Sources of Parallel Inefficiency for Incompressible CFD Simulations

Sven H.M. Buijssen[1,2] and Stefan Turek[2]

[1] University of Heidelberg, Institute of Applied Mathematics,
Interdisciplinary Center for Scientific Computing (IWR),
INF 294, 69120 Heidelberg, Germany
[2] University of Dortmund, Institute for Applied Mathematics and Numerics,
Vogelpothsweg 87, 44227 Dortmund, Germany

Abstract. Parallel multigrid methods are very prominent tools for solving huge systems of (non-)linear equations arising from the discretisation of PDEs, as for instance in *Computational Fluid Dynamics* (CFD). The superiority of multigrid methods in regard of numerical complexity mainly stands and falls with the smoothing algorithms ('smoother') used. Since the inherent highly recursive character of many global smoothers (SOR, ILU) often impedes a direct parallelisation, the application of block smoothers is an alternative. However, due to the weakened recursive character, the resulting parallel efficiency may decrease in comparison to the sequential performance, due to a weaker total numerical efficiency. Within this paper, we show the consequences of such a strategy for the resulting total efficiency if incorporated into a parallel CFD solver for 3D incompressible flow. Moreover, we compare this parallel version with the related optimised sequential code in FeatFlow and we analyse the numerical losses of parallel efficiency due to communication costs, numerical efficiency and finally the choice of programming language (C++ vs. F77). Altogether, we obtain quite surprising, but more realistic estimates for the total efficiency of such a parallel CFD tool in comparison to the related 'optimal' sequential version.

1 Numerical and Algorithmic Approach

A parallel 3D code for the solution of the incompressible nonstationary Navier-Stokes equations

$$\mathbf{u}_t - \nu\Delta\mathbf{u} + (\mathbf{u}\cdot\nabla)\mathbf{u} + \nabla p = \mathbf{f}, \quad \nabla\cdot\mathbf{u} = 0 \tag{1}$$

has been developed. This code is an adaptation of the existing sequential FeatFlow solver (see `www.featflow.de`). For a detailed description of the numerical methods applied see [1,3]. Here we restrict ourselves to a very brief summary of the mathematical background. Equation (1) is discretised separately in space and time. First, it is discretised in time by one of the usual second order methods known from the treatment of ordinary differential equations (Fractional-Step-θ-scheme, Crank-Nicolson-scheme). Space discretisation is performed by applying

B. Monien and R. Feldmann (Eds.): Euro-Par 2002, LNCS 2400, pp. 701–704.

a special finite element approach using the non-conforming $\tilde{Q}_1/Q_0$ spaces (non-parametric version). The convective term is stabilised by applying an upwind scheme (weighted Samarskij upwind). Adaptive time stepping for this implicit approach is realised by estimating the local truncation error. Consequently, solutions at different time steps are compared. Within each time step the coupled problem is split into scalar subproblems using the Discrete Projection method. We obtain definite problems in $\mathbf{u}$ (Burgers equations) as well as in p (Pressure-Poisson problems). Then we treat the nonlinear problems in $\mathbf{u}$ by a fixed point defect correction method, the linearised nonsymmetric subproblems are solved with multigrid. For the ill-conditioned linear problems in p a preconditioned conjugate gradient method is applied. As preconditioner, multiplicative as well as additive multigrid (using Jacobi/SOR/ILU smoothers) has been implemented.

In order to parallelise the multigrid method the coarse mesh is split into parallel blocks by a graph-oriented partitioning tool (Metis, PARTY). Subsequently, each block is uniformly refined. Consistency with the sequential algorithm (MV application, grid transfer) is guaranteed through local communication between at most two parallel blocks (this is possible because of the face-oriented $\tilde{Q}_1/Q_0$ ansatz). The inherent recursive character of global smoothers impedes a direct parallelisation. Therefore, the global smoothing is replaced by smoothing within each parallel block only (block smoothers). To minimise the communication overhead for solving the coarse grid problem, it is treated on a single processor with an optimised sequential algorithm. The costs is two global communications (setting up the right side and propagation of the solution vector).

The code has been tested [1] for many configurations including standard benchmarks like Lid-Driven-Cavity and "DFG-Benchmark" [2] as well as some problems with industrial background: computation of drag values on model car surfaces (automotive industry), simulation of molten steel being poured into a mould (steel industry). Hexahedral meshes with aspect ratios up to 500 and problems with 100 million degrees of freedom in space and up to several thousand time steps have been successfully handled. Examples are presented in the talk.

2 Comparison of Sequential vs. Parallel Implementation

As mentioned earlier, the difference between sequential and parallel implementation is the way how smoothing operations are performed. If invoked on more than one node the parallel smoothers work on subdomains that differ from the one the sequential algorithm uses such that, besides communication costs, the number of multigrid sweeps increases with increasing number of parallel processes. The second major difference is the programming language. The sequential implementation has been done in F77, the parallel in C++. These two aspects have to be taken into account when comparing run times. Table 1 shows for a given problem the run times of both implementations. One notices that there is a significant loss associated with stepping from the sequential CFD solver to a parallel one and from F77 to C++. On the very same architecture 4 nodes are necessary to match the sequential F77 run time. If using a parallel supercomputer whose

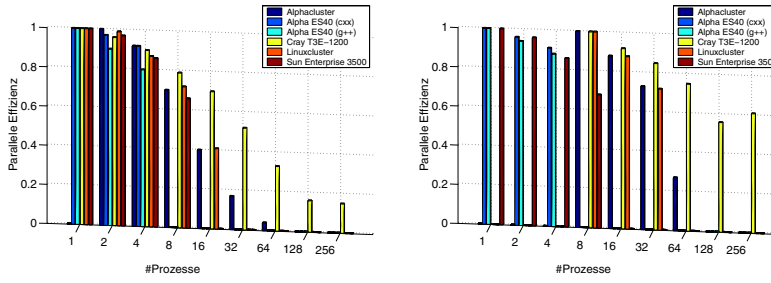
Table 1. Run time comparison for nonstationary DFG-Benchmark 3D-2Z [2] using sequential (F77) and parallel implementation (C++).

implementation	architecture	d.o.f.		cpu time
		space	time	[min]
sequential	Alpha ES40	5,375,872	1,455	2086
parallel (1 node)	Alpha ES40	5,375,872	1,446	6921
parallel (4 nodes)	Alpha ES40	5,375,872	1,466	2151
parallel (4 nodes)	Cray T3E-1200	5,375,872	1,466	4620
parallel (32 nodes)	Cray T3E-1200	5,375,872	1,522	831
parallel (70 nodes)	Cray T3E-1200	5,375,872	1,502	615
parallel (130 nodes)	Cray T3E-1200	5,375,872	1,514	688

single nodes have approximately half the performance of a workstation node, the sequential run time can of course be beaten, but only using brute compute power. Several reasons play a role: the parallel implementation uses a more abstract programming language and is less close to hardware, the optimisation skills of compilers probably differ. The main question, however, is: How good/bad is the total efficiency of the parallel implementation? What causes the run time losses observed? Which influence has the different (blockwise) smoothing?

3 Examination of Parallel Efficiency

In order to explain the behaviour of the parallel implementation seen in Table 1 we primarily focused on its scalability. For a mesh with small aspect ratios (AR ≈ 3) a medium and a big size problem were studied for different platforms and numbers of processes. Figure 1 shows the measured parallel efficiencies. It can be observed that parallel efficiency is not too bad if the problem size is sufficiently large (Figure 1 (b)) and if the parallel infrastructure is well-designed (as on a



(a) 1.3 million d.o.f. in space (b) 10.6 million d.o.f. in space

Fig. 1. Parallel efficiency for a problem with 1.3 resp. 10.6 million d.o.f. in space on a mesh with small aspect ratios. (Platforms: Alphacluster ALiCE Wuppertal, Alpha ES40 with cxx/g++ compiler, Cray T3E-1200 Jülich, Linuxcluster PIII 650 MHz Heidelberg, Sun Enterprise 3500 Dortmund)

Cray T3E). The performance on multi-processor workstations and clusters is worse. This is clearly due to increasing communication loss on these platforms. But besides communication loss (which can be expected) there is another effect playing an important role as soon as the number of parallel processes increases: the number of iterations needed to solve the Pressure-Poisson problems increases by a factor of 3 if stepping from 1 to 256 processes.[1] This means the amount of time the program spends solving the Pressure Poisson problems increases from 9 to 33 percent. Without this effect, parallel efficiency for the big problem using 256 processes on a Cray T3E-1200 would be 0.72 instead of nearly 0.60.

If a mesh with worse aspect ratios (AR ≈ 20) is probed, the problem becomes much more obvious. Using such an anisotropic mesh and comparing two runs with 1 and 64 processes, respectively, the mean number of iterations needed to solve the (elliptic) Pressure Poisson problems increases by a factor of 4-5: More than half of the run time is now spent solving these subproblems. Consequently, parallel efficiency regresses even more. Additionally, solving Pressure Poisson problems needs more communication than any other part of the program. Thus, the increasing mean number of iterations gives us additional communication loss.

4 Conclusions

The detailed examinations in [1] show that our "standard" parallel version of an optimised sequential 3D-CFD solver has (at least) three sources of parallel inefficiency: Besides the obvious overhead due to inter-processor communication, the change from F77 to C++ compilers is a factor of 3.[2] However, the biggest loss is due to the weakened numerical efficiency since only blockwise smoothers can be applied. Consequently, the number of multigrid cycles strongly depends on the anisotropic details in the computational mesh and the number of parallel processors. As a conclusion, for many realistic configurations, more than 10 processors are needed to beat the optimised sequential version in FeatFlow. Thus, new and improved numerical and algorithmic techniques have to be developed to exploit the potential of recent parallel supercomputers and of modern Mathematics at the same time (see [3] for a discussion).

References

[1] Sven H.M. Buijssen. Numerische Analyse eines parallelen 3-D-Navier-Stokes-Lösers. Diploma Thesis, Universität Heidelberg, 2002.
[2] M. Schäfer and S. Turek. Benchmark computations of laminar flow around cylinder. In E. H. Hirschel, editor, *Notes on Numerical Fluid Mechanics*, volume 52, pages 547–566, Wiesbaden, 1996. Vieweg.
[3] Stefan Turek. *Efficient Solvers for Incompressible Flow Problems - An Algorithmic and Computational Approach.* Springer-Verlag, Berlin Heidelberg, 1999.

[1] Replacing ILU smoother by SOR results in less degeneration, but longer run time. Jacobi smoothing is at no time competitive, it represents an upper run time bound.
[2] PCs and Sun systems were tested, too; similar behaviour was observed.