Helsinki University of Technology
CFD-group/ Laboratory of Applied Thermodynamics

MEMO No CFD/TERMO-29-99 DATE: May 28, 1999

TITLE

Transportation of FINFLO to a cluster of PCs

AUTHOR(S)
Patrik Rautaheimo and Petri Kaurinkoski

ABSTRACT

For a long time, the UNIX workstations have been superior against to the personal computers as far as computational speed is concerned. However, the gap between the speeds has been decreasing. Today's personal computers are fast and relatively cheap, whereas the UNIX workstations are expensive. A comparison between the available personal computers and traditional engineering tools like UNIX workstations, servers and also the supercomputers is needed. In this report such a study is made for a cluster of PCs. It will be shown that with ideal cases the speed and parallel performance figures of a cluster of personal computers are competitive with those of the workstation.

MAIN RESULT

Benchmark runs in a cluster of Linux personal computers, and a comparison with different UNIX-platforms

PAGES
9

KEY WORDS
Parallel computing, Linux, UNIX, MPI, CFD

APPROVED BY

Timo Siikonen May 28, 1999

1 Introduction

For over a decade parallelization has been used to enhance the efficiency of flow solvers. The simplest method of parallelization, which can be used with shared memory machines, takes place on the DO-loop level. DO-loop level parallelization of the low-level loops is ineffective for a large number of processors. Better performance from a large number of processors can be obtained by dividing the space into smaller sub-domains. With a shared memory machine like the Cray C94, the parallelization over the sub-domains is a trivial task, but with a distributed memory system like the Cray T3E or cluster of workstations things get more complicated. A common approach, applied e.g in [1] and [2], is to divide the computational domain into equally sized blocks and to apply message passing between the blocks.

Recent progress in personal computer has risen a question whether the personal computers have reached the speed of UNIX workstations. Traditionally, UNIX operating systems in workstations and supercomputers have been used to solve task demanding large computing resource, like computational fluid dynamics (CFD) codes. Actually, the operating system has been one of the biggest reasons why personal computers have not gained much popularity to solve these problems. Lately Linux operating system, which is closely related to the UNIX operating system, has gained more and more popularity and made the use of personal computers feasible. Most of the desired software, like a FORTRAN-compiler and MPI, is now available to the Linux system.

In this paper, test runs on a cluster of Linux personal computers with a parallel multiblock Navier–Stokes flow solver are described. The parallelization is based on the Message Passing Interface (MPI) Standard [3]. The computer is located in CSC, Espoo. One purpose of this work is to make a comparison between expensive UNIX servers and cheaper PC-cluster. Also the knowledge of the performance of FINFLO software on PCs is one of the goals. The basic ideas behind the flow solver and parallelization strategy is described. The single processor performance comparison is presented, and finally the parallelization of the solver is compared in different platforms.

2 General Description

2.1 Flow Solver

The flow simulation is based on the solution of the Reynolds averaged Navier–Stokes equations:

$$\frac{\partial U}{\partial t} + \frac{\partial (F - F_v)}{\partial x} + \frac{\partial (G - G_v)}{\partial y} + \frac{\partial (H - H_v)}{\partial z} = Q \tag{1}$$

where U is the vector of dependent variables, F, G, H and F_v, G_v, H_v represent the inviscid and viscous parts of the fluxes, and Q is a possible source term. The flow solver utilizes a structured multiblock grid. For the solution Eq. (1) is written in a finite-volume form

$$V_i \frac{dU_i}{dt} = \sum_{\text{faces}} -S(\hat{F} - \hat{F}_v) + V_i Q_i \tag{2}$$

where V_i is a cell volume, $\hat{F}$ and $\hat{F}_v$ are the inviscid and viscous parts of the flux on the cell surface. taken over the faces of the computational cell.

The solution proceeds blockwise after explicitly defined boundary conditions. In each block an implicit LU-factored solution with a multigrid acceleration of convergence is performed [4]. The underlying solution method is based either on the flux-difference [5] or flux-vector [6] splitting. The flux calculation utilizes a MUSCL-type differencing with a second- or third-order accuracy. The code has been applied for external [7] and internal [8] flows.

2.2 Parallel Implementation

The code structure forms an ideal base for the parallelization. All the essential procedures are treated block by block including the updating of boundary conditions. By using block sizes like $32^3, 40^3$ and 48^3 several multigrid levels can be used in each block. If all the blocks are of an equal size, the work between the processors is balanced. With the current RISC processors and the block sizes given above, the calculation takes in the order of 10 seconds per iteration cycle. Since the majority of the calculation takes place slabwise, it is impractical to use very small block sizes owing to the useless computation of ghost cells. Even more important is to obtain a suitable balance between the times spent on the computation and the communication, which with current fast processors requires that blocks have a sufficient size.

The parallelization is based on the Message Passing Interface (MPI) Standard [3]. MPI routines are implemented so that the program also runs in an environment, where MPI is not implemented. The updating of boundaries between different processes is done using the basic MPI_SSEND and MPI_RECV commands.

The code has been used in various parallel platforms. These include the Cray T3D, the Cray C94, the Cray T3E, a multiprocessor UNIX server and, a cluster of UNIX workstations. The parallel implementations are described in details in Refs. [9, 10, 11, 12].

2.3 Platform

Platform is provided by Center for Scientific Computing (CSC). It contains 16 Dell's precision workstation 410 computers. Each computer has Dual Intel Pentium II 400 MHz processors with 128 Mbytes memory. The computers are connected to each other by a 100 Mbit ethernet. Operating system is Linux. More information of the cluster and it's software can be found in
http://www.csc.fi/metacomputer/pckluster/.

3 Compiling and Running the Program

Software is compiled using f90. The biggest problem in this particular application became from the fact that the Absoft Corporations compiler does not have inlining option. Part of the code had to be inlined by hand. Best performance was found if the program is run through a FORTRAN preprocessor called kapf90 by Kuck and Associates. This software was found from the CSC's Digital server (caper.csc.fi). Other changes were not needed. The compiling command was the following

```
[rautahei@terttu kappi2]$ mpif90 -B100 -O -c ns3c.f
```

The Program can be automatically run from the server (terttu.csc.fi). A user must only specify the number of processors needed in the run (16 in this case) and the processor that are used

Table. 1: Effect of the compiler directive on the performance.

Directive	t/t_{best}
-O	1.135
-O -B101 + inlining by hand	1.117
-O -B100 + inlining by hand	1.068
-O -B100 + kapf90	1.000

Table. 2: Single-process performance in different platforms.

Platform	Speed (Mflops)	Speed (Cray T3E)
C94 (240MHz Vector processor)	367	6.75
SGI Indigo^2 (R10000 195 MHz IP28 with 1 Mbyte secondary cache)	90	1.94
SGI Origin2000 (R10000 250 MHz IP27 with 4 Mbyte secondary cache)	140	2.63
COMPAQ AlphaServer 8400 (525 MHz EV6 with 4 Mbyte secondary cache)	244	4.596
T3D (150 MHz Digital alpha processor 21064 with 8 kbyte secondary cache)	11	0.21
T3E (375 MHz EV5 with 96 kbyte secondary cache)	53	1.00
T3E (with no streams)	48	0.91
Dell precision 410 (400 MHz Pentium II)	73	1.38

```
[rautahei@terttu ~/delta16]$ time mpirun -nolocal
-machinefile  ../nodet -np 16 finflo
```

where the file `../nodet` is a list of the PCs that will be used.

4 Results

There are two commonly used test methods in the parallelization. One is to keep the size of one process computation constant, so-called scaling. This means that total problem size rises as the number of processes increases. Another is to keep the total problem size constant and divide it between the processes. Hereafter this is called blocking.

First, a single processor performance is presented with various platforms. Secondly, the parallel test computations are made for the scaling and blocking cases.

4.1 Single Processor Performance

As a test case, a Delta-wing in transonic flow was calculated. A grid contains 32^3 computational cells and it required roughly 20 Mbytes memory. Table 1 shows the effect of different options. The best performance is achieved with kapf90 FORTRAN preprocessor and flags **-O -B100**, although the difference is not big, if the compiler could do the inlining (roughly 7%).

Table 2 depicts a comparison of a single-processor performance for different platforms. The simulation case was the same as above. The performance was measured in Mflops

Table. 3: Performance of the parallelization in scaling.

NPS	N. of cells	η T3E	η PC-cluster	η SGI
1	32 768	1.000	1.000	1.000
2	65 536	0.998	0.981	0.950
4	131 072	0.998	0.980	0.850
8	262 144	0.998	0.987	0.904
16	524 288	0.998	0.931	0.859
16b	524 288		0.726	
32	1 048 576	0.998	0.700	N/A
64	2 097 152	0.998	N/A	N/A

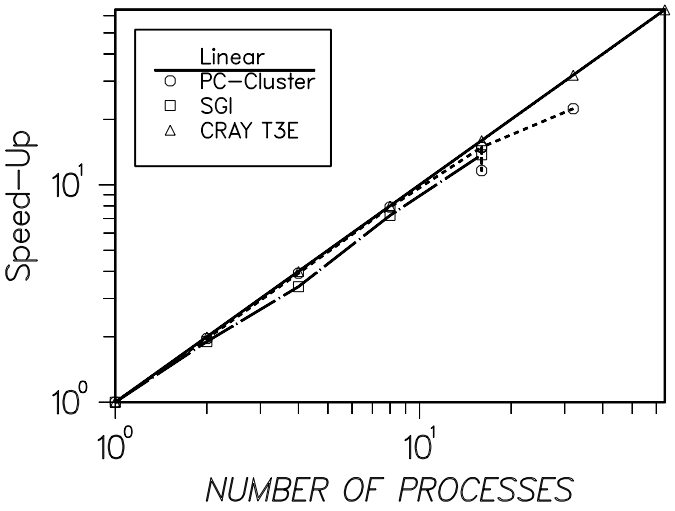

Fig. 1: Speed-up of the parallelization in scaling.

(flops per iteration cycle is obtained from the C94 computation), and also as a ratio of the speed of the Cray T3E. The performance with the Linux PC is 73 Mflops. This value is larger than that for the Cray T3E single processor, and also only about 50 % lower than a value of a bit old UNIX server (Origin2000 with R10k). Note also that the performance with the C94 will be better (≈ 500 Mflops), if a larger grid is used.

4.2 Scaling

Grids were generated so that all the blocks have a size of $32 \times 32 \times 32$. The computational domain size is from one to 64 different blocks, and each block was calculated in a different process. For the PC cluster, the largest case is 32 blocks. Thus the coarsest grid has 32,768 and the densest grid 1,048,576 computational cells for PC cluster.

The efficiency η is obtained directly from the absolute time spent in the calculation. The results are presented in Table 3 and the speed-up in Fig. 1. It can be seen that a perfect scaling is achieved in these test runs with the T3E up to 64 processes. The PC-cluster has good scaling up to 16 processors. For 32 processors, there is sudden drop in efficiency. Because of this sudden drop of efficiency, when moving from 16 to 32 processors, more tests were done with 16 processor. In the PC-cluster, there are two processors in each PC (16 PCs with 32 processors). Simulation 16b was done with 8 PCs and 16 processors, whereas

Table. 4: Performance of the parallelization in blocking.

NPS	N. of cells/block	η T3E	N. of cells/block	η PC-cluster
1	131 072	1.000	614 400	1.000
2	65 536	0.992	307 200	N/A
4	32 768	0.981	153 600	0.988
8	16 384	0.948	76 800	0.951
16	8 192	0.927	38 400	0.965
16b			38 400	0.801
32	4 096	0.894	19 200	0.848
64	2 048	0.876		

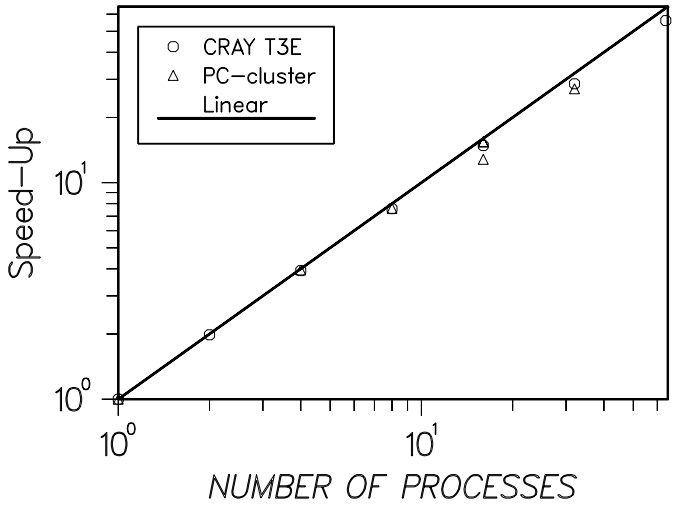

Fig. 2: Speed-up of the parallelization in blocking.

simulation 16 was done with 16 PCs and processors. It can be seen that if both processors are used in a single PC, the performance collapses. With the T3E and the PC-cluster tests are made so that the global iteration history is not collected during the simulation. This reduced the communication work into some extent. With the SGI cluster the global iteration history has been collected during the iteration, and consequently, the parallelization is not as good as in the case of the T3E or PC-cluster. Also the code version is a bit older with the SGI simulation, and communication is done through a low-speed ethernet (10 Mbit/s). The speed-up can be seen in Fig. 1 with the results obtained from different platforms. Note that both cases 16 and 16b are presented in the figure. The speed-up is linear for the T3E and almost linear up to 16 processor for the PC-cluster. The total speed of the cluster with all the processors in use is $32 \cdot 0.7 \cdot 73$ Mflops $= 1.64$ Glops.

4.3 Blocking

Another way of testing parallelization is to divide a big problem into small ones. With the T3E the grid size is $64 \times 64 \times 32 = 131,072$, and for a PC-cluster the grid size is $192 \times 80 \times 40 = 614,400$. In both cases the size is limited by the processor memory size in the T3E and the PC-cluster. Because of different block sizes, the comparison is not straightforward. Block sizes for different cases can be seen in Table 4.

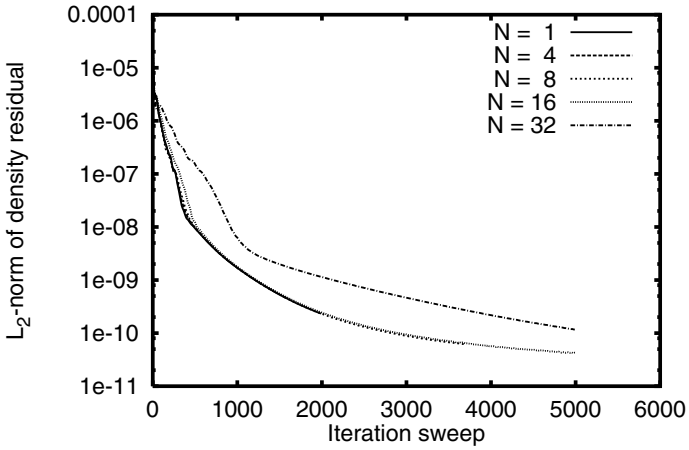

Fig. 3: L_2-norm of ρ-momentum residual.

Partly unexpected results are shown in Table 4 and Fig 2. Firstly, the parallelization is better for the PC-cluster than for the T3E. The situation is different for the PC-cluster, because the block sizes are larger in the PC-cluster than in the T3E simulation. Secondly, the parallelization is better for the PC-cluster for blocking than parallelization for scaling. One explanation can be that the single processor performance increases, when the simulation case gets smaller. This is called a super-linear scaling. As an evidence of this behaviour, the test runs were made in the PC-cluster. The computing time per iteration per computational node is taken for one processor run with two different mesh sizes. The time spent in one computational cell is 142.98 μs with the smaller grid and 151.85 μs with the larger grid. Grid sizes are 32,768 and 614,400 points. It is seen that difference in the single processor performance is increased 5%, when the grid size gets smaller. Maybe the speed of the T3E is not so sensitive to the grid size. Also, with a very small block size, as used in the T3E test runs, the fact that unnecessary ghost cells are calculated in each block also decreases the efficiency. The faces of the block are not of an equal size, and the communication times between different processors are not in balance. With these observations it is clearly seen that the block size should be sufficiently large, in this case 8,000 cells seems to be the lower limit. Again with 32 processors the performance collapses with the PC-cluster. Thus behaviour is similar as for the scaling. Simulation 16b was done with 8 PCs and 16 processors. Difference between the simulations 16 and 16b is clearly seen in Fig. 2. The speed of the total cluster in this case is $32 \cdot 0.848 \cdot 142.98/151.85 \cdot 73$ Mflops $= 1.87$ Gflops.

Since the boundary conditions are treated explicitly, splitting of the computational domain into smaller parts will also reduce the performance of the implicit stage. This was tested with the ONERA M6 wing by dividing the original grid of size $192 \times 80 \times 40$ into pieces. Iteration histories of L_2-norm of the ρ-momentum can be seen in Fig. 3 as calculated with different block sizes. It can be seen that the convergence is not much affected by the grid size. Only for a 32 processors the convergence is slower. However, it should be noted that in this case the smallest block size is still relatively large, $48 \times 40 \times 10$.

5 Discussion

In this study parallel test runs in the PC-cluster are presented. The parallelization takes place over the blocks and the communication between the blocks utilizes the MPI Standard.

A comparison have been done between the PC-cluster, an UNIX-workstation cluster and a massively parallel computer. With the UNIX-workstation cluster the performance curve obtained is almost linear up to 16 processes. This is in spite of the fact that the workstations were connected to a standard, low-speed Ethernet. With the Cray T3E machine, test runs indicate an excellent parallelization. With scaling, the parallelization is almost 100%, and with blocking it is still 88%, when 64 processors were used. With the PC-cluster, the parallelization is excellent up to 16 processors. For both scaling and blocking the parallelization efficiency is over 90% up to 16 processors. In case of 32 processors the performance collapses. This is caused by the fact that both the processors in a single PC are in use simultaneously. Linux operating system cannot use dual processors efficiently.

The single processors performance is greatly affected by the problem size. The smaller the problem is, the better the performance will be. Because of this reason the blocking up to 16 processors performs even better for the PC-cluster than in the T3E. Also the test case was different for the PC-cluster and the T3E.

If the size of the case is kept constant, and the parallelization is performed by dividing the grid into smaller and smaller blocks, the efficiency of the code decreases as the number of processors is increased. This is caused by a larger ratio between communication and the calculation as the blocks are getting smaller, and also the extra time spent in the calculation of the ghost cells. Because of this property, and also of the explicit treatment of the block boundaries, the block size should not be smaller than (24^3). In practice this is not a limitation, because computational time is sufficiently small with that size of the block for traditional CFD problems. However, if the time-accurate simulation is done, an order of 100 times more iteration sweeps is needed. In order to keep the computing time acceptable, the smaller block size must be used with the present processing speeds.

Some problems were encountered concerning the operating system and it's software. A list of some unanswered questions is shown below:

- Makefile does not realize if files are changed
- No flexible way was found to read/write binary files from UNIX systems
- To read files through NFS from a FORTRAN code did not work
- No inlining option in the compiler
- Performance of different parallel runs could vary greatly. The dependence was clearly found, if someone else is logged in one of the nodes, ank also some other randomly changing phenomena are present.
- Two processors cannot be used in one PC efficiently

The cost of the computing power is lower in PCs, if it is compared to UNIX workstation or server. Because the parallel work scales very well for the PC-cluster, the larger number of processors can balance a bit faster UNIX processors. If it is assumed that the cost of one PC-processor is one fourth of the cost of UNIX-processor, then the PC-computing is about 3 times cheaper than the UNIX-computing. This number is highly increased, if a comparison is made between the PC and the massively parallel supercomputer. Now, and especially in

the future, the PCs offer a good alternative to computing demanding problems against the more expensive UNIX-systems.

One drawback of the Linux-based systems is the lack of traditional commercial support. This lifts the threshold for jumping in to a Linux-cluster higher, because it is evident that the labour costs involved with the necessary in-house support will be higher in comparison with a fully commercial UNIX-system. On the other hand, the net-community often provides solutions faster than a fully loaded commercial support organization.

In this paper, all the test cases are optimal for parallel work. It is possible that for non-optimal cases the slower communication hardware of PC would begin to run more important role, and thus parallelization would not be as good. Also, if the number of processors is increased, the performance of the PC-cluster could collapse. Although not tested in this paper, it is expected that for massively parallel works, like for use of over 100 processors, the performance of the PC-cluster will be poor. The supercomputers are still needed to solve very large computing tasks.

Acknowledgements

We would like to thank the Center for Scientific Computing (CSC) to let us access on the PC-cluster.

References

[1] Bärwolff, G., Ketelsen, K., and Thiele, F., "Parallelization of a Finite-Volume Navier–Stokes Solver on a T3D Massively Parallel System," in *Sixth International Symposium on Computational Fluid Dynamics*, (Lake Tahoe, Nevada), Sept. 1995.

[2] Sawley, M. and Tegner, J., "A Comparison of Parallel Programming Models for Multiblock Flow Computations," *Journal of Computational Physics*, Vol. 122, 1995, pp. 280–290.

[3] Message Passing Interface Forum, "MPI: A Message-Passing Interface Standard.," Computer Science Dept., University of Tennessee, Knoxville,TN, 1994.

[4] Siikonen, T., Hoffren, J., and Laine, S., "A Multigrid LU Factorization Scheme for the Thin-Layer Navier–Stokes Equations," in *Proceedings of the 17th ICAS Congress*, (Stockholm), pp. 2023–2034, Sept. 1990. ICAS Paper 90-6.10.3.

[5] Roe, P., "Approximate Riemann Solvers, Parameter Vectors, and Difference Schemes," *Journal of Computational Physics*, Vol. 43, 1981, pp. 357–372.

[6] Van Leer, B., "Flux-Vector Splitting for the Euler Equations," in *Proceedings of the 8th International Conference on Numerical Methods in Fluid Dynamics*, (Aachen), 1982. (also Lecture Notes in Physics, Vol. 170, 1982).

[7] Siikonen, T., Kaurinkoski, P., and Laine, S., "Transonic Flow over a Delta Wing Using a $k - \epsilon$ Turbulence Model," in *Proceedings of the 19th ICAS Congress*, (Anaheim), pp. 700–710, Sept. 1994. ICAS Paper 94-2.3.2.

[8] Siikonen, T. and Pan, H., "Application of Roe's Method for the Simulation of Viscous Flow in Turbomachinery," in *Proceedings of the First European Computational*

Fluid Dynamics Conference, (Brussels), pp. 635–641, Elsevier Science Publishers B.V., Sept. 1992.

[9] Rautaheimo, P., Salminen, E., and Siikonen, T., "Parallelization of a Multi-Block Navier–Stokes Solver," in *Proceedings of the Third ECCOMAS Congress*, (Paris), John Wiley & Sons, Ltd., Sept. 1996.

[10] Rautaheimo, P., Salminen, E., and Siikonen, T., "Parallelization of a Multi-block Flow Solver," Helsinki University of Technology, Laboratory of Applied Thermodynamics, 1997. ISBN 951–22–3416–5.

[11] Rautaheimo, P., Salminen, E., and Siikonen, T., "Parallelization of a Multi-block Flow Solver," *CSC News*, Vol. 9, No. 3, 1997, pp. 15–18.

[12] Kaurinkoski, P. and Hellsten, A., "FINFLO: the Parallel Multi-Block Flow Solver," Helsinki University of Technology, Laboratory of Aerodynamics, 1998. ISBN 951–22–3940–X.