Industrial mixed OpenMP / Mpi CFD-application for calculations of free-surface flows

A. Kneer[a], E. Schreck[b], M. Hebenstreit[c], A. Göszler[c]

[a] Battelle Ingenieurtechnik GmbH, Eschborn, Germany

[b] ICCM Institute of Computational Continuum Mechanics GmbH, Hamburg, Germany

[c] Technikum Joanneum GmbH, Fachhochschule of Graz, Austria

Abstract

For most industrial companies, parallel methods have become standard for CFD calculations as a means to speed up design and development work. During the European Commission-funded Europort project, a number of commercial CFD codes were parallized under MPI. Most ports were based on domain decomposition, using MPI for all communication. Thus the resulting parallel applications could easily be run on distributed memory machines and simple workstation or PC-clusters (BEOwulf). Nevertheless, the price paid for parallelizing a numerical method using message passing paradigm is high, because parts of the code have to be completely restructured and the IO must be reorganized. Since this work cannot be done automatically, MPI parallel code development is expensive and time consuming.

OpenMp, a new programming model for shared memory parallel architectures is becoming more and more popular. Within the EU-funded POST project, the CFD application Comet has been parallized using OpenMP; the work was assisted by a tool, to reduce the effort to create the OpenMP code. The OpenMp / MPI mixed code has been benchmarked on different platforms. OpenMP is used on a node and MPI between the nodes using the domain decomposition approach. The results, which will be described here, show that OpenMP combined with MPI is an interesting standard for practical use in industry, where many MPI-based parallel codes are already available. We describe our work with Comet, showing different parallelization strategies considered, and benchmarks based on them. We will also show how assistance tools can help in parallelization with OpenMP.

Keywords: fluid dynamics, cfd, numeric, domain decomposition, program analysis, parallel, parallelisation, FORTRAN, program transformations, OpenMP

1 Introduction

Multiphase flows like flows with free surface cover a wide range of applications from the sloshing of fuel in the tank during a car crash to the flow field in a chemical reactor. There have been significant advances in the science and technology of multiphase flows in the past few years because of enhanced computational and experimental capabilities. Because of the wide variety and importance of multiphase flows in industrial processes, in the last few years computational fluid dynamics (CFD) became more and more a standard tool in the industry. CFD-calculations are used by engineers from the design phase to the final optimization phase of cars, chemical reactors and various flow devices.

The physics of fluid flow with free surface is characterized by complicated time dependent flow fields which is obvious for situations like the flow in a tank of a car during a crash or complex chemical mixing procedures. Especially in the case of a crash the acceleration of the fluid is enormous, so that the calculation of such processes requires very small time steps to calculate the free surface with the required precision. The hydrostatic hight of the liquid at least defines the pressure load on the wall of the tank. Two-dimensional calculations based on the Finite-Volume method using unstructured grids needs up to twenty hours of CPU-time on a one processor machine. Fine meshes and small timesteps are necessary to resolve the physics. Therefore, such simulations requires considerable CPUtime. Threedimensional time dependent calculations of such flows increase the time for calculations by one order of magnitude. In the short time schedule of product development this is not acceptable.

Because of these requirements most of the commercial CFD-tools have been parallelized using PVM or MPI using domain decomposition technique. A high number of publications show the enormous success in reducing the simulation time by parallel methods ([8],[11],[12], [13]). Furthermore, this new feature became quite popular and was soon of great interest for the industrial market. Tools have been implemented in order to assist the code developpers parallelizing there applications using MPI [2]. Performance measurement tools completed the set of tools available on the market now. The advantage of the standard MPI is the fact that it can be used on a wide range of hardware platforms. MPI is available for distributed memory systems like workstation clusters, PC-clusters and massivly parallel architectures like the Cray T3E as well as for shared memory systems.

A lot of benchmarks show the benefit of parallel CFD-codes in reducing the simulation time. Nevertheless parallelizing codes under MPI needs a complete

restructuring of parts of the code which is very time intensive. Typically, even in parallelized CFD packages there are still some constraints regarding the partitioning in subdomains of advanced features like sliding grids. Furthermore, the change of the number of processors during a calculation requires a re-formating of the restartfiles to get the code restarted. Such constraints are not existing for shared memory parallelized packages. May be these things are not so important in the parallel research context but during the day by day work with parallel numerical methods it is a non negligible factor.

Most of major hardware vendors market cost-effective parallel systems which consists of a modest numbers of processors and shared memory. Also large scale platforms are deployed which are a combination of distributed memory and shared memory systems connected via a high speed network. At the other end of the performancescale multiprocessor workstations and PCs with shared memory are increasingly being deployed. OpenMP which has originally being developed for SMPs is also available on ccNUMA systems. The wide spectrum of Dual-pentium architectures up to ccNUMAs like a SGI Origin can be used for parallel calculations based on OpenMP. OpenMP as a high-level programming model can easily integrated in applications if there is no data dependency inside the loop-construction. Tools are available on the market to support the engineer in parallelizing his application under OpenMP. One of such a tool, named Foresys [3] has been extended to OpenMP within the EU-funded project POST (Programming with the OpenMP Standard). Opposite to MPI the parallelization under OpenMP can particularily performed automatically by using such tools. In the area of CFD-Calculations OpenMP is an very interesting issue to be used by CFD-codes to speed-up the applications. Therefore, the commercial CFD-code Comet which is already parallelized under MPI (domain decomposition), has additionally been parallelized under OpenMP. Benchmarks have been performed with the parallelized application in order to compare the efficiencies of the pure OpenMP code to the MPI-version. Finally mixed calculations on several platforms have been performed to get informations about the hardware dependencies of the code. The results of those benchmarks show, that OpenMP, which has a lot of advantages in the practical use for industrial issues, doesn't scale as well as with MPI. However there is a great potential which may be achieved by a concerted action of hardware providers, providers of compilers and finally the end users from industry.

2 Description of the CFD Method

Comet is a multi-purpose CFD-Software for the solution of continuum mechanics problems (both fluid and solid mechanics). It is a self-contained package incorporating mathematical models of a wide range of thermo-fluids and solids phe-

nomena, powerful solvers enabling full complex-geometry capabilities, highly efficient and stable set of numerical solution algorithms, flexible pre-processing and post-processing facilities tailored for both expert and novice users.

A unique feature of Comet is the possibility to use blockwise-generated grids, where the grids do not have to match at block interfaces (only the interface surface has to be the same). Every polygon common to two CVs is treated as a cell face; CVs next to an interface may thus have many faces. These faces are treated in the same way as other cell faces – implicitly and conservatively (the interface is not a boundary!). This feature enables easier creation of better grids; also, for big problems, the grid can be generated piece-by-piece (even by different people on different computers).

Segregated solution algorithms are preferred to block solutions because of their small memory demands (only one transport equation is solved at a time). Because of the non-linear nature of the governing equations, an iteration for inter-equation coupling becomes a part of a non-linear iteration, such that for most of the practically relevant situations, no extra cost is to be payed in terms of increasing computer time.

The linearized algebraic equations are solved using efficient conjugate gradient based solvers (e.g. ICCG, CGSTAB). Additional improvements in the algorithm are achieved by employing 'full multigrid' (FMG) algorithm. FMG should deliver in a typical situation computer times which scale linearly with the number of discretisation elements.

A generation of progressively finer grids can be aided with a number of error estimators, leading towards an optimal numerical grid which gives the maximum accuracy for a given number of discretisation elements. The main features are listed below:

- finite-volume
- polyhedral control volumes
- cell-wise local grid refinement
- numerical grids made of a number of non-matching blocks
- discretization schemes:
  - midpoint rule integration
  - linear interpolation
  - second order discretisation of diffusive fluxes
  - convective fluxes discretised using upwind, linear upwind, central, MINMOD or HRIC discretization

– fully implicit: 3 time levels (2nd order), Euler (1st order)

Interface-capturing method enables computation of free-surface flows with an arbitrary deformation of the interface between the two fluids. Surface tension effects can also be taken into account. The flow of both fluids is computed. Figure 1 shows a typical geometry of a mixer. The liquid inside the vessel has a free surface. The liquid is mixed by rotating baffles. At the same time liquid is let in from the top.

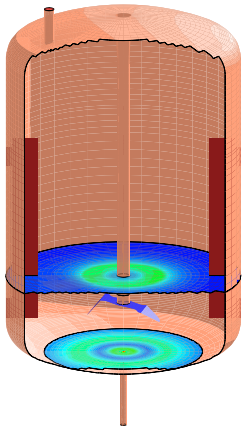

Figure 1: Typical geometry of a mixer

The numerical method relies on the finite-volume discretization of the governing fluid mechanic equations on a body fitted, unstructured computational mesh. With cell oriented local refinement even exceptionally complicated geometries can be treated in detail, keeping the number of grid points to a minimum. Popular grid generators can be used to create computational meshes. Import of grids is facilitated by the unstructured data arrangement.

3 Parallelization

Large problems can be solved using Comet on parallel computers or on workstation or PC clusters. Parallelization is based on domain decomposition and message passing. This provides flexibility to port Comet parallel version to almost any parallel configuration.

3.1 Domain Decomposition

The unstructured grid is partitioned using a standard graph partitioner [4]. The parallel version of Comet is implemented according to the Single Programme Muliple Data (SPMD) programming style using MPI or PVM message passing library for communication. Each partition is surrounded by an auxiliary layer of cells originating from neighbouring partitions, in order to accomodate halo data which are needed for the solution of the current partition.

3.2 OpenMP

In the shared memory programming model grid partitioning is equivalent to partitioning the loops running over the whole grid. This aim can by acchieved by applying the PARALLEL DO directive to the corresponding do-loops. An analysis of the most time consuming routines in Comet shows that there are mainly occuring three different loop structures:

- The first kind of loops is running over all cells with the cell index as loop index.
- In the second type the cell face index is the loop index so the loop is covering all control volume faces of the computational domain.
- The third loops structure is a nested loop where the inner loop visits all neighbours of a cell and the outer loop covers all cells.

Loop over cells Between the iterations of the cell loops there are no data dependencies. Therefore, the parallelizing was possible in a straight forward manner. Figure 2 shows a typical loop over cells which has been parallelized by Foresys. These kind of loops are very common and need to be parallelized several times in several units.

```
!$OMP PARALLEL DO
      do ip=nstc,nenc
        ps(ip)=q(ip)+bet*ps(ip)
      end do
!$OMP END PARALLEL DO
```

Figure 2: Loop over cell

Loops over faces An example for a parallel face oriented loop is given below in Figure 3. This kind of loop can not automatically be parallelized. For that loop structure a reorganisation of the loop is necessary: The parallelization strategy is to rearrange the order in which the contributions of the faces are calculated. This new order is stored in the array `ifacsq`. Also, the loop has to be transformed in two nested loops where the inner loop `do iom=ivecst(i), ivecen(i)` is running over the cell range and the outer loop `do i=1, maxnnb` is running over the neighbours of the cells.

```
!$OMP PARALLEL PRIVATE(I)
      do i=1,maxnnb
!$OMP DO PRIVATE(J,IP1,IP2)
        do iom=ivecst(i), ivecen(i)
          j=ifacsq(iom)
          ip1=lf(1,j)
          ip2=lf(2,j)
          wrk(ip1)=wrk(ip1)+af(1,j)*sv(ip2)
          wrk(ip2)=wrk(ip2)+af(2,j)*sv(ip1)
        end do
!$OMP END DO
      end do
!$OMP END PARALLEL
```

Figure 3: Loop over faces

Loop over cells and faces A third type of loops occurs in the sparse LU factorization preconditioner of the CG methods: the forward and backward substitution. There are only two loops in the code with this structure but as one can see from the profiling there is lot of computing time spent in these two loops. Therefore, the parallelization of these loops is necessary. These loops could be parallelized by Foresys in a straightforward manner. But the efficiency of the resulting code is very poor. The reason is the large overhead for the language constructs in OpenMP. If OpenMP constructs are used in the innermost loop a huge increase of computing time can be observed. To create an efficient OpenMP version the only possibility is the usage of the OpenMP directives for outermost loops or at subroutine level. Also, a wavefront reordering approach of these loops requires to much overhead due to OpenMP. Therefore, a complete reorganization of these strongly recursive loops is only possible at cost of numerical performance to gain speedup. A typical way are multi color methods which are used in the OpenMP version of Comet .

Tool assessment Several tools for parallelization are available on the market for a long time, mainly to support the software engineers parallelizing their application. As part of the POST project OpenMP code creation tools are being developed. The commercial reverse engineering software product Foresys [3] from Simulog is at the heart of this development. During the application development, extensions of the basic tool Foresys have been found necessary in order to be able to guide the user in parallelizing his application and support him by a transformation tool to get semi-automatically loopwise OpenMP directives inserted. Foresys, a commercial Fortran source code analysis and restructuring system, which supports also data-dependency analysis, has been extended to assist in the creation of OpenMP source. Foresys has been used for the CFD-code Comet for data-dependency analysis and for creation of OpenMP source where possible. Typical CFD-codes have only a few subroutines where most of the CPU-time is spent during a calculation. These subroutines need to be parallelized efficiently to achieve good speed-up results. As described in the previous sections the loops inside those specific subroutines have a complicated structure with a number of data dependencies which can not easily be transformed to OpenMP. Therfore the idea came up to couple the basic tool Foresys with a kind of expert tool called TSF [1] and AGM (Automatic Guidance Tool). TSF is based on ”scripts ”which can easily be adopted to specific requirements. Therefore code specific structures can then be described by those scripts and transformed to a parallelizable code structure. It is clear that a wide script-database is necessary to fulfill the requirements concerning various codes. Nevertheless this database can be fitted and extended.

4 Parallel Performance

In order to provide a wide spectrum of hardware several hardware architectures have been benchmarked using the CFD-code Comet. A test case of 100.000 Control Volumes has been created and been ported to following platforms:

- Small Linux-cluster, 4 Pentium 800MHz, Dual Processor Board
- SGI, SGI Power Challenge, 195 MHz R10K processors
- SGI, workstation cluster, 8 dual processor worstations SGI Octane (R10000, 225Mhz)
- SGI, SGI Origin 2000, 48 processors R10000 195 MHz

To be able to get only the parallel efficiencies the number of outer and inner iterations have been fixed. Therfore, in all cases the same number of iterations

were performed. Furthermore, the Comet code has been benchmarked in the following way:

- Only OpenMP
- Only MPI
- Mixed OpenMP/Mpi, different MPI, OpenMP processes

Power Challenge Concerning a real SMP machine Comet has been benchmarked on a Power Challenge which is not the newest architecture. However the results obtained are good. Table 1 shows the results achieved. The code scales up to 16 processors quite well. But it can be seen that there is a loss of time from one to two threads.

OpenMp Threads	Time [s]	Efficency [%]
1	346	-
2	195	89
4	92.8	93
8	48.5	89
16	27.2	80

Table 1: Measured Efficiencies for a Power Challenge (OpenMP)

OpenMp Threads	Time [s]	Efficiency [%]
1	229	-
2	147.7	77.5
4	74.4	76.9
8	37.3	76.3
16	22.9	62.5

Table 2: Measured Efficencies for a Origin (OpenMP)

Origin This is much better on a Origin. As it can be seen the code scales very good up to 8 Processors. But increasing the number of processors to 16, the efficiency is smaller compared to that one achieved on the Power Challenge. To get an impression how MPI works on such a architecture the CFD-code Comet has been run with pure MPI. Table 3 shows the efficiencies achieved running the code

OpenMp Threads	MPI processes	Time [s]	Efficiency [%]
1	1	229	-
1	2	130.2	88.0
1	4	44.5	128.6
1	8	22.4	127.8
1	16	14.4	99.3

Table 3: Measured Efficiencies for a Origin (MPI)

with the domain decomposition method using MPI. It turns out that computing time and efficiencies using MPI are better compared to the OpenMP values. The obvious reason for this behaviour is the considerable overhead of the OpenMP language constructs [6], [7]. The reason for the super-linear speedup for the MPI case (Tab. 3) is a cache effect which increases the processor performance. A small test case shows that for the grid sizes used in this benchmark the floating point performance is more then 40% larger for the 4 processor partitioning.

Linux-cluster Benchmarks have also been performed on a small Linux-cluster which is used at Battelle as an experimental cluster for numerical calculations. This Linux-cluster consists of 12 PCs. Four nodes are dual pentium III machines with 800Mhz. The PCs are connected via 100Mbit Ethernet. Tables 4 and 5 show the results achieved on that cluster. The code does not scale very good from one OpenMP thread to the second. Among other reasons this is due to the memory contention on the dual pentium boards when running applications with large data sets which fits not to the cache. But overall it scales well up to eight processors two OpenMP threads on each of the four PCs and MPI used for the overall communication from PC to PC. By using MPI processes instead of OpenMP threads the code scales nearly in the same way (table 5).

OpenMp Threads	Number of processors (MPI)	Time [s]	Efficiency [%]
1	1	156.3	-
2	1	100.7	77.6
2	2	53.8	72.6
2	4	30.6	63.8

Table 4: Measured Efficiencies for a Linux-cluster (OpenMP / MPI)

OpenMp Threads	Number of processors (MPI)	Time [s]	Efficiency [%]
1	1	155.4	-
1	2	102.4	75.8
1	4	54.3	71.5
1	8	29.5	65.8

Table 5: Measured Efficiencies for a Linux-cluster (MPI)

SGI-cluster The fourth hardware be tested is a SGI workstation cluster built of 8 dual processor SGI Octane (R10000, 225Mhz) workstations connected via a 100Mbit Ethernet. Regarding table 6 the speed-up from one thread to the second one using OpenMP an efficiencyiy of 85% is achieved. This is better than on the previously benchmarked Linux machine. Probably because of the better memory bus there is less memory contention. But the performance achieved for a increasing number of nodes each of it providing two OpenMP threads does not scale very well.

OpenMp Threads	Number of processors (MPI)	Time [s]	Efficiency [%]
1	1	215.3	-
2	1	126.7	85
2	2	79.4	79.8
2	4	47.9	66.1
2	8	30.8	51.4

Table 6: Measured Efficiencies for a SGI-cluster (OpenMP / MPI)

5 Conclusions

Industrial needs to get computational results for complex physical problems during one day is essential to benefit from those results. Therefore, parallel computations of fluid flow using domain decomposition technique under MPI is a standard tool which will be also used in future. The benchmark results presented in this paper show that the CFD-technology can benefit from OpenMP. For workstation clusters the efficiencies achieved by using MPI / OpenMP are similar. Most of the smaller and medium sized companies have clusters and it could be expected that clusters consiting of SMP machines will be wide spread in industry in future.

As opposed to proprietary shared memory programming models OpenMP is available on several shared memory platforms. Therefore, it is possible to offer

commercial CFD packages on a lot of shared memory systems but the effort for parallelizing the application has to be made only once. It turns out that a shared memory parallel version is more comfortable for the users. There is no grid partitioning necessary, and no constraints which results from this partitioning has to be taken into account:

- no constraints in the use of sliding interfaces
- Restart of runs are not dependent on the number of processors previously used
- Coupled problems are easier to handle (Lagrange and Euler)
- parallel user codings are not necessary

Results achieved on SMPs or the ORIGIN show that the code is scaling well upto 16 processors. However if there are a higher number of processors involved the efficiency would decrease. But in the industrial scale 16 upto a maximum of 32 processors in the area of CFD-technology are usual. In general the results show the benefit gained by using OpenMP for a wide spectrum of hardware architectures. Also, the benchmark results shows that there is still room for improvement on the OpenMP in reduzing the overhead as well as in the applications for increasing the efficieny.

References

[1] F. Bodin, Y. Mevel and R. Quiniou. *A user level program transformation tool*, Proc. Int. Conference on Supercomputing, 1998

[2] B. Chapman, F. Bodin, L. Hill, J. Merlin, G. Viland and F. Wollenweber. *FITS—A Light-Weight Integrated Programming Environment*. To appear in Proc. Europar '99, Toulouse, 1999

[3] Simulog SA, *FORESYS, FORtran Engineering SYStem, Reference Manual V1.5*, Simulog, Guyancourt, France, 1996

[4] G. Karypis and V. Kumar. *A fast and high quality multilevel scheme for partitioning irregular graphs* Technical Report 95-035, University of Minnesota, 1995.

[5] M. Wolfe. *Loop Skewing: The Wavefront Method Revisited.* Intl. Jnl. Parallel Programming 15,4, pp279-294,August 1986.

[6] J.M. Bull *Measuring Synchronisation and Scheduling Overheads in OpenMP* First European Workshop on OpenMP - EWOMP'99, Lund, September 30 - October 1, 1999.

[7] R. Berrendorf and G. Nieken *Performance Characteristics for OpenMP Constructs an different Parallel Computer Architectures* First European Workshop on OpenMP - EWOMP'99, Lund, September 30 - October 1, 1999.

[8] S. Muzaferija and V. Seidl and H. Fogt and A. Kneer *Parallel Multidimensional Calculation of Steady-State and Time-Dependent Flow with Combustion* Euro-Par97, Springer, 1997.

[9] H. Fogt and M. Perić and A. Kneer and V. Seidl *Entwicklung eines parallelen Berechnungsverfahrens für teilchenbeladene Strömungen mit Verbrennung* Statustagung HPSC-97, G. Wolf and R. Krahl, DLR-PT-IT, 1997.

[10] H. Fogt and A. Kneer and V. Seidl *Three-Dimensional Parallel Numerical Calculation of pressure Loads Generated by Hydrogen Deflagration in Complex Geometry* Proceedings of ICONE5, Paper 2614, ASME, 1997

[11] J. H. Ferziger and M. Perić *Computational Methods for Fluid Dynamics* Springer, 1996.

[12] Meurant *Domain Decomposition Methods for Solving Large Sparse Linear Systems* ATO ASI Series F, 185-206, 1991

[13] E. Schreck and M. Perić *Computation of fluid flow with a parallel multigrid solver*, International Journal for Numerical Methods in Fluids, 303-327, 1993.