COST-EFFICIENT USE OF PARALLEL COMPUTERS IN AIRCRAFT DESIGN

Mattias Sillén*
***Saab Aerosystems**

Keywords: *Linux-cluster, Parallel performance, CFD, Cluster configuration*

Abstract

Linux PC-Clusters are a cost-efficient platform for parallel computational fluid dynamics (CFD) applications. This paper investigates the performance characteristics of an unstructured explicit flow solver on multiple clusters. Different node configurations and network architectures are considered in the evaluation. Single processor performance as well as overall parallel performance are compared and evaluated and recommendations are made for performance and cost-efficiency.

1 Introduction

The current challenges in the aerospace field are to offer products that are both better in performance and also faster and cheaper to produce. Thus the current business market forces aircraft designers towards risk minimization and a definitive reduction in cost and time to market. The possibility to influence life cycle cost is largest in the early design stages. This means that the confidence in the design must increase in the early phases compared to today. Introducing high fidelity simulations early in the design process will facilitate this if the turn around time requirement can be met. In early phases the allowable time frame to conduct flow simulations is very limited, in extreme cases in the order of minutes. Also in later design stages there is a need to reduce the total simulation wall time to allow for more advanced physical modeling or increased resolution that due to computational cost not are feasible today.

The necessity of high fidelity modeling in the design process is often conflicting with the requirement of short turn around time. A balance between modeling requirement and turn around time limitation needs to be established at every phase in the design cycle. A common way to reduce the turn around time is to use powerful parallel computer systems. Traditionally, supercomputer resources have been equivalent with large cost and therefore not widespread in industry. This started to change in the late 1990s when PC–cluster with Linux, so called Beowulf systems ref. [1], became popular. Larger and application specific computer systems are now designed using cheap commodity components.

When designing a PC–cluster for a specific application several design choices have to be made. The compute node configuration and interconnecting network are the two most important components in a cluster affecting the performance. Compute nodes are generally equipped with one or two processor even though other configurations are available. In this study single and dual processor nodes are evaluated and differences between 32- and 64-bit architectures are investigated. A comparison is made between commodity communication hardware, such as switched Gigabit Ethernet, as well as hardware specifically designed for high-performance computing, such as SCI and Infiniband.

Making the better choices will provide improved performance of the flow solver and thereby offering a chance to improve the modeling capability or reducing the turn around time in the design process. Other important issues for parallel performance are parallel implementation strategy, inter-processor

communication and load balancing. The influence of these factors on total performance is also analyzed on a typical aerodynamic design example.

A previously reported study on parallel CFD performance on Linux-clusters provides some additional background of the code and the parallel implementation, see ref. [2].

2 Numerical Solution Method

This study is based on results obtained with the unstructured Navier–Stokes solver Edge [3], developed at the Swedish Defence Research Agency (FOI). The solver has an edge based formulation that makes it possible to compute on any type of mesh: structured, unstructured (with tetrahedral, hexahedra, prism or pyramids) or hybrid. 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. All results below are obtained with the central scheme with artificial dissipation.

3 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 node-centered finite volume discretization ghost points are introduced where the partition boundaries cross edges to compute the fluxes locally, see Fig. 1.

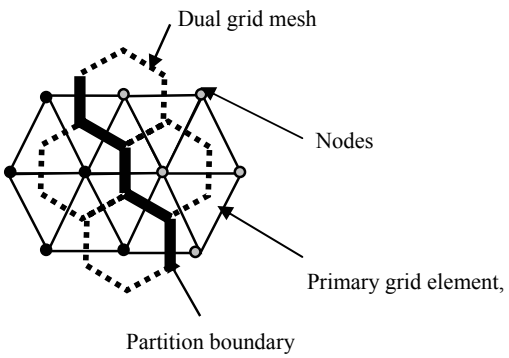

Figure 1 Layout of primary grid cells, dual control volume and partition boundary.

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 [4]. 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 amount of communication when changing grid level but will guarantee a better load balance also on coarser grids levels, see [5].

4 Benchmark architecture

In this performance study five different clusters are evaluated, listed in Table 1. They are based on either 32- or 64-bit Intel processors. The nodes are equipped with single or dual processors and the interconnecting network is either Gigabit Ethernet and/or high-speed interconnects as SCI and Infiniband. In the following sections the clusters are briefly described. All clusters are designed by National Supercomputer Centre (NSC) at Linköping University in Sweden.

Monolith – 32-bit Intel Xeon

Monolith consists of 200 dual Xeon nodes. Each node has 2 GB RAM and a 2.2 GHz Xeon processors. The nodes are connected both by Fast Ethernet and SCI. The topology for the SCI network is a 3D torus (5x8x5). It entered service in November 2002. The system is dedicated to academic research in Sweden. A special mpi implementation, ScaMPI, is used for the SCI network. ScaMPI performance between two nodes is about 4.5 μs in latency and 260 MB/s in bandwidth. The Fast Ethernet network is only intended for file transfer, not parallel applications.

Maxwell – 32-bit Intel Xeon

Maxwell was build following the same principles are Monolith. It contains 40 nodes with dual 2.4 GHz Xeon processors linked by SCI (2D 5x8) and Gigabit Ethernet. It entered service in March 2003. The system is dedicated to aeronautical simulations at Saab Aerosystems. ScaMPI is used as mpi implementation to communication over the SCI network. For the Gigabit Ethernet the mpich implementation is used. The mpich performance between two nodes is about 30 μs in latency and 70 MB/s in bandwidth.

Stokes – 32-bit Intel P4

Stokes is a small commodity cluster with 32 Intel P4 processors each equipped with 2 GB RAM and connected with a switched Gigabit Ethernet network.

Name	Processor	# Nodes	# proc./node	GHz	Cache size	Network
Monolith	Intel Xeon IA32	200	2	2.2	512 kB	SCI, Fast Ethernet
Maxwell	Intel Xeon IA32	40	2	2.4	512 kB	SCI, Gigabit Ethernet
Stokes	Intel P4	32	1	2.8	512 kB	Gigabit Ethernet
Dunder	Intel Xeon EM64T	52	2	3.4	2 MB	Infiniband, Gigabit Ethernet
Darkstar	Intel Xeon EM64T	44	2	3.4	2 MB	Gigabit Ethernet

Table 1 Cluster specifications.

Dunder – 64-bit Intel Xeon

Dunder is equipped with 52 dual nodes connected by Gigabit Ethernet and Infiniband

(10 Gbps) network. The processors are 3.4 GHz 64-bit Xeon with 2MB L2 cache. The system is dedicated for weather predictions and entered service in September 2005. Parallel applications use Scali MPI connect which enables runtime selection of the interconnect.

Darkstar – 64-bit Intel Xeon

Darkstar is dedicated to aeronautical simulations at Saab Aerosystems. It consists of 44 dual processor (3.4 GHz 64-bit Xeon) nodes. The nodes are connected by Gigabit Ethernet. The processors are quipped with a 2MB L2 cache. The system entered service in April 2006. The lam mpi implementation is used.

5 Cluster Performance Evaluation

In parallel processing speedup and efficiency are two important measures of the quality of the parallel algorithm and there are a number of factors limiting the speedup.

1) **Software Overhead** – Even with a completely equivalent algorithm, software overhead arises in the parallel implementation, i.e. there are generally more lines of code to be executed in the parallel program than in the sequential program.
2) **Load Balancing** – Speedup is generally limited by the speed of the slowest node. So an important consideration is to ensure that each node performs the same amount of work, i.e. the system is load balanced.
3) **Communication Overhead** – Assuming that communication and calculation can not be overlapped, any time spent communicating the data between processors directly degrades the speedup. Because of this, a goal in the design of a parallel algorithm is to make the grain size (relative amount of work done between synchronizations - communications) as large as possible, while keeping all the processors busy. The effect of communication on speedup is reduced, in relative terms, as the grain size increases.
4) **Amdahl's Law** – This states that the speedup of a parallel algorithm is effectively limited by the number of operations which must be performed sequentially.

Analyzing the implementation of the flow solver reveals that the load balancing and the communication overhead are the most significant factors affecting the parallel performance. The communication overhead is tightly coupled to network performance of the cluster. There is very little software overhead in the parallel implementation except the lines of code added for data transfer between processors and only few operations need to be performed sequentially, typically global reductions.

A fixed size problem is used in the performance evaluation as the focus is on industrial applications and the intention is to reproduce the situation in the design process. When the problem is parallelized over more processors two parts will influence the performance results more than the other. Firstly the computation to communication ratio will decrease as the partitioning introduces new internal boundaries between domains. Both the total amount of data communicated as well as the number of messages increase. The communication pattern becomes more fragmented and the mean message size decreases. Secondly, when more processors are added the total amount of fast cache memory also increases. This means that a larger part of the total problem will reside in the cache with a subsequent performance gain. This is called cache effect and can result in a super linear speedup, i.e. higher speedup numbers than number of processors.

The evaluation is performed with the Edge code using inviscid flow modelling, given by the Euler equations, around the highly resolved Gripen fighter with external stores. A depiction of the surface grid used in this evaluation is shown in Figure 2. The case is geometrically complex with detailed external stores placed underneath the wings. A total of 3 million points 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 transonic conditions with sideslip. Figure 3 presents an example of the pressure distribution on the upper side of the aircraft where a blue color indicates low pressure regions.

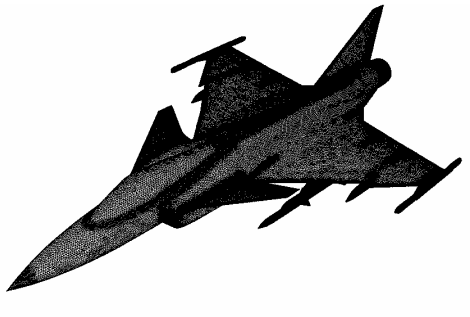

Figure 2 Surface mesh on test example

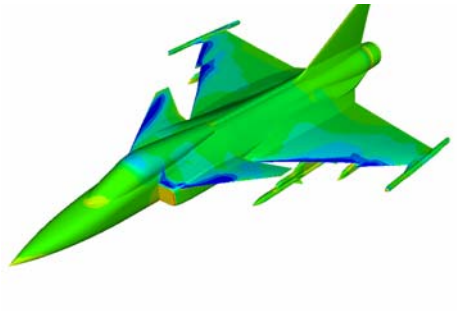

Figure 3 Pressure distribution on test case

Processor performance

Initially the single processor performance is compared for the 32 and 64 bit processors in Fig. 4. The processor performance is measured using both one and both (if available) processors in a node. The Intel 8.1 F90 compiler is used in all computations. A notable observation is that when using nodes with dual processors, which may be attractive from the point of view of cost–efficiency when using an expensive network or from compactness aspects, the performance is reduced roughly 20 % when two processors have to share on a common node memory. The memory bandwidth is in this case not up to the demands of the memory intensive application.

The performance on the 64-bit Xeon is 2.3 times the performance on the 2.4 GHz 32-bit Xeon even though the clock speed only differs 40 %. This is an effect of significantly larger L2 cache (2 MB compared to 512 kB) in the 64-bit version and a twice as high memory bandwidth, 6.4 GB/s compared to 3.2 GB/s. Another observation that the memory bandwidth is a limiting factor is that increase in clock frequency in not fully retrieved in the performance for the 32-bit processors.

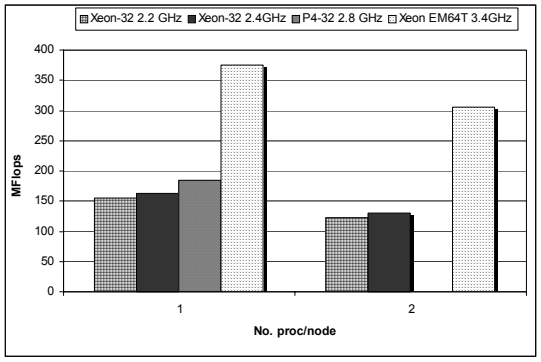

Figure 4 Single processor performance using 1 and 2 processors per node

Network performance

The influence of the network is first evaluated on the large SCI cluster Monolith. Using dual nodes an aggregated computational performance of 34 GFlops is reached on 256 processors. Using only one processor per node it delivers 21

GFlops on 128 processors, see Fig. 5. The application demonstrates a nearly linear parallel speed-up. This is also seen in the graph, Fig. 6, where the parallel efficiency stabilizes on 1.05 for the single processor node and 0.85 for the dual node. From this we conclude that the network capacity is sufficient at least up to 256 processors. Analyzing the communication behavior of the code reveals that the communication pattern quickly gets latency bound. Already at 8 processors the mean message transfer time is affected by latency. The low latency in the SCI network is found to be crucial for good performance on larger number of processors. From the speedup and the efficiency graphs it is clear that the network performs equally well using single and dual nodes. It is believed that this is an effect of using shared memory internally in the node to handle messages between the processors belonging to the same node. The load on the network interface is relatively modest compared to the network capacity, averaging at a few Mbytes/s.

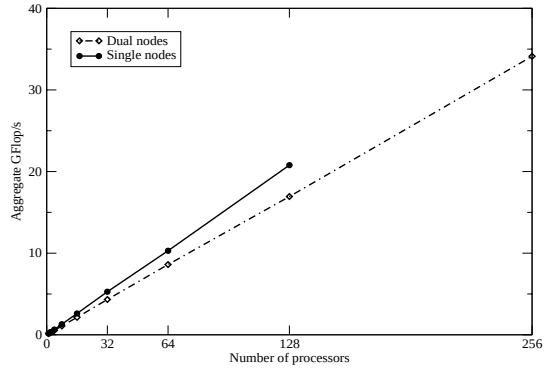

Figure 5 Overall performance on Monolith using single and dual processor nodes.

Gigabit Ethernet is tested up to 64 processors. The overall performance is presented in Fig. 7 together with SCI and Infiniband cluster performance graphs. After 8 processors the efficiency starts to fall and is around 0.9 at 64 processors, see Fig. 8. The main reason for this is the much higher latency (25-30 μs) compared to the high-performance

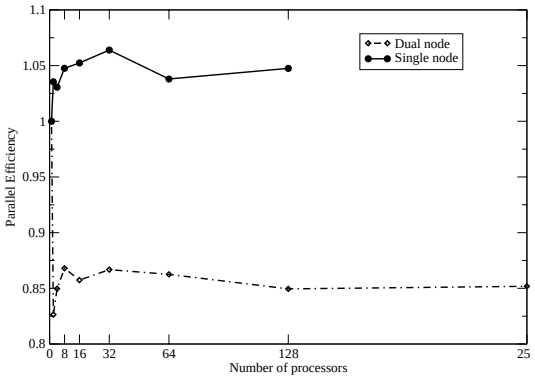

Figure 6 Parallel efficiency on Monolith using single and dual processor nodes.

alternatives (3-5 μs) The Infiniband network performance is comparable to the SCI network, even though the much larger L2 cache in the 3.4 GHz nodes give superlinear speedup effects.

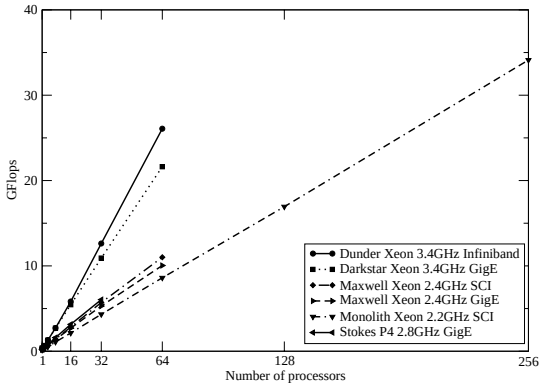

Figure 7 Parallel performance on clusters with different network.

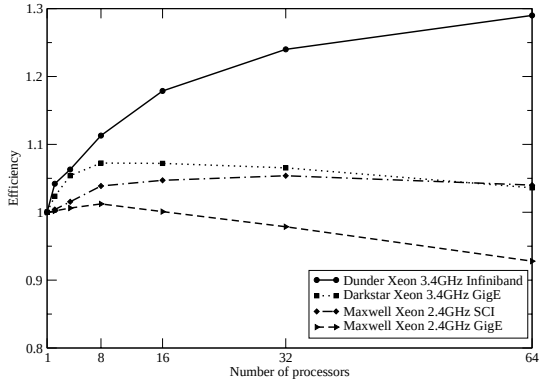

Figure 8 Parallel efficiency on clusters with different network.

6 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 number of grid points to each processor. The partitioning can be performed using various techniques; in this case the standard graph partitioning program MeTiS [6] is used. Balancing the workload alone is however not sufficient when load balancing for larger number of processors. Communication load must also be kept at a low and balanced level. This is exemplified here with a tetrahedral grid containing 3 Mpoints partitioned from 2 up to 256 partitions. Two versions of the MeTiS software are used; p-MeTiS and k-MeTiS. Both deliver completely balanced partitions concerning number of points but they differ in total number of boundary point. Figure 9 shows the mean number of interface boundary points per partitions. Above 4 partitions this is in favor of the k-MeTiS algorithm that also tries to minimize the number of points in the interface 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.

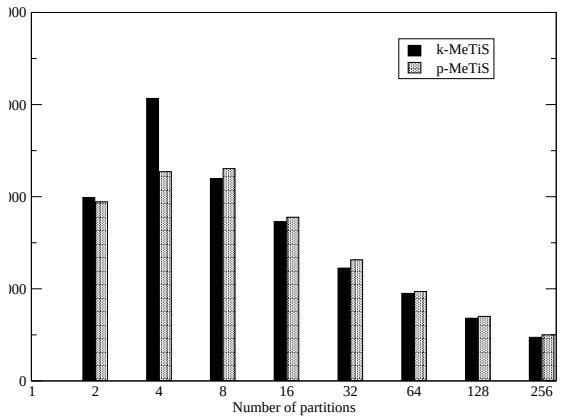

Figure 9 Mean number of points communicated between partitions using different MeTiS algorithms.

In this performance evaluation the load balancing is performed with the k-MeTis algorithm that both distributes an equal amount of grid points to all processors as well as minimizes the number of grid points in the domain boundary region. The additional effect of reducing the number of grid points in the boundary interface compared to only load balancing the number of grid points, using p-MeTiS, is visible from 64 processors, see Fig. 10. The difference at 256 processors is just below 5 %. This will however be of much larger importance when using larger parallel systems.

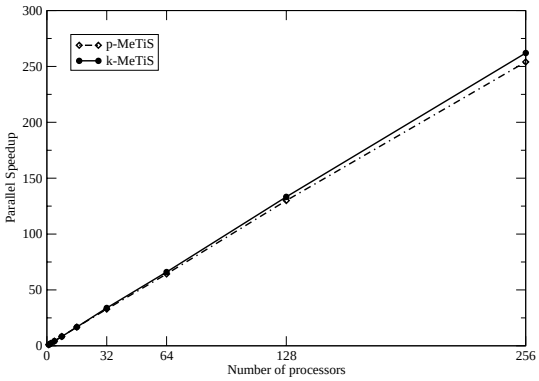

Figure 10 Parallel speedup on Monolith using k-MeTiS and p-MeTiS.

Conclusions

Different cluster configurations are evaluated for performance using an unstructured flow solver in an industrial environment. CFD solvers are memory intense applications and with dual processor nodes the memory bandwidth will be a limiting factor. Examples show a 20 % decrease in performance using dual nodes compared to single processor nodes. The cost difference between a dual node and two single nodes is roughly in the 20-25 % range.

In typical design applications the communication is latency bound starting from approximately 8 processors. Obtaining good parallel performance for hundreds of processors will require a low latency network, which however is significantly more expansive than the Gigabit Ethernet alternative. Gigabit

Ethernet is a cost-efficient alternative for parallel applications up to about 80-100 processors. Above that it performs poorly.

The most appropriate cluster combination for this flow solver depends on the total size of the cluster and the size of the parallel application. For a small cluster, up to 48 processors, single nodes with Gigabit Ethernet will be a cost-efficient solution. For larger clusters dual nodes are preferred. Depending on the parallelization strategy, number of processors per case, low latency networks can be required. The typical mesh size for a flow analysis around a complete aircraft is today in the range of 3–20 million points and this is usually parallelized on 20–80 processors. This will be well suited to run efficiently on cluster configurations with Gigabit Ethernet. Using more than 80–100 processors per case will require low latency networks for efficiency.

References

[1] Sterling T L, Salmon J, Becker D J and Savarese D. *How to build a Beowulf*, Cambridge, MA, MIT Press, 1999.

[2] Sillén M. Evaluation of parallel performance of an unstructured CFD code on PC-clusters", *Journal of Aerospace Computing, Information, and Communication*. Vol. 2, pp. 109-119, 2005.

[3] Eliasson P. Edge, A Navier-Stokes solver for unstructured grids, *Proceedings of Finite Volumes for Complex Applications III*, ISBN 1 9039 9634 1, pp. 527-534, 2002.

[4] MPI Forum, URL: http://www.mpi.org

[5] Mavriplis D J. Parallel performance investigation of an unstructured mesh Navier-Stokes solver. *The International Journal of High Performance Computing Applications*, Volume 16, No. 4, Winter, pp. 395-407, 2002.

[6] Karypis G and Kumar V. *Analysis of Multilevel Graph Partitioning*, Technical Report 95-037, University of Minnesota, 1995.