JOURNAL OF AEROSPACE COMPUTING, INFORMATION, AND COMMUNICATION
Vol. 2, February 2005

Evaluation of Parallel Performance of an Unstructured CFD Code on PC-Clusters

Mattias Sillén*
Saab Aerosystems, Linköping, Sweden

This paper presents an evaluation of parallel performance of a computational fluid dynamics solver on PC-clusters. The solver uses unstructured grids and is parallelized by domain decomposition using MPI as the communication library. The influence of node configuration and type of cluster network on the parallel performance is investigated for typical applications from an aerospace design environment. The high performance network becomes significant when the system scales up. More important is the node configuration where memory bandwidth saturation limits the performance on dual nodes.

Nomenclature

E_P	=	parallel efficiency, S_P /P
M_∞	=	free stream Mach number
N_{oper}	=	number of operations on the finest grid level
N_{comm}	=	number of bytes communicated on finest grid level
$N_{connodes}$	=	number of connecting nodes between partition
n	=	message length, in bytes
$n_{1/2}$	=	message length when latency, λ, and bandwidth-related transfer time, τn, are equal, $\lambda/\tau=\lambda\theta$
P	=	number of processors
S_P	=	parallel speed-up, T(1)/T(P)
T(1), T(P)	=	elapsed time per iteration on 1 and P processors, respectively
t_{comm}	=	time for message transfer, $\lambda+\tau n$
α	=	angle of attack
λ	=	latency, or start up time
θ	=	network bandwidth, $1/\tau$
τ	=	time to transmit 1 byte

I. Introduction

In the aerospace industry today computational fluid dynamics (CFD) plays an important role as a tool for design and analysis. Typical characteristics for the industrial CFD environment are the often complex geometry, the frequency of large problems, the requirement of high fidelity results, the short time schedule for producing results, etc. The various stages in the aerodynamic design process utilize different aspects of the CFD technology. Early stages of the design process involve configuration evaluation and comparisons often performed on simplified

Received 1 October 2004; revision received 13 December 2004; accepted for publication 20 December 2004.
*Research Scientist, Aeronautical Engineering, Member AIAA.

geometries with low order flow modeling approaches but with very short turnaround time requirements. Traditionally the analysis is performed with panel methods but today the use of inviscid (Euler) flow modeling is growing strongly. During later stages in the design process high-fidelity flow modeling methods are applied on detailed geometries for overall and in-depth analysis as well as troubleshooting. Typically are Euler or Reynolds Average Navier-Stokes (RANS)-based 3D methods used at this stage. These methods are computationally expensive. To further improve to confidence of the aerodynamic predictions in the design process advanced flow modeling methods must be deployed at an even earlier stage than today. Early in the design process when the geometry is rapidly changing the allowable time frame to produce a complete flow analysis is no more than a day. To be able to meet this goal the CFD program execution time has to be in the order of minutes to hours. This allows for evaluation of the results and preparation of improved design during the working day. A cost-efficient way to reduce the problem turnaround time is to use parallel computers and efficient numerical algorithms to solve the flow equations. To achieve good parallel performance of a CFD solver the workload has to be evenly distributed among the processors and the communication in-between the processors has to be efficient. In this paper a CFD solver is analyzed regarding parallel implementation, load balancing and impact of parallel platform configuration.

Many codes for solution of the Euler and Navier-Stokes equations in the aerospace industry today are based on unstructured grids using numerical techniques made popular by Jameson et al.[1] in the 80s. The shift towards using unstructured or hybrid grids instead of the previously dominating multi block grids is explained by maturing grid generation and flow solution techniques, especially for viscous flows around complex geometries. Probably the most important factor from an industrial point of view is the turn-around time issue. By using unstructured/hybrid grids instead of structured multi block grids the grid generation task can be significantly reduced in time. The more automated process using unstructured grids also reduces the long learning curve experienced with the structured multi block approach.

The goal of this investigation is to evaluate the performance of Beowulf-class systems in a typical CFD scenario. CFD has experienced an impressive growth since the early 70s when the first simulations of inviscid fluid flow in complex geometries were available.[2] This development has been based on improvements of both numerical algorithms and novel computer architecture. In fact, simulations of large and complex physical systems, such as those required by the aerospace industry, are only possible through efficient exploitation of high-performance parallel computers. This is not only due to the computational cost but also to the large memory requirements. Focusing on the architecture, several trends[3] have made Beowulf-class systems one of the most popular choices for building cost-efficient parallel computers. The Beowulf clusters share some characteristics - they rely on commodity of the shelf (COTS) components. Usually microprocessors (Intel P4, Xeon, etc) are connected through Gigabit Ethernet and Gigabit Ethernet switches with Linux as the operating system but also are networks of Unix workstations (NOW) found. With the broad acceptance of this class of parallel systems more high-end components have entered the market. With processors like Intel Itanium2, AMD Opteron and Apple G5 connected though high-end interconnects as Myrinet, SCI or Infiniband large systems with several thousand processors are outperforming most of the traditional supercomputer technology. Looking at the 24th top 500 list (Ref. 4) from November 2004, six out of ten of the most powerful computers worldwide are Beowulf-clusters.

When designing a Beowulf system for a specific application or a spectrum of applications there are several design choices to be made. The two most important are the compute node configuration and the internal network connecting the nodes. This paper details our experience using PC-based Linux-clusters in a typical industrial aerodynamic analysis environment with emphasis on performance evaluation using an unstructured CFD solver and the impact of different networks and node configurations on the performance. Programming models and implementation aspect to achieve high parallel performance is a wide and active research field. Techniques applicable to CFD solvers and their influence on parallel performance can be found in Ref. 5. One of the larger applications presented using both shared memory and distributed memory approaches with several thousand processors is discussed in Ref. 6. Additional aspects of using parallel CFD codes in industrial applications are found in Refs. 7-9. CFD is, as one of the most computationally demanding disciplines, a driving force behind the development of new computer architectures. The design and evaluation of parallel systems is often based on CFD workloads. One example is the NAS parallel benchmark[10] that tries to mimic the computation and data movement characteristics of large scale CFD application. An evaluation of parallel performance on Beowulf cluster using the NAS-MG kernel benchmark as well as a CFD multi grid solver is found in Ref. 11. In Ref. 12 a CFD code is used to evaluate node configurations and network choices affecting the parallel performance on PC-clusters. The evaluations in Refs. 11 and 12 are, however, limited to a small number of processors and both use structured grids. This paper presents experience from larger PC-clusters and also examines additional aspects influencing the parallel performance. In the next section the governing equations and the numerical implementation are briefly outlined. This is followed by a description of the parallel implementation and the domain decomposition. A description of the

hardware platforms and the performance results form the main part of the paper. At the end of the paper some conclusions are drawn.

II. Numerical Solution Method

The compressible Navier-Stokes equations represent the flow of air around airplanes very well. At least close to most surfaces the flow is turbulent with many different scales in space and time. Because of insufficient memory capacity and computational speed of available computers today, not all scales can be resolved in the direct numerical solution of the Navier-Stokes equations, except for low Reynolds numbers and simple geometries. The influence of turbulence must therefore be modeled. A suitable choice between modeling capability and computational complexity is here the Explicit Algebraic Reynolds Stress Model (EARSM). The present formulation is based on a traditional two-equation model using the concept of effective turbulent viscosity.[13] This results in a numerical behavior similar of the underpinning linear two-equation model with only a minor extra computational overhead. Neglecting the viscous effects in the flow results in the Euler equations. These are a cost-efficient alternative for cases where the viscous effects are small.

The flow equations are solved on unstructured grids of arbitrary elements. The flow equations are discretized in a node-centered finite volume approach. Different element types in the grid are handled by a common edge-based data-structure in the flow solver. The solver uses a node-centered finite-volume technique where the control volumes are formed by a dual grid obtained from the control surface for each edge. The spatial discretization is either central with artificial dissipation or upwind: both approaches are second order accurate. The basic iterative scheme for the equations is a Runge-Kutta algorithm. Local time stepping and implicit residual smoothing accelerate convergence. An agglomeration multi grid algorithm is used to further accelerate the convergence. Within the multi grid cycle a time-step is performed on the fine grid, transferring the solution and the residuals to the next coarser grid level, performing a time-step on the coarse grid level and interpolating the corrections back from the coarse grid level to update the fine grid solution. This process is applied recursively to all coarse grid levels in the sequence. A more comprehensive description of the algorithms is found in Ref. 14. All results below were obtained with the central scheme with artificial dissipation.

III. Parallel Implementation

The parallel implementation is based on domain decomposition. In this approach each processor executes its own copy of the program but operates on a subset of the computational domain. This is often referred to as the single program, multiple data (SPMD) paradigm. For parallel efficiency it is crucial that processors are kept equally busy with local computations and that the overall communication is kept to a minimum and equally distributed between the processors.

In the serial code, the flux balancing process is computed by adding flux contributions from each control surface of a control volume to appropriate nodes. In the parallel implementation, within each processor, flux contributions are calculated in the same way. Due to the cell-centered finite volume discretization ghost points are introduced where the partition boundaries cross edges to compute the fluxes locally, see Fig. 1. Ghost point values are updated from the partition holding their real images at each Runge-Kutta stage. To maintain a complete correspondence between serial and parallel solutions additional entities are also communicated between the domains. This includes edge-based variables as spectral radius, residual smoothing operations and boundary conditions. Global reduction operations and synchronization are handled by one processor and communicated after a complete iteration when also a decision is made to proceed with the iteration process or stop and write the solution. Communication between the processes is implemented using the MPI message-passing library.[15] The communication pattern is predetermined at run-time following the logics of the domain decomposition. Communication is performed by packing data from all boundary points on a given processors to be sent to another processor into a single buffer that is sent as a single message using non-blocking send. This standard approach to communicate between processors has the effect of reducing latency overheads by creating fewer and larger messages.

In the current parallel implementation the same processor operates on all grid levels of a partition. The domain decomposition is performed only on the finest grid level. Control volumes on coarser levels are assigned to the partition that contains the largest part of each individual control volume. This minimizes the communication between processors when changing grid level but may lead to load imbalance on coarser grid levels. An alternative is to perform domain decomposition on each grid level separately. This will increase the number of communication when changing grid level but will guarantee a better load balance also on coarser grids levels.[6]

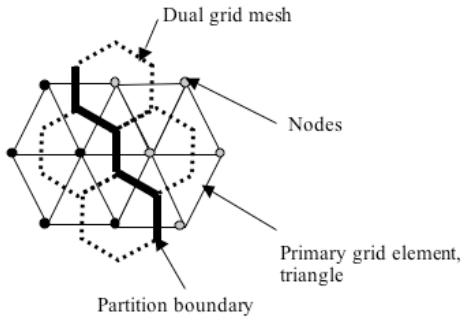


Fig. 1 Layout of primary grid cells, dual grid control volumes and partition boundary.

IV. Domain Decomposition

A key aspect in efficient use of multiprocessor systems is the load balancing. For explicit solvers such as the present CFD solver, the amount of computational work per grid point is roughly constant. A good load balance can therefore be achieved by mapping approximately the same amount of grid points to each processor. The partitioning can be performed using various techniques; in this case the standard graph partitioning program MeTiS[16] is used. Balancing the workload alone is however not sufficient. Communication load must be kept at a low and balanced level, and these objectives are not entirely compatible. This is exemplified here with a tetrahedral grid containing 3 Mpoints partitioned from 2 up to 256 partitions. Using both k-MeTiS and p-MeTiS completely balanced partitions are obtained concerning number of points. Figure 2 shows the mean number of points communicated between partitions. Above 4 partitions this is in favor of the k-MeTiS algorithm that also tries to minimize the number of interface nodes between partitions. It is not clear why the k-MeTiS algorithm fails to consistently deliver a lower number of points to communicate also for 2 and 4 partitions.

Due to the large size of the grids considered in this work, all preprocessing operations must be performed on a large memory machine. This includes the dual grid generation, agglomeration procedure and the partitioning of the various coarse and fine grid levels. This is mainly due to the large memory requirement of these procedures, rather than CPU time requirement, which is small compared to the flow solver. The memory requirement of the preprocessor part equals that of the flow solver with approximately 1.5 kbytes per grid point.

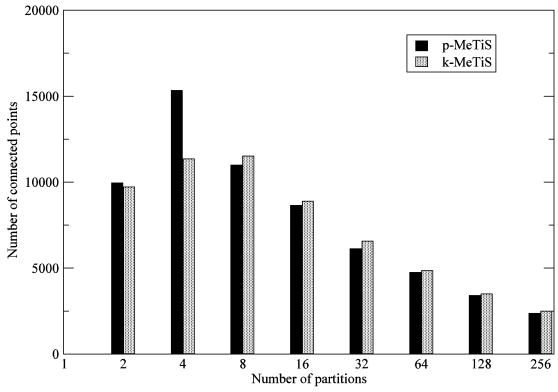


Fig. 2 Mean numbers of points communicated between partitions using different MeTiS algorithms.

V. Performance Evaluation

A. General features of parallel processing

The basic configuration of a PC-cluster is a number of compute nodes connected via an internal network. The node is usually composed of one or two processors and one local memory. This configures a MIMD distributed memory environment on which the natural way of parallelization is that of message passing.

In this computing environment efficient codes rely on factors as

1) the number of processors and the capacity of their local memories,

2) the processors inter-connection,
3) the ratio of computation and communication, and
4) the computational speed of each individual processor.

Besides, performance of distributed memory architectures depend greatly on the network features

1) Topology: how the nodes are connected
2) Latency: time required to initiate the communication
3) Bandwidth: maximum speed of the data transfer

B. Parallel architecture

The computing platforms evaluated in this study belong to two distinct classes of Beowulf systems. The first class is often referred to as capacity clusters with characteristics as single processor nodes and "low-cost" commodity network focusing on applications with few processors or moderate communication requirements. The second class, commonly referred to as capability clusters, is often equipped with dual nodes and high-performance network and addresses the needs from large parallel applications with high bandwidth and low latency requirements.

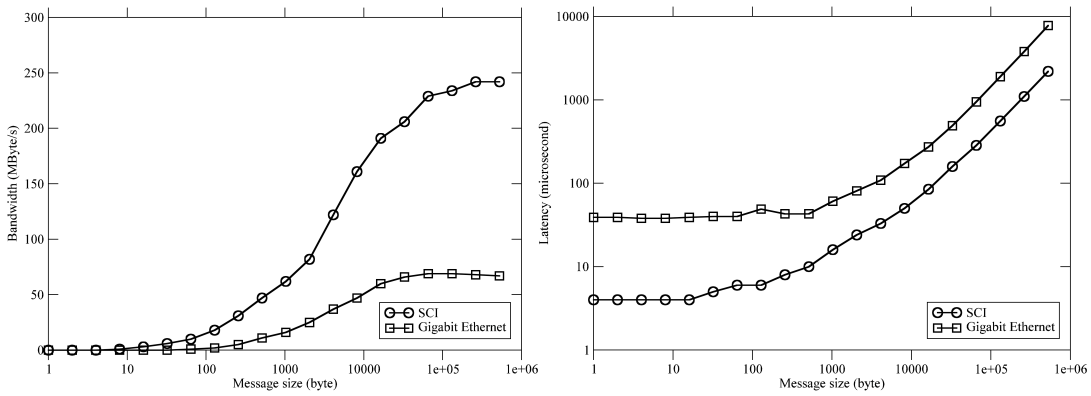
The systems in this study are all based on Intel processors. A contributing factor for choosing Intel processors is the good code performance obtained with the Intel 7.1 Fortran90 compiler. It delivered executable code running 15% more efficient compared to the Portland Group compiler on Intel processors. The smallest system consists of 32 Intel P4 (2.8 GHz) compute nodes with 1 GByte of memory each. The nodes are connected with Gigabit Ethernet with a Gigabit switch with full backplane capacity. This system belongs to the class of low-cost capacity cluster often found at department levels in industry or academia. The two larger systems, belonging to the capability class, are based on a common design principle; dual Xeon nodes with SCI[17] high-speed network. In addition to SCI is also Gigabit alternatively Fast Ethernet available on the systems, mainly for file transfer and non-MPI applications. The SCI network is based on Dolphin Wulfkit network cards with ScaMPI, the MPI communication software from Scali.[18] The network topology is a 2D torus for the 40-node SCI system and a 3D torus for the 200-node system. The clusters with SCI are approximately twice as expensive per processor compared to a pure Gigabit Ethernet cluster. Table 1 summarizes the most important features of the clusters. All systems are designed and assembled by the National Supercomputer Centre[19] (NSC) at Linköping University in Sweden.

Table 1 Main features of evaluated clusters

System	Processor	Network	Nodes	Proc/node	Linpack performance
Stokes	P4, 2.8 GHz	Gigabit Ethernet	32	1	98 GFlops
Maxwell	Xeon, 2.4 GHz	SCI/Gigabit Ethernet	40	2	209 GFlops
Monolith	Xeon, 2.2 GHz	SCI/Fast Ethernet	200	2	1.13 TFlops

C. Network performance

Before discussing the impact of the network on the evaluated CFD solver, it is worthwhile to compare the raw performance achieved by MPI on different network. Especially is the point-to-point performance of interest as it accounts for the greater part of the communication cost in the CFD solver. As a point-to-point benchmark the well-known ping-pong test between two nodes is used, see Fig. 3a, b for results. In this basic benchmark the SCI network delivers 250 Mbyte/s with a short message latency of 5 μs using the ScaMPI implementation of MPI. The Gigabit Ethernet delivers 69 Mbyte/s with a latency of 39 μs with the mpich implementation. The results from the ping-pong test give performance on an idle network. Other competing traffic in the network can also reduce the communication performance. However, the impact of that is not significant on the investigated CFD solver and the effect will be omitted in the following discussion. Another measure of interest is the length of a message when latency and bandwidth-related transfer time are equal. For the SCI $n_{1/2}$=1250 bytes and for the Gigabit Ethernet $n_{1/2}$=2690 bytes. Messages of shorter length than $n_{1/2}$ would be dominated by latency, whereas messages of longer length than $n_{1/2}$ would be dominated by bandwidth.[20]

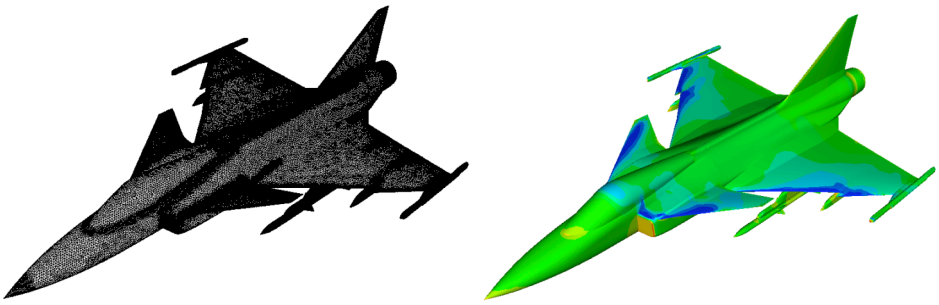


a) Bandwidth **b) Latency**

Fig. 3 MPI performance measured with the ping-pong test.

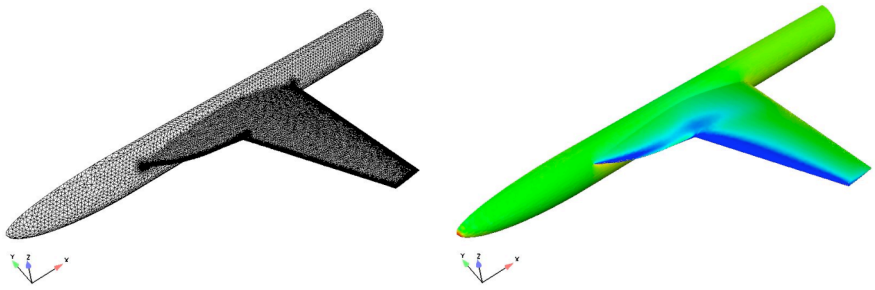
D. Benchmark test cases

The evaluation is performed using two test cases representing typical aerospace applications. The first models inviscid flow, given by the Euler equations, around a highly resolved geometry. The second solves the Navier-Stokes equations around a simplified configuration. These examples are typical representatives of the complexity and problem size of the day-to-day CFD-analysis process found in the aerospace industry.[8] In the Euler case the flow field is solved for a fighter jet with external stores. A depiction of the surface grid used in this computation is shown in Fig. 4a. The case is geometrically complex with detailed external stores placed underneath the wings and a total of 3 million nodes corresponding to approximately 18 million tetrahedral volume-elements are needed for a full-span model to adequately resolve the geometry and the flow features. A fully converged steady-state solution can be achieved in about 500 multi grid cycles. Computational models of this type and resolution are currently employed for configuration analysis, aerodynamic interference analysis and aerodynamic data generation. Often a large number of cases with different flow conditions are computed. In the present case the aerodynamic installation effect on the external stores is studied at M_∞=0.8 and 10^o angle of attack. Figure 4b shows the pressure distribution on the upper side of the aircraft. The Navier-Stokes case represents the other dominating area for CFD-applications, detailed design and analysis, where advanced physical modeling and time-dependent phenomena are the key elements. This is here exemplified by flow analysis around a wing-strake combination mounted on a simplified body.[21] Using a simplified body reduces the geometrical complexity and allows for high resolution of the wing-strake combination while maintaining a reasonable total number of grid points, see Fig. 5a. The flow condition is M_∞=0.2 and 12^o angle of attack. The flow is dominated by a strong vortex on the strake and on the inner part of the see Fig. 5b. To capture the flow separation correctly both the leading edge resolution and the boundary layer resolution must be adequate. Here a hybrid grid is chosen with prismatic elements in boundary layer region and tetrahedral elements outside. A total of 3.1 million points are used for a half-span model.



a) Surface mesh **b) Pressure distribution**

Fig. 4 Inviscid flow around fighter aircraft at M_∞=0.8 and α=10^o



a) Surface mesh b) Pressure distribution

Fig. 5 Viscid flow on wing-strake configuration at M_∞=0.2 and α=12°

E. Analysis of node configuration

In recent years, dual node configurations have become a standard in cluster computing. The drop in system cost and power consumption, the reduction of space and wiring complexity and the possibility to use shared memory paradigm are some key reasons for this development. When using a high-performance network the cost of the network often represent a major part of the total system cost. A way to reduce the network cost per processor is to use dual nodes. For the SCI clusters in this study the network share of the total cost is approximately 50% using dual nodes. For Gigabit Ethernet clusters the network cost is often below 10% even for single node configurations. However, single node configurations can usually obtain a better performance compared to their dual counterparts, mainly due to memory bus contention in the dual node. The choice between single and dual nodes must therefore be made based on the target application, network requirements and budget restrictions. In this study the node configurations are analyzed using the unstructured CFD solver discussed above as target application. In Table 2 is the performance measured in Mflop/s in single grid mode presented for serial and 2-cpu runs. The 2-cpu performance is measured both using single and dual node configurations. The performance is measured using the Euler case described above with 3 Mpoints. Results are obtained using the Intel 7.1 compiler. For comparison one result with the Portland Group compiler is enclosed, showing a performance 15% below what is obtained with the Intel compiler. The performance scales slightly less than the clock-speed. The reason is that there is still a rather high number of cache misses. Edge reordering is used to improve the cache-hit ratio by storing data needed consecutively closely in memory, but the present edge reordering method still has an improvement potential. The overall performance of the serial code is however in comparison with similar unstructured mesh solvers used in industry.[6] Using dual nodes instead of single nodes decrease the 2-processor performance by 20%. In this case the memory bandwidth becomes the limiting factor.

Table 2 CFD solver performance using different node configurations

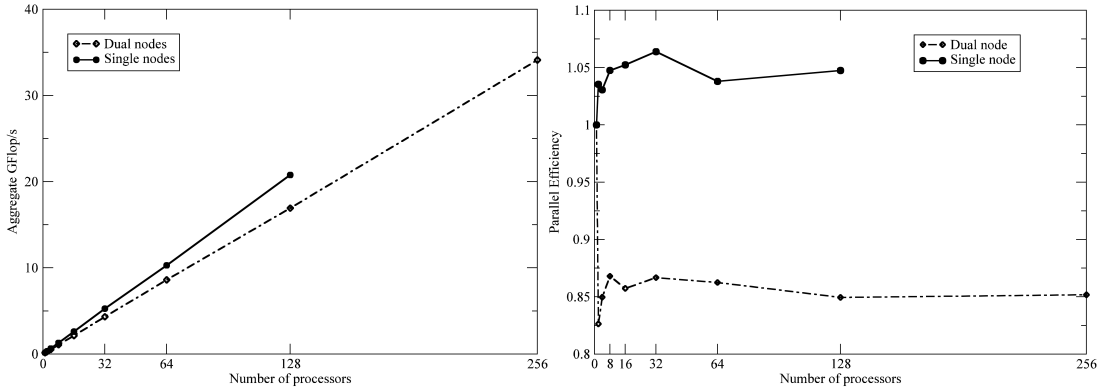
Processor	Performance, Mflop/s			Compiler
	Serial	2 proc MPI	2 proc dual, MPI	
P4, 2.8 GHz	191.8	387.5	-	Intel 7.1
Xeon, 2.4 GHz	169.2	332.2	276.8	Intel 7.1
Xeon, 2.2 GHz	155.9	321.0	257.7	Intel 7.1
Xeon, 2.4 GHz	147.0	-	-	PGI

F. Analysis of parallel CFD code performance

The parallel performance is analyzed in a number of aspects using the two fixed size test cases described above. The reason for choosing fixed size problems is mainly to reproduce a typical scenario from the aerospace design environment. As the code is in daily industrial use the benchmark results concentrate on overall performance measures on the entire code (excluding I/O), rather than detailed instrumentation of selected code sections. To measure the performance the number of floating point operations per iteration is collected through hardware

counters and the wall clock time per iteration is subsequently measured for the parallel runs and the performance is computed.

We begin with a presentation of the performance obtained using the 3 Mpoints Euler case in single grid iteration. Using 128 dual nodes with 256 processors a total computational performance of 34 Gflop/s is achieved using the SCI network see Fig. 6a. The figure also shows that on 128 single nodes 21.5 Gflop/s is obtained. The parallel efficiency plotted Fig. 6b shows values slightly above the ideal 1.0 in the single node case. This is an effect of the fixed problem size where an improved usage of the cache lines is achieved when dividing the problem among many processors. The memory bandwidth saturation in the dual node configuration is clearly visible in both graphs. In the computational performance the slope of the dual node curve is 20% less than the single node curve. The parallel efficiencies stabilize around 1.05 and 0.85 for the single node and the dual node configurations respectively. The fact that the parallel efficiency reaches a stable level with a large number of processors indicates that the network performance is sufficient.

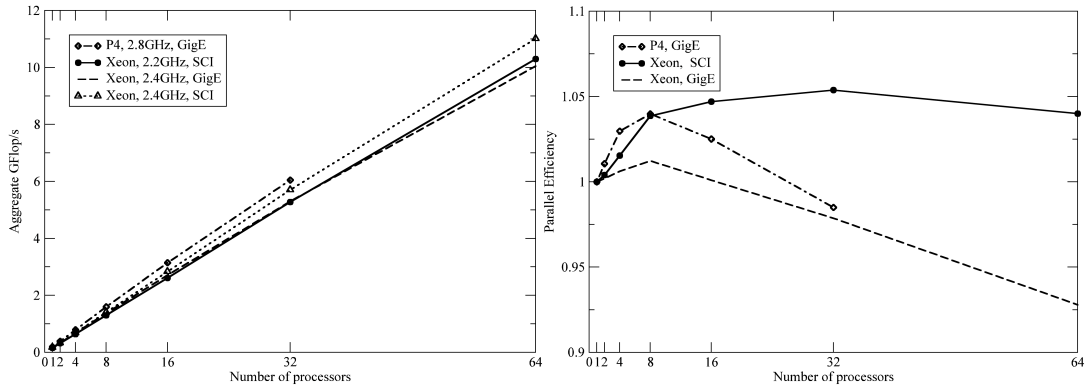


a) Computational performance **b) Parallel efficiency**

Fig. 6 Measured performance on single and dual nodes with SCI network.

At each node is a single network interface responsible for communication to and from the node. Using dual nodes the network card is shared by two processors. The effect of sharing network interface between two processors is negligible as the difference in performance is mainly caused by the node internal memory bus saturation. Actually the dual node configuration performs slightly better when the effect of memory bandwidth limitation is removed. This is believed to be an effect of using shared memory internally in the node to handle messages between the two processors belonging to the same node. The communication load is thus lowered on the network interface compared to the single node configuration. It also indicates that the message transfer between nodes is relatively modest compared to the network capacity, averaging at a few Mbytes/s per node. In Fig. 7a is the computational performance presented versus number of processors for the three different clusters up to 64 processors. The single node P4 cluster with Gigabit Ethernet performs very well up to full cluster size of 32 processors. At 32 processors the efficiency loss is only 2% compared to ideal, see Fig. 7b. The parallel efficiency peaks at 8 processors using Gigabit Ethernet. Using the SCI network the parallel efficiency peaks later, at 32 processors. An interesting aspect is the computational speed in relation to hardware cost. The cluster with Gigabit costs 50% of the SCI clusters per processor. At least up to 64 processors the Gigabit alternative with outperform the SCI configuration in performance/cost.

For good load balancing between the processors the computational load should be evenly distributed while keeping the communication to a minimum. The difference in communication volume when using k-MeTiS and p-MeTiS partitioning algorithms previously shown in Fig. 2 is also observed in the speedup numbers in Fig. 9. The algorithm resulting in the lowest number of points to communicate also gives the best parallel performance.



a) Computational performance **b) Parallel efficiency**

Fig. 7 Measured performance on different node and network combinations

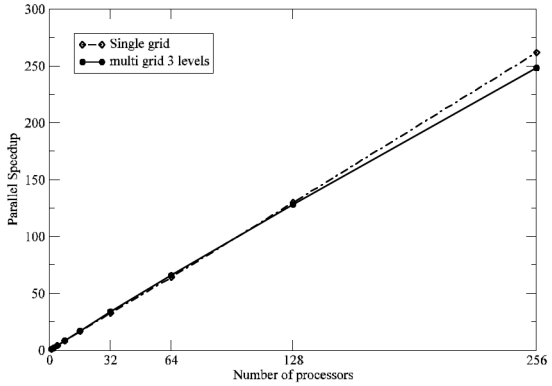


Fig. 8 Observed speedup using single and 3 level multi grid iteration with SCI network.

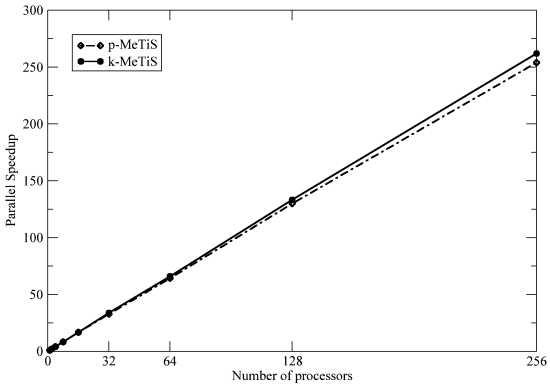
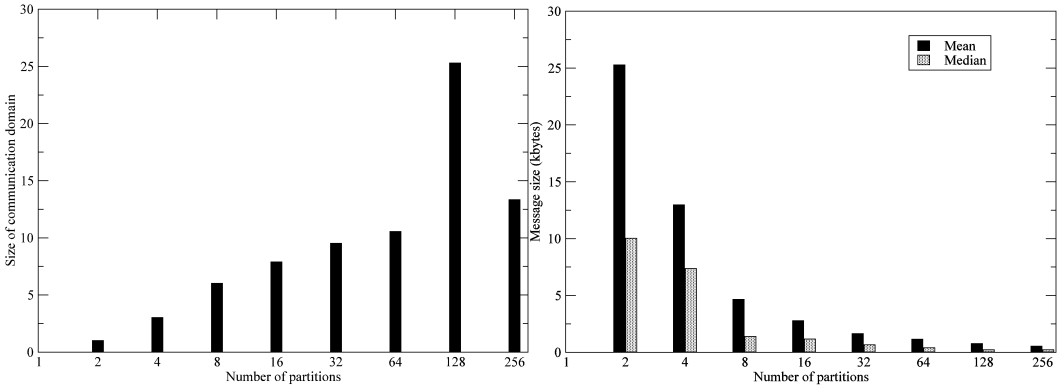


Fig. 9 Observed speedup using p- and k-MeTiS partitioning algorithms.

Solving the Euler equations in single grid mode requires approximately 3820 floating-point operations per grid point and iteration. During each iteration are 33 variables communicated between the connected points in messages of size $N_{connodes}$ and $5 \cdot N_{connodes}$. The communication pattern is statically determined by the partitioning and the mean number of communicating partitions per processor is presented in Fig. 10a. The communication pattern is not entirely symmetric between partitions. A small deviation in the amount of data communicated in each direction can occur when one point in a partition is connected to several points in the neighboring partition.

Figure 10b presents the mean and median message sizes for the k-MeTiS partitioning from 2 to 256 partitions. Already at 8 processors the median message transfer is significantly affected by latency in both the Gigabit Ethernet and SCI configuration. The higher latency of the Gigabit Ethernet affects the parallel efficiency reported in Fig. 7b

where a decrease in efficiency is observed after 8 processors. The much lower latency in the SCI network postpones the effect to after 32 processors and also decreases the efficiency loss. When solving the turbulent Navier-Stokes equations the solution process is more computational intense compared to the Euler equations. The number of floating point operations per grid point and single grid iteration is 3820 in the Euler case and 5850 solving the Navier-Stokes equations with an EARSM modeling the effect of turbulence. Also the communication pattern alters when using the Navier-Stokes equations instead of the Euler equations. More information needs to be transferred between the partitions. Instead of 33 variables communicated in 13 calls as for the Euler case the Navier-Stokes case requires communication of 62 variables per connected grid point in 19 calls. The Navier-Stokes solution process has a lower computation to communication ratio compared to the Euler solution process. Semi-coarsening is used in the boundary layer on the coarse grid levels and this improves the computation to communication ratio. In Fig. 11 the speedup numbers are compared solving the Euler and the Navier-Stokes equations. The better speedup for the Navier-Stokes case is due to increased effect of cache line utilization for the Navier-Stokes equation when using many processors.



a) Size of communication domain **b) Mean and median size of messages**

Fig. 10 Statistics of the communication domain and message sizes using k-MeTiS.

VI. Conclusions

The parallel performance of an unstructured mesh CFD solver is evaluated on PC-clusters using industrial applications. Different factors influencing the parallel performance are investigated. The configuration of the PC-cluster is found to be important, where the two key elements are the node configuration and the choice of internal network. Dual nodes often found on large clusters loose efficiency due to memory bandwidth limitation. Sharing the network interface between two processors introduce only minor effects as the interface is far from saturated and the number of messages is low. The low-cost Gigabit Ethernet network performs very well up to the tested 64 processors and with single processor nodes it forms a very cost-efficient cluster combination. High performance network as the SCI permits excellent speedup beyond 256 processors but is far more expensive. On the software side efficient load balancing and minimizing message transfer between partitions are important factors. Using multi grid iterations is vital for efficient convergence but decreases the parallel efficiency when coarse grid levels introduce relatively more communication. The effect is however reduced when coarse grid levels make better use of the 2nd level cache. The Intel Fortran90 compiler delivers a more efficient executable compared to the Portland Group compiler on Intel processor. The difference is measured to 15% in wall clock time.

The efficiency of the CFD solver on large PC-clusters is important for industrial applications, where low turn around time is crucial. The code has proved to deliver excellent performance on different PC-clusters and different contributing factors for parallel performance are identified. Running 256 processors a performance of 34 Gflop/s is achieved. This means a fully converged Euler solution for a highly resolved full-span configuration in less than 10 minutes. This will definitely have an effect on how and when CFD is applied in the design process.

Acknowledgment

Part of the work presented in the paper was performed when the author worked at NSC. The technical involvement from the NSC staff is highly appreciated.

References

[1]Jameson A., Schmitt, W., and Turkel, E "Numerical Solution of the Euler Equations by Finite Volume Methods Using Runge-Kutta Time-Stepping Schemas," AIAA Paper 81-1259, 1981.

[2]Thomas J. L., "A Perspective of Computational Fluid Dynamics at NASA," *Numerical Methods for Fluid Dynamics V*, edited by K. W. Morton and M. J. Baines, 1995, pp. 19-36.

[3]Salmon J., Stein C., and Steling, T., "Scaling of Beowulf-Class Distributed Systems," *Proceedings of the International Conference of High Performance Computing and Communication (SC98),* IEEE Computer Society, Nov. 1998.

[4]Top500 Supercomputer Sites, available online at http://www.top500.org/lists/2004/11 (cited 9 Dec. 2004).

[5]Behr M., Pressel, M. P., Sturek, W. B., "Comments on CFD Code Performance on Scalable Architectures," *Computer Methods in Applied Mechanics and Engineering*, Vol. 190, 2000, pp. 263-277.

[6]Mavriplis D. J., "Parallel Performance Investigation of an Unstructured Mesh Navier-Stokes Solver," *The International Journal of High Performance Computing Applications*, Vol. 16, No. 4, Winter 2002, pp. 395-407.

[7]Aumann, P., Barnewitz, H., Schwarten, H., Becker, K., Heinrich, R., Roll, B., Galle, M., Kroll, N., Gerhold, Th., Schwamborn, D., and Franke, M., "MEGAFLOW: Parallel Complete Aircraft CFD," *Parallel Computing*, Vol. 27, 2001, pp. 415-440

[8]Vos, J. B., Rizzi, A., Darracq, D., and Hirschel, E. H., "Navier-Stokes Solvers in European Aircraft Design," *Progress in Aerospace Sciences*, Vol. 38, No. 8, Nov. 2002, pp. 601-697.

[9]Fisher, M. S., Mani, M., and Stookesberry, D., "Parallel Processing with the Wind CFD Code at Boeing," *Parallel Computing*, Vol. 27, 2001, pp. 441-456.

[10]Bailey, D. H., Harris, T. V., der Wingaart, R., Saphir, W., Woo, A., and Yarrow, M., "The NAS Parallel Benchmark 2.0," Technical Report NAS-95-020, NASA Ames Research Center, 1995.

[11]Garcia, C., Montero, R. S., Prieto, M., Llorente, I. M., and Tirado, F., "Beowulf Performance in CFD Multigrid Applications," *Proceedings of the 10th Euromicro Workshop on Parallel, Distributed and Network-Based Processing*, IEEE, 2002.

[12]Bessonov, O., Fougère, D., and Roux, B., "Using a Parallel CFD Code for Evaluation of Clusters and MPPs," *Proceedings of the International Parallel and Distributed Processing Symposium (IPDPS03)*, IEEE, 2003.

[13]Wallin, S., and Johansson, A., "An Explicit Algebraic Reynolds Stress Model for Incompressible and Compressible Turbulent Flows," *Journal of Fluid Mechanics*, Vol. 403, pp. 89-132, 2000.

[14]Eliasson, P., "Edge, A Navier-Stokes Solver for Unstructured Grids," *Proceedings of Finite Volumes for Complex Applications III*, 2002, pp. 527-534.

[15]MPI Forum, "MPI: A Messages-Passing Interface Standard," *International Journal of Supercomputing,* Appl. 8, 1994.

[16]Karypis, G., and Kumar, V., "Analysis of Multilevel Graph Partitioning," Technical Report 95-037, Univ. of Minnesota, 1995

[17]The Dolphin SCI Interconnect - white paper, available online at http://www.dolphinics.com/papers/index.html (cited 9 December 2004).

[18]Scali High Performance Clustering Products, available online at http://www.scali.com (cited 9 Dec. 2004).

[19]National Supercomputer Centre in Linköping, Sweden, available online at http://www.nsc.liu.se (cited 9 Dec. 2004).

[20]Sterling, T. L., Salmon, J., Becker, D. J., and Savarese, D. F., "How to Build a Beowulf," MIT Press, Cambridge, MA, 1999.

[21]Sedin, Y., Persson, I., and Sillén, M., "Computational Analysis and Re-Design of a Wing-Strake Combination at High Angles of Attack," *Proceedings of the 2004 ICAS Congress*, 2004.