the two small refinement levels, there is simply not enough local work to achieve good local and global speedups. This can also be seen by comparing the rise of $T_{\text{loc}}^{\text{C}}$ with $T_{\text{loc}}^{\text{G}}$ for increasing level of refinement. On the finest level, the GPUs accelerate the solution of the local problems by a factor of more than 12. However, the acceleration potential of the

solver is (excluding $L = 7$) only approximately 60%, limiting the achievable speedup to 2.3–2.6 (cf. Amdahl's Law). The measured total speedup is very close to this predicted maximum. Another way to interpret this result is that further tuning of the GPU solver will not lead to significant overall performance improvements, as (again excluding the smallest test case) the increase in local speedup does not translate into a corresponding increase of the total speedup.

Table 2: Performance of the Stokes Solver. Top: one CPU Core vs. one GPU, bottom: one CPU Core vs. one GPU.

L	$T_{\text{lin}}^{\text{C}}$	$T_{\text{loc}}^{\text{C}}$	$T_{\text{lin}}^{\text{G}}$	$T_{\text{loc}}^{\text{G}}$	R_{acc}	S_{max}	S_{loc}	S_{tot}
7	5.1	2.3	3.9	1.1	0.45	1.82	2.30	1.31
8	18.6	10.7	9.9	2.2	0.58	2.35	4.86	1.88
9	69.4	42.6	31.5	5.0	0.61	2.59	8.52	2.20
10	327.6	201.8	145.5	16.3	0.62	2.60	12.38	2.25

L	$T_{\text{lin}}^{\text{C}}$	$T_{\text{lin}}^{\text{G}}$	$S_{\text{single- vs. dualcore}}$	S_{tot}
7	3.4	3.9	1.50	0.87
8	11.9	9.9	1.56	1.20
9	45.3	31.5	1.53	1.44
10	202.8	145.5	1.62	1.39

An ideal GPU cluster includes one GPU per CPU core to maximise performance improvements, different from our test setup. Nonetheless, we perform the same experiment again, but this time we schedule the two subdomains per node not to one core, but to both cores. As the two cores in the Santa Rosa architecture share the memory controller and have disjunct caches, we do not expect ideal strong scaling. Table 2 (bottom) summarises the timing measurements and derived quantities, the GPU timings are repeated from the previous table. On average, using two cores is 1.5–1.6 times faster than using only one core, and consequently, the total speedup obtained by the GPU reduces to approximately 1.4. For small local problem sizes, two cores even outperform a single GPU. As we can only use two subdomains per node due to memory limitations, we cannot test the ideal hybrid configuration of one CPU core executing the solver on one subdomain, and one CPU core managing the GPU which treats, e. g., two or three subdomains. We have tested this configuration successfully for the Poisson problem in a previous publication [10].

5.3. Analysis of the Navier–Stokes Solver

For the stationary Navier–Stokes simulation, we are mainly interested in how the various fractions of the total runtime of the individual solver components change by GPU acceleration. We perform two test series, one for Reynolds number 100 and one for 250. Note that the problem is harder to solve the higher the Reynolds number gets since the pressure mass matrix becomes a less feasible preconditioner. The outer BiCGStab method requires more iterations, leading in turn to an increase in the total number of global multigrid iterations. For our discussion, we need the following (derived) quantities:

- R_{acc} – accelerable fraction
- R_{ass} – fraction spent in matrix assembly
- R_{lin} – fraction spent in linear solver
- S_{tot} – total speedup

We first confirm that in all cases, both the unaccelerated and the accelerated solver compute exactly the same result, by measuring the kinematic energy $\frac{1}{2}\int_\Omega ||\mathbf{u}||^2 d\Omega$, a grid independent quantity regularly compared for driven cavity tests [24]. This endorses our previous results [12], that even for very ill-conditioned problems, the restriction of the innermost linear solver to single precision does not have any negative side effect. The two solvers do not converge in exactly the same way, occasionally the linear solves are slightly cheaper on the CPU, and on a different refinement level, the local multigrid solvers on the GPU converge minimally better. This is actually expected behaviour, due to the huge amount of floating point operations and the ill-conditionness of the problem: The descent directions of the outer Krylov space solver differ slightly in the event of minor variations of the global multigrid preconditioner. In the course of the nonlinear loop, these differences build up, but are always within 5%, and we do not normalise our timings to hide the small differences. All our performance indicators as listed above are computed directly from the wall clock runtime of the simulation.

Table 3: Stationary Navier–Stokes Solver Timings

Re	L	$R_{\text{ass}}^{\text{C}}$	$R_{\text{lin}}^{\text{C}}$	$R_{\text{ass}}^{\text{G}}$	$R_{\text{lin}}^{\text{G}}$	R_{acc}	S_{tot}	S_{max}
100	7	0.31	0.67	0.34	0.63	0.31	1.09	1.45
	8	0.33	0.65	0.44	0.54	0.39	1.33	1.63
	9	0.33	0.66	0.46	0.52	0.42	1.41	1.72
	10	0.28	0.71	0.50	0.48	0.46	1.74	1.86
250	7	0.12	0.87	0.12	0.87	0.43	1.05	1.76
	8	0.12	0.87	0.18	0.81	0.53	1.45	2.12
	9	0.12	0.88	0.22	0.77	0.56	1.86	2.29
	10	0.11	0.89	0.24	0.75	0.58	2.09	2.38

Table 3 presents the results we obtain. As we use the same linear solver as for the Stokes tests, the local acceleration factor is identical and we do not repeat the exact numbers here. Looking at the $Re = 100$ testcase first, we observe global speedup factors up to 1.7, which is very close to the achievable maximum of 1.85 because at most 46% of the entire solver can be accelerated. For the CPU variant, the linear solver dominates the runtime. Note that the fractions do not necessarily add up to 100 because we exclude the few percent needed by the nonlinear defect correction loop. This is not the case for the accelerated solver and the highest level of refinement; here, assembly and solving take approximately the

same time. For the more involved higher Reynolds number, the speedup is roughly 2 for the largest test instance, a consequence of the better acceleration potential R_{acc} due to the linear problem being harder to solve (see above). In this case, the linear solving still constitutes the most time-consuming substep of the entire simulation, despite the acceleration.

5.4. Channel Flow Around a Cyclinder

Our final test is based on the 1996 DFG benchmark configuration *Benchmark computations of laminar flow around a cylinder* [25]. Figure 3 shows the domain, the unstructured coarse grid and the computed pressure and velocity. As our Schur complement approach is designed for non-stationary flows and is not the optimal choice for the computation of steady flows [21], we reduce the Reynolds number Re from 1000 to 100, but otherwise, we solve exactly the same configuration as described in the benchmark. The coarse grid comprises 24 subdomains, so we cannot perform the simulation for refinement level 10 due to memory constraints. The largest problem size is 18 891 264 degrees of freedom.

Table 4: Performance of the Channel Flow Benchmark

L	$R^{\text{C}}_{\text{ass}}$	$R^{\text{C}}_{\text{lin}}$	$R^{\text{G}}_{\text{ass}}$	$R^{\text{G}}_{\text{lin}}$	R_{acc}	S_{loc}	S_{tot}	S_{max}
7	0.04	0.96	0.05	0.95	0.59	1.20	1.21	2.43
8	0.08	0.92	0.11	0.89	0.62	2.14	1.37	2.60
9	0.13	0.86	0.25	0.74	0.60	5.97	1.88	2.50

Again, the original and the accelerated solver compute the exactly same result in a slightly different way. In detail, for $L = 7$ the CPU requires 12% more linear iterations, and for refinement levels 8 and 9, the GPU needs 13% and 6% more iterations, respectively. Table 4 summarises the relevant derived quantities from our timing measurements. The results confirm the general speedup trends established in all our experiments: The local speedup is—across all refinement levels—in the same range as measured before, reaching a factor of 6 for $L = 9$. Approximately 60% of the entire simulation can be accelerated, and we achieve a total speedup factor of 1.88 for the largest configuration. This is again close to the theoretical maximum, at least when factoring out the 6% difference in iterations.

6. CONCLUSION

Our experiments are designed to demonstrate the feasibility of our minimally invasive GPU acceleration in the case of the Navier-Stokes equations, but also to explore its limitations. The local speedup of the accelerable parts of the solution process is excellent, reaching 12 and higher. However, if less than half of the total time is spent in these parts, the acceleration of the entire scheme is limited (Amdahl's Law).

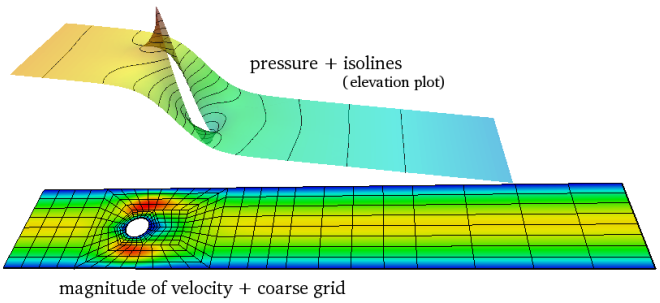

Figure 3: Channel Flow Around a Cylinder

One idea to shift the ratio in favour of the accelerable portion of the linear solver is to replace the nonlinear defect correction loop with a Newton–Raphson method. This would lead to less nonlinear steps, equivalent to reducing the fraction of non-accelerated assembly time, while typically, a more accurate solution of the linear subproblems would be required. For stationary problems this is clearly worth pursueing in future. For large-scale time-dependent complex simulations, though, it is in general challenging to predict whether it pays off to invoke—in each time step—such a method instead of the computationally cheaper defect correction loop. Either way, the actual runtime for time-dependent simulations is measured in hours and days, and a speedup of factor two with the presented approach against a highly tuned CPU code is indeed relevant.

An alternative avenue for future work is to relax our minimally invasive approach by shifting more functionality onto the GPU. We are convinced that matrix assembly can be efficiently implemented for GPUs, the challenge lies in rearranging memory accesses and memory writes for optimal locality and coherency. The second important consideration is that matrices must be assembled in double precision, as the elliptic character of the Poisson-like operators dominates. GPUs have only recently begun to support double precision, and peak performance reaches less than 10% of single precision performance.

Our experiments also reveal that, eventually, more than the local multigrid solvers within the linear solvers should be moved to the GPU. This is even more challenging, because it violates the locality principle: Only a very small amount of work (e. g., one matrix-vector multiplication) is performed on the device before the data has to be moved through two bandwidth bottlenecks: Data is first copied via the PCIe bus from device memory to host memory, again copied from host memory into some MPI buffer, sent over the network, and copied again twice until it reaches the destination device memory. It is currently not possible to use DMA transfers to move data directly from device memory to interconnect buffers, bypassing the CPU. Future hardware genera-

tions might lift this restriction. For the same reason as before, double precision is needed for this step.

ACKNOWLEDGEMENTS

The authors would like to thank Christian Becker and the co-developers of FEAST. The 'minimally invasive' concept of co-processor acceleration has been developed in close collaboration with Robert Strzodka, Jamaludin Mohd-Yusof and Patrick McCormick. This work has in parts been supported by Deutsche Forschungsgemeinschaft (DFG), projects TU102/22-1, TU102/22-2, TU102/27-1 and TU102/11-3. Thanks to NVIDIA for donating hardware.

REFERENCES

[1] T. Apel, T. Knopp, and G. Lube. "Stabilized finite element methods with anisotropic mesh refinement for the Oseen problem", *Applied Numerical Mathematics*, 58(12):1830–1843, Nov. 2007.

[2] C. 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, May 2007.

[3] A. N. Brooks and T. J. R. Hughes. "Streamline upwind/Petrov-Galerkin formulations for convection dominated flows with particular emphasis on the incompressible Navier-Stokes equations", *Computer Methods in Applied Mechanics and Engineering*, 32(1–3):199–259, Sept. 1982.

[4] P. Colella, T. H. D. Jr., W. D. Gropp, and D. E. Keyes. A science–based case for large–scale simulation. Technical report, Office of Science, US Department of Energy, July 2003.

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

[6] E. Elsen, P. LeGresley, and E. Darve. "Large calculation of the flow over a hypersonic vehicle using a GPU", *Journal of Computational Physics*, 227(24):10148–10161, Dec. 2008.

[7] Z. Fan, F. Qiu, A. Kaufman, and S. Yoakum-Stover. GPU cluster for high performance computing. In *SC '04: Proceedings of the 2004 ACM/IEEE conference on Supercomputing*, page 47, Nov. 2004.

[8] K. Fatahalian and M. Houston. "A closer look at GPUs", *Communications of the ACM*, 51(10):50–57, Oct. 2008.

[9] D. Göddeke, R. Strzodka, J. Mohd-Yusof, P. S. McCormick, S. H. Buijssen, M. Grajewski, and S. Turek. "Exploring weak scalability for FEM calculations on a GPU-enhanced cluster", *Parallel Computing*, 33(10–11):685–699, Sept. 2007.

[10] D. Göddeke, R. Strzodka, J. Mohd-Yusof, P. S. McCormick, H. Wobker, C. Becker, and S. Turek. "Using GPUs to improve multigrid solver performance on a cluster", *International Journal of Computational Science and Engineering*, 4(1):36–55, Nov. 2008.

[11] D. Göddeke, R. Strzodka, and S. 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, Jan. 2007.

[12] D. Göddeke, H. Wobker, R. Strzodka, J. Mohd-Yusof, P. S. McCormick, and S. Turek. "Co-processor acceleration of an unmodified parallel solid mechanics code with FEASTGPU", *accepted for publication in the International Journal of Computational Science and Engineering*, 2008.

[13] D. E. Keyes. "Terascale implicit methods for partial differential equations", In X. Feng and T. P. Schulze, editors, *Recent Advances in Numerical Methods for Partial Differential Equations and Applications*, volume 306 of *Contemporary Mathematics*, pages 29–84. American Mathematical Society, Jan. 2002.

[14] S. Kilian. *ScaRC: Ein verallgemeinertes Gebietszerlegungs-/Mehrgitterkonzept auf Parallelrechnern*. PhD thesis, Universität Dortmund, Fachbereich Mathematik, Jan. 2001.

[15] M. Köster, D. Göddeke, H. Wobker, and S. Turek. How to gain speedups of 1000 on single processors with fast FEM solvers — benchmarking numerical and computational efficiency. Technical report, Fakultät für Mathematik, TU Dortmund, Oct. 2008.

[16] M. F. Murphy, G. H. Golub, and A. J. Wathen. "A note on preconditioning for indefinite linear systems", *SIAM Journal on Scientific Computing*, 21(6):1969–1972, May 2000.

[17] J. D. Owens, M. Houston, D. Luebke, S. Green, J. E. Stone, and J. C. Phillips. "GPU computing", *Proceedings of the IEEE*, 96(5):879–899, May 2008.

[18] J. D. Owens, D. Luebke, N. Govindaraju, M. J. Harris, J. Krüger, A. E. Lefohn, and T. J. Purcell. "A survey of general-purpose computation on graphics hardware", *Computer Graphics Forum*, 26(1):80–113, Mar. 2007.

[19] E. H. Phillips, Y. Zhang, R. L. Davis, and J. D. Owens. Rapid aerodynamic performance prediction on a cluster of graphics processing units. In *Proceedings of the 47th AIAA Aerospace Sciences Meeting*, number AIAA 2009-565, Jan. 2009.

[20] J. Stam. "Stable fluids", In *Proceedings of SIGGRAPH 1999*, pages 121–128, 1999.

[21] S. Turek. *Efficient Solvers for Incompressible Flow Problems: An Algorithmic and Computational Approach*. Springer, 1999.

[22] S. Turek, C. Becker, and S. Kilian. “Hardware–oriented numerics and concepts for PDE software”, *Future Generation Computer Systems*, 22(1-2):217–238, Feb. 2004.

[23] S. Turek, D. Göddeke, C. Becker, S. H. Buijssen, and H. Wobker. “FEAST – Realisation of hardware-oriented numerics for HPC simulations with Finite Elements”, *submitted to Concurrency and Computation, Practice and Experience, Special Issue Proceedings of ISC 2008*, 2008.

[24] S. Turek, A. Ouazzi, and J. Hron. “On pressure separation algorithms (PSepA) for improving the accuracy of incompressible flow simulations”, *International Journal for Numerical Methods in Fluids*, 59(4):387–403, Apr. 2008.

[25] S. Turek and M. Schäfer. “Benchmark computations of laminar flow around a cylinder”, In E. Hirschel, editor, *Flow Simulation with High-Performance Computers II*, volume 52 of *Notes on Numerical Fluid Mechanics*, pages 547–566. Vieweg, 1996. co. F. Durst, E. Krause, R. Rannacher.