Velocity-Pressure Coupling on GPU

M. Emans M. Liebmann

SFB-Report No. 2013–003 February 2013

Supported by the
Austrian Science Fund (FWF)

SFB sponsors:

- **Austrian Science Fund** (FWF)
- **University of Graz**
- **Graz University of Technology**
- **Medical University of Graz**
- **Government of Styria**
- **City of Graz**

Das Land Steiermark

Stadt GRAZ Wissenschaft

VELOCITY-PRESSURE COUPLING ON GPU

MAXIMILIAN EMANS[†] AND MANFRED LIEBMANN[‡]

Abstract. We explore the possibilities to accelerate simulations in computational fluid dynamics (CFD) by additional graphics processing units (GPUs). By examining some examples of stationary incompressible flows from the industrial practice we demonstrate that the potential speedup obtained by deploying GPU accelerated linear solvers alone is limited if standard segregated algorithms are used. However, recently presented velocity-pressure coupling algorithms are an attractive alternative to these segregated algorithms. We present an efficient AMG solver for the coupled linear system of mixed elliptic-hyperbolic character and show that the GPU-accelerated version of this linear solver accelerates the velocity-pressure coupling scheme by almost 50% compared to a competitive CPU implementation. Compared to standard segregated methods, the computing time of the whole simulation is reduced by up to 75%.

1. Introduction

The graphic processing units (GPUs) on modern graphics boards are equipped with a large number of processing cores and with an on-board memory of reasonable size that is connected by a fast memory bus to the processing cores. The potential computational power of the GPUs has therefore attracted the attention of the scientific computing community looking for ways to accelerate numerical simulations. Important steps on the way towards numerical simulations on GPUs were e.g. the implementations of solver algorithms for linear systems by Krüger and Westermann [22], and solutions of systems of partial differential equations in two dimensions by Goodnight et al. [17], both published in 2003. However, the restrictions in particular with respect to the programming environment and the hardware were severe at that time. Recent development in both, hardware design and software tools made it possible to exploit the large computational power of the GPUs for numerical simulations of practical relevance in some areas which is in particular due to the development of efficient solver algorithms for the linear systems of equations. Examples for this are the red-black SOR by Konstantinidis [21] and the conjugate gradient implementation by Cevahir et al. [2]. With respect to CFD, however, multigrid solvers are particularly attractive: Göddeke et al. [16] and Haase et al. [18] report on significant speedup of such solvers for finite element calculations.

Although there is a strong interest in an acceleration of simulation software like commercial CFD software on GPUs, it is not easy to achieve this goal with reasonable effort since usually both, the global structure and the implementation of the compute-intensive parts of the code of long-running software projects is not well suited for a GPU implementation. But the existing implementations often contain a large amount of experience with regard to both, numerical and modelling

[†]Johann Radon Institute for Computational and Applied Mathematics (RICAM), and Industrial Mathematics Competence Centre GmbH (IMCC), both Linz, Austria

[‡]Institute for Mathematics and Scientific Computing, University of Graz, Graz, Austria

E-mail addresses: `maximilian.emans@ricam.oeaw.ac.at`, `manfred.liebmann@uni-graz.at`.

issues, that is in particular essential for the solution of real-world problems. It is not feasible to re-start such a software project from scratch in order to achieve better performance by GPU acceleration. The way that has to be chosen is therefore to re-use as much as possible of the existing implementation while the accelerating effect is achieved through re-implementation of compute-intensive parts of the program.

The numerical kernel of each software in CFD is an algorithm which solves the Navier-Stokes equations. CFD software packages with commercial relevance are based on the concept of the finite volumes, see Ferziger and Perić [15]. Robust discretisation techniques for unstructured meshes with collocated variable arrangement are the method of choice in the industrial practice. From the discretisation, a system of algebraic equations is obtained. This system is still nonlinear and coupled, i.e. the velocity components and the pressure as physical unknowns appear in the same equations.

With regard to application in commercial CFD tools, the most important discretisation technique is that of the finite volumes on unstructured grids with collocated variable arrangement, see Ferziger and Perić [15], and the algorithm of choice to solve the resulting system of algebraic equations is SIMPLE ("Semi-implicit pressure-linked equation") by Patankar and Spalding [25]. It is based on a segregated approach, i.e. only linear systems for a single physical unknown (pressure, velocity components) are solved at a time. Because of this, the SIMPLE algorithm (and methods derived from it) has moderate memory requirements and it is robust, but eventually, the convergence is slow and a large number of iterations might be necessary. If we want to use the efficient GPU accelerated solvers for these linear systems and keep the remaining calculations including the setup of the algebraic system on the CPU with mainly untouched source code, we need to transfer the whole data for each of these systems to the GPU and the calculated solution back to the CPU. But the data transfer between the CPU memory and the GPU memory is slow, i.e. about one order of magnitude slower than the on-chip data transfer on the CPU. Besides, in order to obtain fast running code on the GPU, other matrix data formats than on the CPU are used such that a conversion is needed, see e.g. Haase et al. [18], Cevahir et al. [2]. Therefore one cannot expect to obtain an efficient GPU acceleration if only the solution of the linear problems is accelerated on the GPU. If it is deemed not feasible to transfer significant parts of the setup of the system of algebraic equations to the GPU which would require a major restructuring of large parts of the source code, then it might be still possible to reconsider the numerical method which is used to find a solution to the system of algebraic equations.

In fact, there are different attractive algorithms for the solution of this particular system. Coupled approaches, i.e. methods were the solution of linear systems for more than one physical unknown at a time is required, appear to have gained some attention recently. With regard to the incompressible flows, "coupling" the variables means that the momentum equations and the pressure-correction equations form a single linear system. Chen et al. [3] and Darwish et al. [5] have devised an algorithm that can replace the SIMPLE scheme in existing CFD codes since the coupled system relies on the same operators as segregated SIMPLE. This makes it possible to use the existing code to assemble the coefficients. The advantages of the coupled approach (compared to segregated SIMPLE) are more rapid convergence and better parallel scalability due to the fact that a larger portion of the computational work incurs in well structured parts of the program, i.e. the linear solver which has usually a better parallel efficiency than the rest of the program. But the disadvantage of the coupled solver is the necessity to solve a large linear system: While in the segregated approach the number of unknowns is equal to the number of finite volumes, the velocity-pressure coupling requires to solve a system with four times more unknowns. Moreover, while the systems of the momentum equations in the segregated scheme are usually diagonal dominant and the matrix of the pressure-correction equation is symmetric positive definite, the large system of the velocity-pressure coupling scheme lacks such properties; it is therefore more difficult to compute a numerical solution of the latter system.

In this paper we combine innovative coupled solution techniques for the Navier-Stokes equations (that can be used to solve advanced real-world CFD problems from the industrial practice) and fast appropriate implementations of linear solvers on GPUs. The resulting program that we present here has preserved the implementation of the discretisation of the physical problem in the form of program code while it delivers – by means of GPU acceleration – the numerical solution of the discretised problem significantly faster than the conventional program running on CPU-based machines. For this, we proceed as follows: In section 2 we recall the segregated and the coupled numerical schemes that are used in CFD. In section 3 we present our solvers for the linear systems and we introduce our GPU implementation of these algorithms. In section 4 we present three benchmarks from industrial applications of a CFD program that are run on a cluster equipped with both, modern conventional CPUs and graphics boards for scientific computing. Section 5 contains our conclusions.

2. Segregated and coupled solution in CFD

For the sake of compactness, let us denote the discrete velocity field by $\vec{u}$, where

$$\vec{u} = \begin{pmatrix} \vec{u}_x \\ \vec{u}_y \\ \vec{u}_z \end{pmatrix} \tag{1}$$

contains the three components (in the directions of the cartesian coordinate system) $\vec{u}_x$, $\vec{u}_y$, and $\vec{u}_z$ of the velocity vector. The discrete pressure is $\vec{p}$. The finite volume discretisation momentum and the continuity equation may be written as

$$A(\hat{\vec{u}})\vec{u} + M\vec{p} = \vec{b} \tag{2}$$

$$C\hat{\vec{u}} = \vec{c}. \tag{3}$$

Here, A denotes the discretised and linearised operator that acts on the velocity field in the momentum equations, i.e. it expresses convective, diffusive, and eventually inertia components. M is the discretisation of the pressure term, $\vec{b}$ the body force term, C represents the discretised continuity equation, and $\vec{c}$ is a mass source in the incompressible case. The hat ($\hat{}$) on top of the vectors indicates that the vector is discretised on the grid that corresponds to the cell faces. The discretisation of the operators requires that both, the velocity in the cell centre $\vec{u}$ and the velocity at the cell faces $\hat{\vec{u}}$ appear in the system of equations. Both variables are linked by a linear interpolation operator Ω.

In the iterative SIMPLE algorithm, the velocity field that is calculated in the m-th iteration is split into a tentative velocity field $\vec{u}^*$ that is calculated from the momentum equations with the pressure from the previous iteration, and a velocity update $\vec{u}'$

$$\vec{u}^{(m)} = \vec{u}^* + \vec{u}'. \tag{4}$$

A similar splitting is applied to the pressure:

$$\vec{p}^{(m)} = \vec{p}^{(m-1)} + \vec{p}', \tag{5}$$

where $\vec{p}'$ is referred to as the pressure-correction.

If a collocated variable arrangement is chosen, i.e. all variables are placed in the cell centres, then in the SIMPLE algorithm for incompressible flows, see Patankar and Spalding [25], it is not appropriate to apply a linear interpolation scheme Ω (acting on all three components of $\vec{u}$, M, and so on at once) to $\vec{u}^*$ in order to evaluate an expression that substitutes $\hat{\vec{u}}^*$ in the continuity equation. Instead, the interpolation of Rhie and Chow [26] is applied which uses

$$\hat{\vec{u}} = \Omega(\vec{u} + A_D^{-1} M \vec{p}^{(m-1)}) - \hat{A}_M \hat{M} \vec{p}^{(m-1)}. \tag{6}$$

The approximations to A are defined as

$$A_D := \mathsf{diag}(A) \quad \text{and} \quad \hat{A}_M := [\Omega A_D^{-1} \vec{1}], \tag{7}$$

and the operator $\hat{M}$ is the gradient operator on the cell faces that uses the difference of the values at the adjacent cells as one would intuitively do. With this, the link between the pressure-correction and the velocity update is deduced

$$\hat{\vec{u}}' = -\hat{A}_M \hat{M} \vec{p}' \tag{8}$$

while $-\Omega A_D^{-1}(A - A_D)\vec{u}'$ is neglected. The pressure-correction equation for the SIMPLE algorithm for collocated grids is then obtained by substituting $\hat{\vec{u}}^{(m)} = \hat{\vec{u}}^* + \hat{\vec{u}}'$ into the continuity equation

$$-C\hat{A}_M \hat{M} \vec{p}' = \vec{c} - C\hat{\vec{u}}^*. \tag{9}$$

Thus the SIMPLE algorithm on collocated grids for incompressible flows is algorithm 1. Note that step 1 of this algorithm requires the solution of three different

Algorithm 1 Segregated SIMPLE, see Patankar and Spalding [25]

```
(1) solve $A(\hat{\vec{u}})\vec{u} + M\vec{p} = \vec{b}$ for $\vec{u} = \vec{u}^*$
(2) evaluate $\hat{\vec{u}} = \Omega(\vec{u} + A_D^{-1} M \vec{p}^{(m-1)}) - \hat{A}_M \hat{M} \vec{p}^{(m-1)}$
(3) assemble coefficients and right-hand side of $-C\hat{A}_M \hat{M} \vec{p}' = \vec{c} - C\hat{\vec{u}}^*$ and
    solve for $\vec{p}'$
(4) compute $\hat{\vec{u}}'$ with $\hat{\vec{u}}' = -\hat{A}_M \hat{M} \vec{p}'$
(5) update $\vec{u}^{(m)} = \vec{u}^* + \vec{u}'$ and $\vec{p}^{(m)} = \vec{p}^{(m-1)} + \vec{p}'$
```

linear systems. They are set up and solved one after the other, once per iteration of SIMPLE. The solution of these systems is fairly easily obtained since the systems reflect the hyperbolic part of the problem and the matrices are diagonal-dominant. Moreover, within the context of SIMPLE, it is sufficient to reduce the residual with respect to some norm by a factor of 10 only. The system in step 2 reflects the elliptic part of the problem, i.e. the mass conservation. A more accurate solution is required for this system, i.e. the residual usually needs to be reduced by a factor of 50-1000. For incompressible problems the system is symmetric and positive semi-definite. Since for each unknown a separate system is solved, this type of algorithm is referred to as segregated solution.

In a coupled solution, eqns. (2) and (3) are solved at once. Let us therefore write down this system in a way where the vector $\vec{u}$ is split into the velocity components like in eqn. (1); the discrete body force $\vec{b}$ is split in an analogous manner:

$$\vec{b} = \begin{pmatrix} \vec{b}_x \\ \vec{b}_y \\ \vec{b}_z \end{pmatrix} \tag{10}$$

The contributions of the components of the velocity vector in the momentum, i.e. the block matrices A_x, A_y, and A_z are exactly the system matrices which the linear systems in step 1 of algorithm 1 is solved for. The block matrices M_x, M_y, and M_z

can also be taken directly from the SIMPLE scheme since they are used to assemble the right-hand side of these systems. In this notation, the coupled system is

$$\begin{pmatrix} A_x & 0 & 0 & M_x \\ 0 & A_y & 0 & M_y \\ 0 & 0 & A_z & M_z \\ C_xS & C_yS & C_zS & 0 \end{pmatrix} \begin{pmatrix} \vec{u}_x \\ \vec{u}_y \\ \vec{u}_z \\ \vec{p} \end{pmatrix} = \begin{pmatrix} \vec{b}_x \\ \vec{b}_y \\ \vec{b}_z \\ \vec{c} \end{pmatrix}. \tag{11}$$

There is a zero block on the block diagonal in the continuity equations which makes this system hard to treat. But by using some of the relations that are also used in the derivation of SIMPLE, one can reformulate this equation in a manner that the zero block is filled and, at the same time, the interpolation of Rhie and Chow [26] is introduced, see Chen et al. [3] and Darwish et al. [5]. For this, the continuity equation (3) is transformed into a pressure-correction equation for the velocity-pressure coupling scheme simply by substituting equation (6) into (3). The resulting equation reads

$$C\Omega\vec{u} - C\hat{A}_M\hat{M}\vec{p} = C\Omega A_D^{-1}M\vec{p}. \tag{12}$$

Ω interpolates the three velocity components independently from the cell centre to the cell faces. If we denote the three identical blocks of Ω as S, we can write the entire coupled system as

$$\begin{pmatrix} A_x & 0 & 0 & M_x \\ 0 & A_y & 0 & M_y \\ 0 & 0 & A_z & M_z \\ C_xS & C_yS & C_zS & -C\hat{A}_M\hat{M} \end{pmatrix} \begin{pmatrix} \vec{u}_x \\ \vec{u}_y \\ \vec{u}_z \\ \vec{p} \end{pmatrix} = \begin{pmatrix} \vec{b}_x \\ \vec{b}_y \\ \vec{b}_z \\ C\Omega A_D^{-1}M\vec{p} \end{pmatrix}. \tag{13}$$

Since the problem is nonlinear, see eqns. (2) and (3), a single numerical solution of equation (13) is not enough. Instead, one has to approach the solution iteratively by solving system (13) several times where each time the coefficients are recalculated based on the most recent available values of the unknowns.

3. Numerical solution of the linear systems

In the SIMPLE algorithm, we chose to solve the linear approximations of the momentum equations by means of a BiCGstab algorithm that is preconditioned by an ILU(0) algorithm, i.e. by an incomplete LU factorisation with zero fill-in, see Saad [27]. Usually one iteration of this solver is sufficient. The pressure-correction equation of this algorithm is solved by means of k-cycle AMG, see Notay [24]; it has been demonstrated in Emans [12] that this algorithm is efficient for our type of application.

3.1. **AMG for the coupled linear system.** The numerical solution of system (13) requires particular care since its computational cost determines essentially the cost of the entire procedure. While for the systems in the segregated SIMPLE one can rely on three decades of experience, the solution of system (13) is an actual challenge. Let us therefore consider potential algorithms for this kind of problem in more detail. While Darwish et al. [5] restrict themselves to two-dimensional problems and use an algebraic multigrid solver with ILU(0) smoother for the solution of system (13), Chen et al. [3] rely on BiCGstab with ILU(0) preconditioning. Note that in both papers the described example problems are fairly small problems (maximum $3 \cdot 10^5$ finite volumes in the case of Darwish et al. [5] and only 40000 finite volumes in the case of Chen et al. [3]). In both papers, the spatial discretisation could be done by meshes with a fairly good quality; moreover, no parallel computers were used by either group. While the solver of Chen et al. [3] does not have the favourable scalability of multigrid algorithms, the solver of Darwish et al.

[5] appears to be adjusted to two-dimensional problems and its parallelisation is not straight forward. Since we intend to solve problems with several millions of finite volumes, we must use parallel computers, and we need algorithms with both, good parallel performance on these computers and good convergence for large problems. Moreover, our solver must be robust enough for meshes with a quality typical for industrial applications. We prefer therefore to construct our own solver based on the experience we had with both, segregated (elliptic) problems, see Emans [6, 8], and coupled problems of another kind, see Emans [11, 13].

Let us therefore denote the system (13) for the sake of simplicity as

$$A\vec{x} = \vec{b} \tag{14}$$

where $A \in \mathbb{R}^{n\times n}$ is regular, $\vec{b} \in \mathbb{R}^n$ is the right-hand side vector and $\vec{x} \in \mathbb{R}^n$ the solution; n is the number of unknowns. We assume that the matrix is given in Compressed Row Storage (CRS) format as e.g. described by Falgout et al. [14].

Any variant of the AMG algorithm, see e.g. Algorithm 2, requires the definition of a grid hierarchy with l_{max} levels $A_l \in \mathbb{R}^{n_l\times n_l}$ $(l = 1, ..., l_{max})$ with system size n_l where $n_{l+1} < n_l$ holds for $l = 1, ..., l_{max} - 1$ and $A_1 = A$ as well as $n_1 = n$. As it is common practice in algebraic multigrid, the coarse-grid operators are defined recursively, starting on the finest grid, by

$$A_{l+1} = P_l^T A_l P_l \qquad (l = 1, ..., l_{max} - 1). \tag{15}$$

where the prolongation operator P_l has to be determined for each level l while the restriction operator is defined as P_l^T. It is the choice of the coarse-grid selection scheme that determines the elements of all P_l and consequently the entire grid hierarchy. The definition (or calculation) of the elements of P_l for all levels and the computation of the operators A_l $(l = 2, ..., l_{max})$ are referred to as setup phase of AMG. The choice of the prolongation operator decisively influences the cost of the setup whereof a large portion is the evaluation of eqn. (15), but of course, it has also decisive influence on the convergence of the method and on the cost of each operations within the multigrid cycle. With regard to calculations on GPUs it has to be noted that, due to the algorithmic structure, the solution phase can be accelerated much better on the GPU than the setup phase. For our purpose it is therefore essential that the setup is computationally as cheap as possible.

3.2. **Definition of the prolongation operators.** The prolongation operator P_l maps a vector on the coarse grid $\vec{x}_{l+1}$ to a vector on the fine grid $\vec{x}_l$:

$$\vec{x}_l = P_l \vec{x}_{l+1} \tag{16}$$

We restrict ourselves to the class of aggregation methods that splits the number of nodes on the fine grid into disjoint sets of nodes, the so-called aggregates that act as nodes on the coarse grid, see e.g. Darwish et al. [4]. The mapping from the coarse grid to the fine grid is then achieved by simply assigning the coarse-grid value of the aggregate to all the fine-grid nodes belonging to this aggregate. This corresponds to a constant interpolation; the consequence is that there is only one non-zero entry in each row of P_l with the value one such that the evaluation of eqn. (15) simplifies to an addition of rows of the fine-grid operator. Compared to the so called classical AMG methods, the aggregation methods are characterised by a cheap setup. Therefore they are well suited for calculations with GPU accelerated solution phase.

The particular treatment technique for the coupled system has been discussed previously, see Emans [11]. Here, we apply the same principles with regard to the appropriate definition of the prolongation operators, i.e. a point-based coarsening scheme is used: The aggregation calculated for the elliptic part of the problem, i.e. the pressure-correction equation, is applied to all unknowns.

Algorithm 2 AMG cycle

```
$\vec{x}_l^{(3)}$ = AMG($l$, $\vec{r}_l$, $\vec{x}_l^{(0)}$, $C$)
    Input:   level $l$, right-hand side $\vec{r}_l$, initial guess $\vec{x}_l^{(0)}$,
             type of cycle $C$: "v-cycle" or "k-cycle"
    Output:  approximate solution $\vec{x}_l^{(3)}$

1: pre-smoothing: $\vec{x}_l^{(1)} = S_l(\vec{r}_l, A_l, \vec{x}_l^{(0)})$
2: restriction: $\vec{r}_{l+1} = P_l^T(\vec{r}_l - A_l\vec{x}_l^{(1)})$
3: if $l = l_{max} - 1$ then
4:     solution of coarse-grid system: $\vec{x}_l^{(3)} = A_{l+1}^{-1}\vec{r}_{l+1}$
5: else
6:     if $C$ = "v-cycle" then
7:         coarse-grid correction: $\vec{x}_{l+1}$ = AMG($l+1$, $\vec{r}_{l+1}$, $\vec{0}$,"v-cycle")
8:     else if $C$ = "k-cycle" then
9:         apply Krylov-solver to approximate $\vec{x}_{l+1}$ in $A_{l+1}\vec{x}_{l+1} = \vec{r}_{l+1}$ with AMG
           preconditioner (applied to vector $\vec{z} \in \mathbb{R}^{n_{l+1}}$ defined by AMG($l+1$, $\vec{z}_{l+1}$,
           $\vec{0}$,"k-cycle") )
10:    end if
11: end if
12: prolongation of coarse-grid solution and update: $\vec{x}_l^{(2)} = \vec{x}_l^{(1)} + P_l\vec{x}_{l+1}$
13: post-smoothing: $\vec{x}_l^{(3)} = S_l(\vec{r}_l, A_l, \vec{x}_l^{(2)})$
```

3.2.1. *Pairs-of-Pairs algorithm.* The first step of the Pairs-of-Pairs algorithm that has been used by Notay [24] is the aggregation of the set of nodes into aggregates of pairs of nodes. For this we use algorithm 3.

Algorithm 3 Pairwise aggregation (by Notay [24], simplified version)

```
Input:           Matrix $A = (a_{ij})$ with $n$ rows and rows.
Output:          Number of coarse variables $n_c$ and aggregates $G_i$, $i = 1, ..., n_c$
                 (such that $G_i \cap G_j = \emptyset$ for $i \neq j$).
Initialisation:  $U = [1, n]$
                 for all $i$: $S_i = \{j \in U \setminus \{i\} \mid a_{ij} < -1/4 \max_{(k)}|a_{ik}|\}$,
                 for all $i$: $m_i = |\{j | i \in S_j\}|$,
                 $n_c = 0$.
Algorithm:       While $U \neq \emptyset$ do:
                 1. select $i \in U$ with minimal $m_i$; $n_c = n_c + 1$.
                 2. select $j \in U$ such that $a_{ij} = \min_{k \in U} a_{ik}$
                 3. if $j \in S_i$: $G_{n_c} = \{i, j\}$, otherwise $G_{n_c} = \{i\}$
                 4. $U = U \setminus G_{n_c}$
                 5. for all $k \in G_{n_c}$: $m_l = m_l - 1$ for $l \in S_k$
```

For the Pairs-of-Pairs aggregation, the set of aggregates obtained with algorithm 3 is used to define the intermediate prolongation operator P_{l1}. With this, the elements of the corresponding intermediate coarse-grid matrix $A_{l+1/2} = P_{l1}^T A_l P_{l1}$ are calculated with eqn. (15). In order to obtain the matrix A_{l+1}, the Pairs-of-Pairs algorithm applies the same procedure a second time, this time with input $A_{l+1/2}$ instead of A_l for algorithm 3 which gives rise to the prolongation operator P_{l2}. A_{l+1} is calculated as $A_{l+1} = P_{l2}^T A_{l+1/2} P_{l2}$. The final prolongation operator is formally $P_l = P_{l2}P_{l1}$. Since it contains only the information to which coarse-grid element or aggregate a fine-grid node is assigned, it is sufficient to store it as an array of size n_l carrying the index of the coarse-grid node. The operators $A_{l+1/2}$, P_{l1}, and P_{l2}

are discarded after A_{l+1} has been calculated. The parallel version of the method restricts the aggregates to nodes belonging to the same parallel domain. We refer to the method as PP.

3.2.2. *Plain aggregation algorithm.* Our plain aggregation algorithm comprises the following steps:

(1) **Determine strong connectivity:** Edges of the graph of the matrix A_l for which the relation
$$|a_{ij}| > \beta \cdot max_{(j)}|a_{ij}| \tag{17}$$
holds are marked as strong connections. The criterion β depends on the level of the grid hierarchy l and it is defined according to Vaněk et al. [28] as
$$\beta := 0.08\left(\frac{1}{2}\right)^{l-1}. \tag{18}$$
(2) **Start-up aggregation:** All nodes are visited in the arbitrary order of their numeration. Once a certain node i is visited in this process, a new aggregate is built if this node is not yet assigned to another aggregate. Each of the neighbours of node i that is strongly connected to this node and that is not yet assigned to another aggregate is grouped into this aggregate as long as the number of nodes is lower than the maximum allowed aggregate size.
(3) **Enlarging the decomposition sets:** Remaining unassigned nodes are joined to aggregates containing any node they are strongly connected to as long as the number of nodes in this aggregate is lower than twice the maximum allowed aggregate size. If there is more than one strongly connected node in different aggregates, the one with the strongest connection determines the aggregate this node is joined with.
(4) **Handling the remnants:** Unassigned nodes are grouped into aggregates of a strongly connected neighbourhood. Twice the maximum allowed aggregate size is allowed.

This algorithm follows closely the one proposed by Vaněk et al. [28] with the essential difference that we restrict the number of nodes per aggregate which gives rise to a parameter of this algorithm. In step (3) we allow twice the maximum number of nodes in order to avoid a large number of single-point aggregates. Usually only a few such enlarged aggregates are formed.

The aggregates generated this way are used in a scheme with constant interpolation, i.e. in a way that all fine-grid nodes assigned to a certain aggregate receive the value of the coarse-grid node this aggregate forms on the coarse-grid. The corresponding prolongation operator will have a similarly simple structure as that of the described Pairs-of-Pairs aggregation method. The coarse-grid operator is also calculated with eqn. (15), exactly in the same manner as the operators $A_{l+1/2}$ and A_{l+1} in the Pairs-of-Pairs aggregation. The coarsening scheme defined by this procedure will be only useful, if the maximum number of nodes per aggregate is kept relatively low. In preliminary experiments we found, a chosen maximum number of nodes per aggregate of 6 turned out to result in an efficient and robust algorithm with good convergence properties. We denote this aggregation scheme in a k-cycle scheme as R6.

3.2.3. *Mixed aggregation scheme.* The Pairs-of-Pairs aggregation has the disadvantage that it requires the calculation of the intermediate operator $A_{l+1/2}$ that is later discarded. The calculation of this operator is rather expensive. In particular on the fine grids where the system is large, this is a significant contribution to the computational cost of the setup. The fact that the operators tend to become denser on the first coarse grids makes the calculation of the associated operators additionally expensive. But this scheme is very robust, i.e. it is able to produce good quality meshes, rather independent of the operator structure, i.e. on all levels of the grid hierarchy, also on the coarse grids. The plain aggregation can significantly faster generate a coarse-grid operator with a similar system size as the Pairs-of-Pairs algorithm since it calculates this operator directly. But the convergence of this scheme is worse than that obtained with the Pairs-of-Pairs scheme and, with increasing density of the operator, this scheme tends to produce an increasing number of single points that are cannot be joined to one of the aggregates. For some problems, this can lead to an inefficient coarsening with a low reduction of nodes from one grid level to the next.

Mixing both methods in a way that the fine levels are generated by the plain aggregation while the coarse levels are generated by the Pairs-of-Pairs algorithm appears to be a good way to exploit the strengths of both algorithms while reducing their disadvantages. Our mixed aggregation starts with the plain aggregation with a maximum aggregate size of 6 and switches to the Pairs-of-Pairs algorithm as soon as the maximum number of elements per row becomes greater than 50. Here, we refer to this algorithm as PR.

3.2.4. *Further aspects of the AMG algorithm.* Our experience with the velocity-pressure coupling has shown that the robustness of GMRES-based k-cycle AMG, see Emans [13], is indispensable in the first iteration of this scheme. Later on, if the current solution is no longer far away from the physical solution, the systems become easier to solve. For these cases we employ an f-cycle preconditioned restarted GMRES. On the CPU we use a ILU(0) smoother with one iteration in both cases. On the GPU, we use in both cases an ω-Jacobi smoother with 2 pre-smoothing sweeps and two post-smoothing sweeps where the relaxation parameter $\omega = 0.33$. The coarse-grid system is solved directly in parallel by MUMPS, see Amestoy et al. [1].

The described scheme has been implemented into the CFD software package FIRE$^{(R)}$ 2011. The solver part of this program is coded in Fortran 90 and compiled with the Intel Fortran compiler, version 11.1. The parallelisation of the solver follows the domain decomposition provided by FIRE 2011 with an overlap of one layer of finite volumes. This decomposition is obtained by METIS, see Karypis and Kumar [20]. The parallelisation of the AMG solver is described in Emans [9].

Relevant for the forthcoming discussion is the way the data exchange is implemented. The data of the domain boundaries is exchanged for the matrix-vector multiplications and the smoothing sweeps. This is a point-to-point communication which is implemented by means of the asynchronous (MPI: “immediate send”) transfer protocol. This means that while the exchange takes place, internal work, i.e. operations that do not depend on the data coming in from the neighbours, is done. The remaining part of the computation that depends on the data from the neighbours is carried out as soon as both have finished, the internal computation and the exchange. This principle is illustrated in figure 1. This way, the communication and the internal work overlap and the overhead of the parallelisation is minimised. Apart from this type of communication, some collective communication events for reduction operations occur, e.g. for the computation of the scalar products of the conjugate gradients or GMRES, and norms for the control of the solution phase. The used MPI library is Platform MPI, version 7.01.

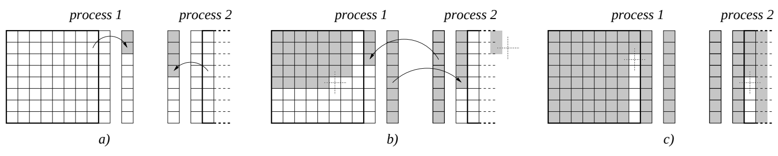


FIGURE 1. Pattern of inter-domain data exchange: a) preparation of data exchange: local boundary data is copied to a coherent array mapping the buffer cells of the neighbour, b) data exchange, i.e. update of buffer cells, and local computations take place simultaneously, c) local computations using information from neighbouring domains (buffer cells) are done.

3.3. **Implementation on GPU.** The setup phase is implemented conventionally on the CPU as it has been described in Emans [10]. After any matrix has been defined or calculated, it is translated into the Interleaved Compressed Row Storage (ICRS) format on the CPU and then transferred to GPU memory. The definition of this format is found in Liebmann [23]. The corresponding algorithm devised in the same publication is used to carry out matrix-vector operations. This applies to the system matrices on all levels. The particularly simple structure of the restriction and prolongation operators of the aggregation algorithms PP, R6, and P6 gives rise to a simplified version of the ICRS format: Since the value of all non-zero matrix elements is the same (one), it does not make sense to store these values. Therefore, only the number of elements per row, the displacement and the column index for each element is stored. The fill-in elements (due to the different number of elements per row) are identified by a negative column index and ignored in the matrix-vector multiplication kernel.

For an efficient parallel implementation, the concept of overlapping the data exchange with the internal operations – implemented by means of the asynchronous point-to-point exchange mechanism of MPI – has been extended: While the internal work is done on the GPU, the CPU manages the data exchange. The extraction of all values that need to be communicated to the neighbours is formulated as an operator B_d. The extraction is implemented on the GPU. The extracted values form the vector $\vec{x}_d^{(b)}$. If there are m neighbours, the components of the vector $\vec{x}_d^{(b)}$ need to be distributed to m vectors $\vec{x}_d^{(i)}$ $(i = 1, ..., m)$; each of these vectors is then transferred in an synchronous point-to-point communication to the corresponding neighbouring process i. As soon as the exchange has terminated and the values from the neighbouring processes are available as $\vec{x}_d^{(e)}$, the external contributions are calculated still on the CPU as $\vec{t}_d^{(b)} := E_d\vec{x}^{(e)}$. Finally they are transferred to the GPU and added to the result vector on the GPU. The mechanism is illustrated in figure 2. While in the conventional CPU implementation only the exchange is overlapped with the internal work, in this GPU implementation the distribution of the extracted vector with regard to the neighbouring processes, the actual exchange, and the calculation of the contributions depending on the neighbours are all overlapped with the internal work. This way, a parallelism between CPU and GPU has been implemented. The GPU related code is written in Cuda and compiled by the Nvidia compiler version 4.0.

4. BENCHMARKS

In this section we describe three benchmarks that demonstrate the benefits of the simulation with the velocity-pressure coupling that is accelerated by GPUs.

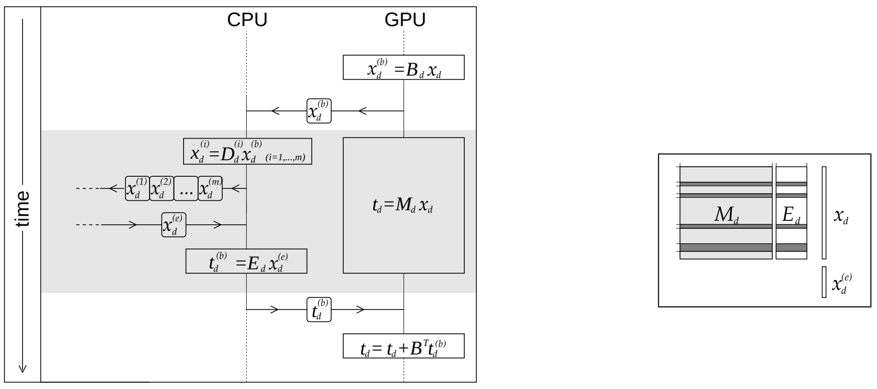


FIGURE 2. Flow chart of parallel matrix-vector product on machines with multiple GPUs, the parallel execution on CPU and GPU is marked grey (left), notation (right)

We have chosen typical stationary simulations in CFD from the industrial practice. Note, however, that the method we examine is also applicable with similar efficiency to time-dependent problems. The meshes have been generated in an automated manner, i.e. with regard to the quality of the mesh our selection reflects the requirements in the industrial practice. The meshes comprise mostly (around 90%) hexagonal cells, the contribution of tetrahedral elements is marginal. The problem sizes (number of finite volumes) reflect that of a wide range of industrial applications.

4.1. **Test cases and hardware.** Problem 1 is a pressure driven steady flow through the symmetric inlet geometry into the cylinder (intakeport) of a combustion engine. Half of the real geometry forms the computational domain; the face towards the missing half is a symmetric boundary condition. The domain is discretised with approximately 700000 cells; the maximum aspect ratio (ratio of the length of the longest edge and the length of the shortest edge of one cell) in this mesh is 11.2, the minimum angle between two adjacent faces is $0.1°$. As initial guess a potential flow solution in the given geometry is computed. Besides solid walls and the mentioned symmetric boundary condition, pressure boundaries confine the computational domain. The fluid is air, the applied pressure difference is 2.5kPa. The residual of the pressure-correction equation is required to be reduced by a factor of 200, that of the momentum equations by a factor of 10. The nonlinear iteration of the velocity-pressure coupling is stopped if the normalised residual of the mass conservation is lower than 10^{-5} and that of all three momentum equations is lower than 10^{-4}. All residuals are calculated in 1-norm. Turbulence is modelled by a standard k-ε model according to Jones and Launder [19]. SIMPLE requires 2233 iteration to reach the convergence criteria, the velocity-pressure coupling scheme 423.

Problem 2 is a flow through another intakeport. In this case, the geometry is non-symmetric such that the computational domain comprises the whole interior of this element. The geometry is resolved by around $1.9{\cdot}10^6$ finite volumes where the maximum aspect ratio in this mesh is 1230 and the minimum angle between two adjacent faces is $0.1°$. The boundary conditions are solid walls and, at the inlet and the outlet, pressure boundaries. The driving pressure difference is 2.0 kPa. The fluid is air. Initially, the fluid is at rest. Tolerances, convergence criteria, and turbulence modelling are the same as in problem 1. The velocity-pressure coupling scheme has reached the convergence criteria after 606 iterations while the SIMPLE scheme needs 2177 iterations.

Problem 3 is an external aerodynamics simulation. A sports vehicle is exposed to an air flow simulating a fast pace of the car on a flat surface. The flow at the face at the front side of the car enters the computational domain with a uniform, constant velocity of 69.4 m/s. The bottom face is a wall moving with the same speed. At the side faces and at the top face symmetry boundary conditions for both, velocity and pressure are enforced. At the rear face, constant pressure is set. The initial condition is stagnant fluid. The computational model consists of around $4.0{\cdot}10^6$ finite volumes where the maximum aspect ratio in this mesh is 218 and the minimum angle between two adjacent faces is 1.7°. Tolerances, convergence criteria, and turbulence modelling are identical with those of problem 1. In order to reach the convergence criteria the segregated SIMPLE algorithm requires 820 iterations; the velocity-pressure coupling scheme converges after 205 iterations.

The benchmarks were run on compute nodes of a GPU-cluster with two Intel X5650 CPUs and four Nvidia Tesla C2070 graphics boards. The most important specifications of the node related hardware are compiled in Table 1. The compute nodes of the GPU-cluster are connected by a 40 GB/s QDR Infiniband interconnect.

TABLE 1. Hardware specification

CPUs	Intel X5650
cores	2×6
L3-cache	2×6 MB, shared
main memory	96 GB
clock rate	2.67 GHz
memory bus	QPI, 26.7 GB/s

GPUs (Nvidia)	Tesla C2070
multiprocessors	4×14
L2-cache	4×768 KB
global memory	4× 5375 MB
memory clock rate	1.49 GHz
memory bus	136.8 GB/s

4.2. **Results.** Before a particular scheme is selected for a GPU implementation, one has to estimate to what extend it can be accelerated. In our case, the efficiency can be expected to be increased if the program consists to a large amount of sparse matrix-vector multiplications that are accelerated by means of the GPU implementation of Liebmann [23]. For this it is necessary to store the matrices in the ICRS format. In figure 3, the performance of a single matrix-vector multiply with sample sparse square matrices with a structure representing a finite volume discretisation of a problem with the corresponding size is shown (average of 10 measurements). The calculation on a GPU is about 20 times faster than that on a CPU for large matrices. But in order to obtain a good acceleration of the whole method, the matrix-vector multiplication with the same matrix should be carried out several times on the GPU since the conversion into the ICRS format (on the CPU) as well as the data transfer is expensive in terms of computing time, see also figure 3. The conversion of the matrix takes, depending on the problem size, 2 to 5 times more time than the execution of the matrix-vector multiplication on the CPU. Using the GPU-accelerated solver for all the systems that need to be solved within the segregated SIMPLE scheme cannot result in an efficient scheme since most of the time only one iteration of the ILU(0) preconditioned BiCGstab for the momentum equations (and also for other transport equations required e.g. for the turbulence models) in the segregated solution is required. One would slow down the whole scheme because copying the matrices is necessary each time a system is solved. Note that the matrices are never the same and it would take more time to copy the matrices than one could gain from the fast execution of the sparse matrix-vector multiplication kernels on the GPU.

For the pressure-correction equation where an AMG solver with typically a few iterations is usually used, a GPU implementation of this solver could make sense, but the accelerating effect of this is limited since the solution phase of the AMG

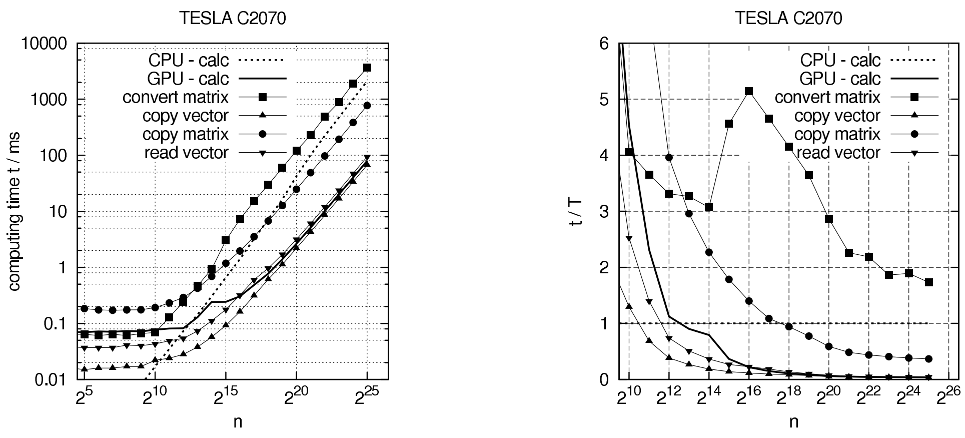


FIGURE 3. Computing time t for the matrix vector multiply (calc) of square sparse matrix with 7 random entries per row and n rows and for other necessary operations (left); ratio of computing time t for the matrix vector multiply (calc) on GPU and for other necessary operations and the computing time for the matrix vector multiply on CPU (T) for the same problems (right)

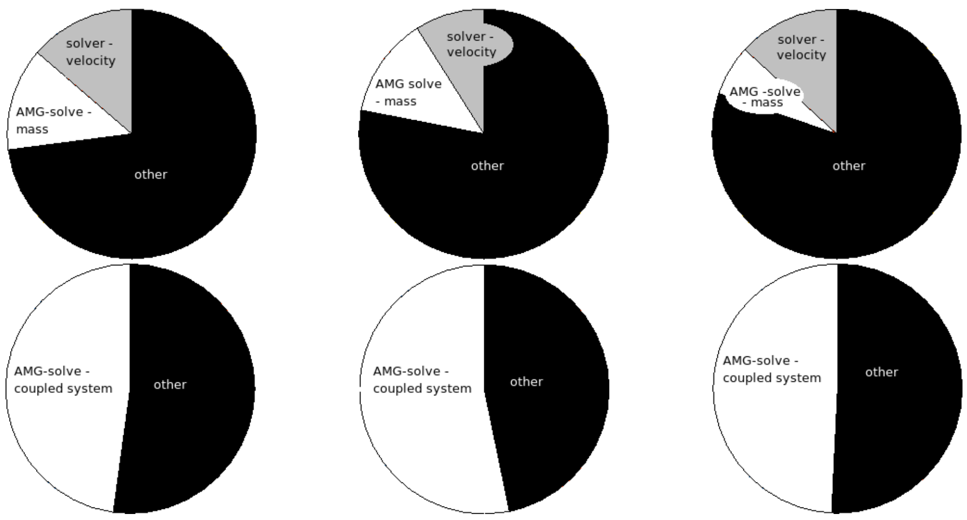


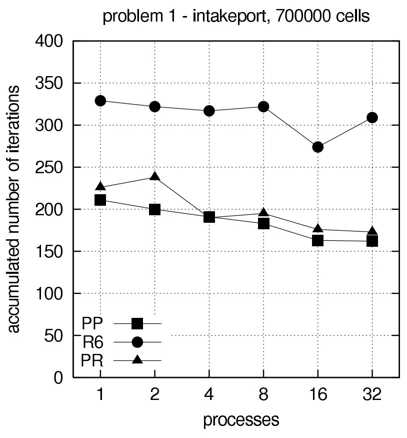
FIGURE 4. Contribution of some partial tasks to the total computing time for problem 1,2, and 3 (from left): segregated SIMPLE (top) and velocity-pressure coupling (bottom)

solver accounts only for a small portion of the total computing time. This can be seen from figure 4: In this figure, the computing time is split with respect to the possible accelerating effect of a GPU implementation of the linear solver. The figure shows the time spent in the solution phase of AMG that can be accelerated, the time spent in the solver of the momentum equations that cannot be accelerated since the data transfer is too expensive, and the remaining time. For the segregated SIMPLE, the remaining time ("other" in figure 4) comprises the assembly of the coefficients of the matrices of the momentum equations (step 1 of algorithm 1, i.e. the matrices A_x, A_y, and A_z in formula 11), the assembly of the matrix of the pressure-correction equation, the setup of the AMG solver, the update of the cell-face velocities (step

4 in algorithm 1), the turbulence modelling which requires among other things the solution of two transport equations, and other necessary operations like the calculation of the residuals. For the velocity-pressure coupling scheme, it comprises the assembly of all block-matrices in equation (13), the setup of the AMG solver for this system, the turbulence modelling, and other necessary operations. The time summarised as “other” in figure 4 is spent in various parts of the program that cannot be implemented as an efficient code in Cuda with reasonable effort due to the sheer number of code lines. The data in the figures 3 and 4 show that the potential reduction of the computing time by GPU-accelerated linear solvers in the segregated SIMPLE scheme is low. Using GPU-accelerated solvers for the momentum equations would not result in a reduction of the computing time at all and the effect of a GPU-accelerated AMG for the pressure-correction equation is limited to 10 – 15 %. We will therefore not present results of benchmarks of calculations with segregated SIMPLE and GPU-accelerated solvers.

Let us now examine the performance of the AMG solver for the velocity-pressure coupling scheme. We compare the Pairs-of-Pairs aggregation PP proposed by Notay [24], the plain aggregation R6 and the mixed aggregation RP. For this examination we have carried out some extra runs of the first 20 (nonlinear) iterations in problem 1 of the velocity-pressure coupling scheme. Figure 5 shows that the number of iterations of the plain aggregation scheme R6 is about 30% higher than the number of iterations of the Pairs-of-Pairs scheme PP; the mixed scheme RP converges almost as rapidly as the scheme PP. The cost of a single iteration of the R6 scheme is only about 10% less than that of PP and RP. Figure 6 shows the time spent in the linear solver for the coupled system, and the distribution between setup and solution phase for both, CPU calculation and GPU calculation. Substituting the Pairs-of-Pairs aggregation by the mixed aggregation reduces effectively the time spent on the setup by around 25%. The setup in the GPU-accelerated calculations takes more time than that of the CPU implementation since it comprises the transfer of the matrices to the GPU memory. On the CPU, the solution phase with the R6 scheme is, due to the slower convergence, significantly larger than that of the other two schemes. The total computing time with this scheme is therefore significantly longer than that of the other two schemes. Since the Pairs-of-Pairs scheme is robust and good experience has been reported, e.g. by Emans [13], we use this scheme for the conventional calculations on the CPU. In the GPU-accelerated computation, the solution phase of the R6 scheme is also significantly slower than that of the other schemes, but since the solution phase is efficiently accelerated, its contribution to the total computing time is low and the method with the R6 scheme is in total faster than the method with the Pairs-of-Pairs scheme. The computing time of the method with the mixed aggregation is about the same as that of the R6 method. Since the mixed method appears to be more robust than the plain aggregation R6 as it employs the Pairs-of-Pairs aggregation on the coarse grids, we use this mixed aggregation for the GPU calculations.

The computing times with the standard segregated SIMPLE, with the velocity-pressure coupling (VPC), calculated conventionally on CPUs, and with the velocity-pressure coupling with GPU acceleration are shown in figures 7, 8, and 9 for problems 1, 2, and 3, respectively. If we compare the segregated SIMPLE method and the velocity-pressure coupling on the CPU, it is noticeable that the coupled scheme converges faster than the segregated scheme, recall the number of iterations in section 4.1. But since the iterations of the coupled scheme are more expensive than those of the segregated scheme (note the time spent in the linear solver), the coupled scheme is not always faster. In particular if the number of processes is low, the segregated scheme appears to be equally fast or even slightly faster. The



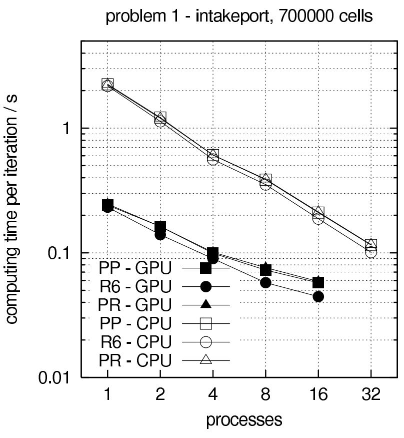
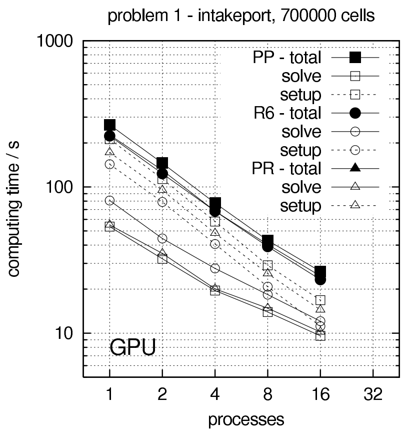


Figure 5. Accumulated number of iterations (left) and computing time per iteration (right) for problem 1 (first 20 iterations)



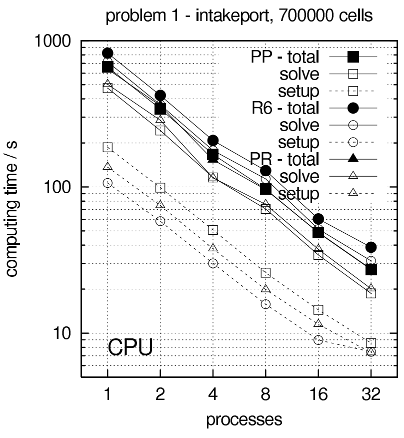


Figure 6. Computing times for the solution of the coupled systems in the first 20 iterations of problem 1 on GPU (left) and on CPU (right)

parallel efficiency of the coupled solution appears to be better due to the good parallel efficiency of the linear solver and the large portion of time that is spent in the linear solver. It has been discussed in previous publications, e.g. Emans [7], that a challenge of the parallel implementation of AMG is the communication on the coarse grids which tends to be expensive in comparison to the computational work for the associated small systems. In other words, on coarse grids the internal work finishes before the exchange terminates such that the program has to wait for the completion of the exchange. Since the coupled system is four times larger in terms of unknowns and ten times larger in terms of matrix entries than the systems of the segregated solver, additional waiting time on the coarse grids is smaller in comparison to the total computing time and the parallel efficiency appears to be better. The consequence of this is that the parallel computations (with more than two processes in our examples) with the velocity-pressure coupling are faster than the computations with the segregated SIMPLE.

With regard to the GPU acceleration of the velocity-pressure coupling it seems obvious that the acceleration of the linear solver brings the expected effect. Figure

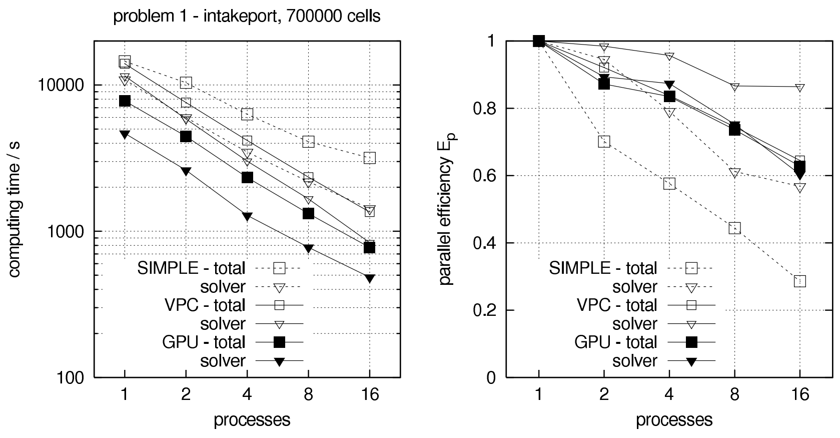


FIGURE 7. Computing times (left) and parallel efficiency (right) for the entire simulations with different algorithms for problem 1

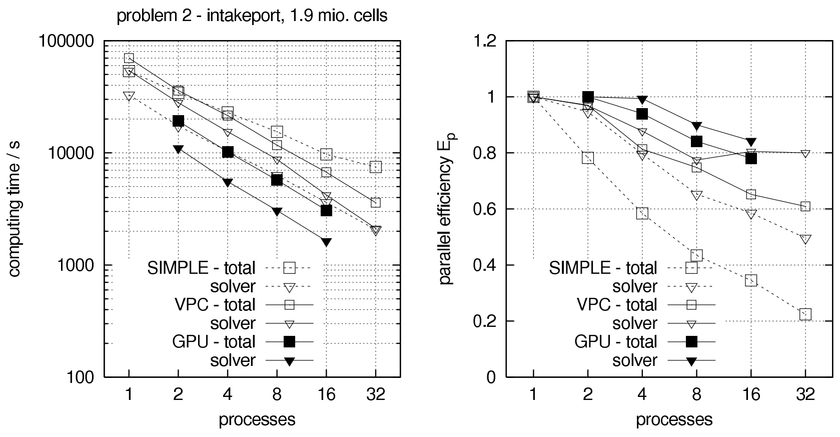


FIGURE 8. Computing times (left) and parallel efficiency (right) for the entire simulations with different algorithms for problem 2. The GPU calculation with one process was not possible due to insufficient on-board memory; the baseline of E_p for the GPU is therefore the calculation with two parallel processes.

10 shows that the reduction of the solution phase of the AMG solver, i.e. the part that has been accelerated on GPUs, runs in fact about eight times faster. The remaining computing time is essentially the time spent in operations that are not accelerated.

5. CONCLUSIONS AND OUTLOOK

We have presented an implementation of an AMG based linear solver for systems representing the discretisation of coupled differential equations. The implementation is modular in a way that it can be integrated with minimal effort into existing solver implementations. With regard to CFD codes, this allows to accelerate the solution of the linear systems of velocity-pressure coupling schemes. These schemes converge faster than conventional segregated schemes, but in conventional CPU implementation, they have the disadvantage of the large computational cost for

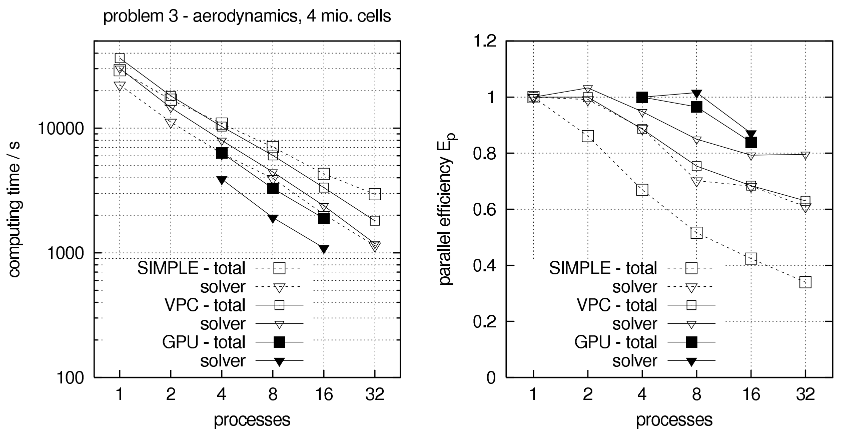


FIGURE 9. Computing times (left) and parallel efficiency (right) for the entire simulations with different algorithms for problem 3. The GPU calculations with less than four parallel processes were not possible due to insufficient on-board memory; the baseline of E_p for the GPU is therefore the calculation with four parallel processes.

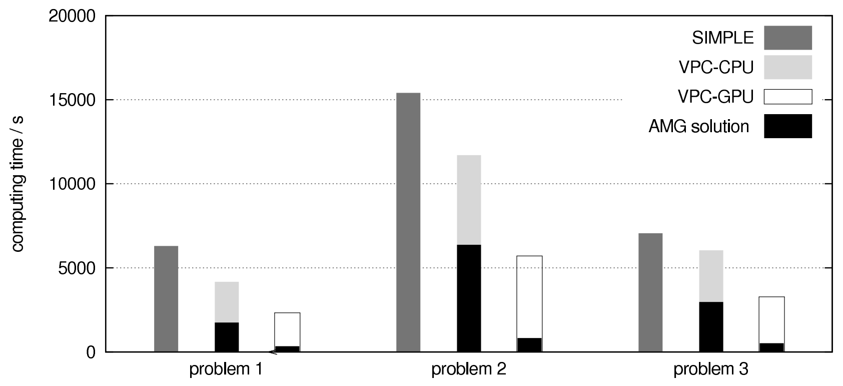


FIGURE 10. Computing times for problems 1 (8 parallel processes), problem 2 (4 parallel processes), and problem 3 (8 parallel processes)

the solution of the linear systems. With GPU acceleration these cost is significantly reduced. We were therefore able to show that the computing times of the velocity-pressure coupling algorithm with GPU acceleration of the linear solver are about 50% faster than the implementation on CPU. Compared to standard segregated SIMPLE (on CPU) the reduction of the computing time is up to 75% for real-world problems of the engineering practice.

ACKNOWLEDGEMENT

Part of the contribution of Maximilian Emans was supported by the government of Upper Austria within the strategic program “Innovatives Oberösterreich 2010plus”. The research of Manfred Liebmann was supported by the Austrian Science Fund FWF project SFB F032 “Mathematical Optimization and Applications in Biomedical Sciences”.

References

[1] Amestoy, P.R., Guermouche, A., L'Excellent, J.Y., Pralet, S.: Hybrid scheduling for the parallel solution of linear systems. Parallel Computing **32**, 136–156 (2006)

[2] Cevahir, A., Nukada, A., Matsuoka, S.: Fast conjugate gradients with multiple GPUs. In: G. Allen, J. Nabrzyski, E. Seidel, G. van Albada, J. Dongarra, P. Sloot (eds.) ICCS2009, vol. 5544, pp. 894–903. Springer Berlin Heidelberg (2009)

[3] Chen, Z., Przekwas, A.: a coupled pressrre-based computational method for incompressible/compressible flows. Journal of Computational Physics **229**, 9150–9165 (2010)

[4] Darwish, M., Saad, T., Hamdan, Z.: Parallelization of an additive multigrid solver. Numerical Heat Transfer **54**(2), 157–184 (2008)

[5] Darwish, M., Sraj, I., Moukalled, F.: A coupled finite volume solver for the solution of incompressible flows on unstructured grids. Journal of Computational Physics **228**, 180–201 (2009)

[6] Emans, M.: Aggregation AMG for distributed systems suffering from large message sizes. In: P. D'Ambra, M. Guarracino, D. Tallia (eds.) EuroPar2010, Part II, *Lecture Notes in Computer Science*, vol. 6272, pp. 89–100. Springer-Verlag Berlin Heidelberg (2010)

[7] Emans, M.: AMG for linear systems in engine flow simulations. In: R. Wyrzykowski, J. Dongarra, K. Karczewski, J. Wasniewski (eds.) PPAM2009, Part II, *Lecture Notes in Computer Science*, vol. 6068, pp. 350–359. Springer-Verlag Berlin Heidelberg (2010)

[8] Emans, M.: Benchmarking aggregation AMG for linear systems in CFD simulations of compressible internal flows. Electronic Transactions on Numerical Analysis **37**, 351–366 (2010)

[9] Emans, M.: Efficient parallel AMG methods for approximate solutions of linear systems in CFD applications. SIAM Journal on Scientific Computing **32**, 2235–2254 (2010)

[10] Emans, M.: Performance of parallel AMG-preconditioners in CFD-codes for weakly compressible flows. Parallel Computing **36**, 326–338 (2010)

[11] Emans, M.: Coarse-grid treatment in parallel AMG for coupled systems in CFD applications. Journal of Computational Science **2**, 365–376 (2011)

[12] Emans, M.: Combining smoother and residual calculation in v-cycle AMG for symmetric problems. In: R. Wyrzykowski, J. Dongarra, K. Karczewski, J. Wasniewski (eds.) PAMM 2011, Part I, *Lecture Notes in Computer Science*, vol. 7203, pp. 651–660. Springer-Verlag Berlin Heidelberg (2012)

[13] Emans, M.: Krylov-accelerated algebraic multigrid for semi-definite and nonsymmetric systems in computational fluid dynamics. Numerical Linear Algebra with Applications **19**, 210–231 (2012)

[14] Falgout, R., Jones, J., Yang, U.: Conceptual interfaces in hypre. Future Generation Computer Systems **22**, 239–251 (2006)

[15] Ferziger, J., Perić, M.: Computational Methods for Fluid Dynamics. Springer Verlag Berin Heidelberg (1996)

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

[17] Goodnight, N., Woolley, C., Lewin, G., Luebke, D., Humphreys, G.: A multigrid solver for boundary value problems using programmable graphics hardware. In: M. Doggett, W. Heidrich, W. Mark, A. Schilling (eds.) Eurographics/SIGGRAPH Workshop on Graphics Hardware 2003, pp. 102–135 (2003)

[18] Haase, G., Liebmann, M., Douglas, C., Plank, G.: A parallel algebraic multigrid solver on graphical processing unit. In: W. Zhang, Z. Chen, C. Douglas, W. Tong (eds.) High Performance Computing and Applications 2010, *Lecture Notes in Computer Science*, vol. 5938, pp. 38–47. Springer-Verlag Berlin Heidelberg (2010)

[19] Jones, W., Launder, B.: The prediction of laminarization with a two-equation model of turbulence. International Journal of Heat and Mass Transfer **15**, 301–314 (1972)

[20] Karypis, G., Kumar, V.: A fast and high quality multilevel scheme for partitioning irregular graphs. SIAM Journal on Scientific Computing **20**, 359–392 (1998)

[21] Konstantinidis, E., Cotronis, Y.: Accelerating the red/black SOR method using GPUs with CUDA. In: R. Wyrzykowski, J. Dongarra, K. Karczewski, J. Wasniewski (eds.) PPAM 2011, Part I, *Lecture Notes in Computer Science*, vol. 7203, pp. 589–598. Springer-Verlag Berlin Heidelberg (2012)

[22] Krüger, J., Westermann, R.: Linear algebra operators for GPU implementation of numerical algorithms. ACM Trans. Graphics **22**(3), 908–916 (2003)

[23] Liebmann, M.: Efficient PDF solvers on modern hardware with applications in medical and technical sciences. Ph.D. thesis, University of Graz (2009)

[24] Notay, Y.: An aggregation-based algebraic multigrid method. Electronic Transactions on Numerical Analysis **37**, 123–146 (2010)

[25] Patankar, S., Spalding, D.: A calculation procedure for heat, mass and momentum transfer in three-dimensional parabolic flows. International Journal Heat Mass Transfer **15**, 1787–1806 (1972)

[26] Rhie, C., Chow, W.: Numerical study of the turbulent flow past an airfoil with trailing edge separation. AIAA Journal **21**(11), 1525–1532 (1983)

[27] Saad, Y.: A flexible inner-outer preconditioned GMRES algorithm. SIAM Journal on Scientific Computing **14**, 461–469 (1993)

[28] Vaněk, P., Mandel, J., Brezina, M.: Algebraic multigrid by smoothed aggregation for second and fourth order elliptic problems. Computing 56 pp. 179–196 (1996)