

# Scalable Performance of FLUENT on NCSA IA-32 Linux Cluster

Wai Yip Kwok

National Center for Supercomputing Applications, Champaign, IL 61820, USA  
kwok@ncsa.uiuc.edu,  
WWW home page: <http://www.ncsa.uiuc.edu/~kwok>

**Abstract.** A leading computational fluid dynamics (CFD) software FLUENT is ported to the NCSA IA-32 Linux cluster with Myrinet interconnect. Scripts are written to integrate FLUENT with the open-source PBS scheduling system. Scalable performance of FLUENT is benchmarked with two engineering problems from Caterpillar Inc. and Fluent, Inc. A maximum of 64 processors are used to accommodate up to 10 million cells. The Linux cluster computes more than 2.5 times faster than a SGI Origin2000 supercomputing at NCSA. A speed-up of 7 times is observed when 32 processors are used instead of 2. This scalability suggests that cluster computing can substantially shorten design cycles in industrial design processes. Linux clusters provide a promising means of large-scale CFD simulations at an affordable cost.

## 1 Introduction

Computational fluid dynamics (CFD) is the study of motion, force and physics of fluids (e.g. water, air) using mathematics and computation. Design of many industrial and engineering applications, such as jet engines, sewage treatment plants, and artificial heart valves, depends on clear understanding of fluid motion. Computational technology is appealing as it can bring the cost of design and testing down substantially. However, efficient simulations of complex flow phenomena require intensive computational resources to achieve fast design cycles. Nowadays, many CFD design processes rely on high-performance parallel computers to perform flow simulations.

Traditionally, many high-performance computers are custom-designed and built from scratch to be supercomputers. These machines are powerful but carry a huge cost. They present a cheaper solution in engineering design compared with experimental testing, yet are still very expensive to install and maintain. In the last few years, a new high-performance platform, cluster computers, has emerged and become very important for the business sector. Cluster computers are commodity workstations connected by high-speed network. They cost substantially less than traditional supercomputers because of the cost benefits of commodity technology. Choice of open-source software such as Linux operating system drives the cost further down. Yet rapid development of processor and network performance makes cluster supercomputers as capable as their more expensive

counterparts. Cluster computers also offer flexibility, as use of the machine is taken into consideration in the selection of processors, network, input/output file system, etc.

Although a very promising solution, cluster computers pose challenges for commercial software developers and users. Specific software executables have to be built to make the best use of a particular cluster. Software development has to take into account processors, version of operating systems, and network connection. These components come from different sources and evolve fast. As a result, the deployment of commercial software requires stronger communication between developers and users, and probably more effort from the developers.

In this project, we successfully port a leading commercial CFD software FLUENT [1] to a Linux cluster platform at NCSA. The scalable performance of FLUENT on the cluster computer is analyzed. In Section 2, background information on the Linux cluster and software FLUENT is presented. Technical details involved in the installation of FLUENT are described in Section 3. The scalability test problems are presented in Section 4. Results of the scalability analysis are discussed in Section 5. Concluding remarks are given in Section 6.

## 2 Machine description and parallel-processing features of software

FLUENT has been ported to a distributed-memory IA-32 cluster in NCSA [2]. This computer is based on IBM eServer x330 thin servers. Each server has two 1 GHz Intel Pentium III processors. The cluster runs Red Hat Linux version 7.2 and kernel 2.4.9, and uses both Myrinet interconnect and 100Mbit Ethernet network. For comparison, performance of FLUENT on an NCSA SGI Origin2000 supercomputer is also reported. This machine is a shared-memory SGI Origin2000 supercomputer based on 195MHz MIPS R10000 processors.

Running on multiple-processor supercomputers, FLUENT partitions the computational domain and assigns segments to individual processors. FLUENT also performs cell migration between partitions throughout the run to assure approximately equal workload among all processors.

## 3 Installation of FLUENT on IA-32 Linux cluster

In this section, two main technical challenges overcome in the installation process are described. The first one is the incompatible versions of Myrinet network device drivers in NCSA and Fluent, Inc. At the moment of installation, FLUENT was built on Myrinet GM driver 1.5.1, while NCSA was running GM driver 1.5.2. This problem was resolved when Myricom, Inc. provided a patch to the NCSA GM driver that made it recognizable by FLUENT.

FLUENT requires a list of available processors in order to run parallel jobs. For a small cluster used by a few users, a static list that includes all the machines can be written simply. For a large Linux cluster in NCSA that serves hundreds of

users, however, a distributed job management tool is used to allocate resources to individual jobs. Use of this Portable Batch System (PBS) implies that the list of available processors changes with time. To ensure that FLUENT works smoothly with PBS, a script was written to generate the processor list each time a FLUENT job starts. This script extracts information from PBS about processors assigned to the job and transfers it to FLUENT. It is included in the Appendix together with a sample batch script.

After the above two problems were resolved, FLUENT could be run on the IA-32 Linux cluster and was ready for performance tests.

## 4 Scalability test problems

Two engineering problems are used to test the parallel-processing capability of FLUENT on the NCSA platforms. The first flow simulation is that of an engine exhaust manifold provided by Caterpillar Inc. The second and larger problem is a turbulent flow through a duct provided by Fluent, Inc.

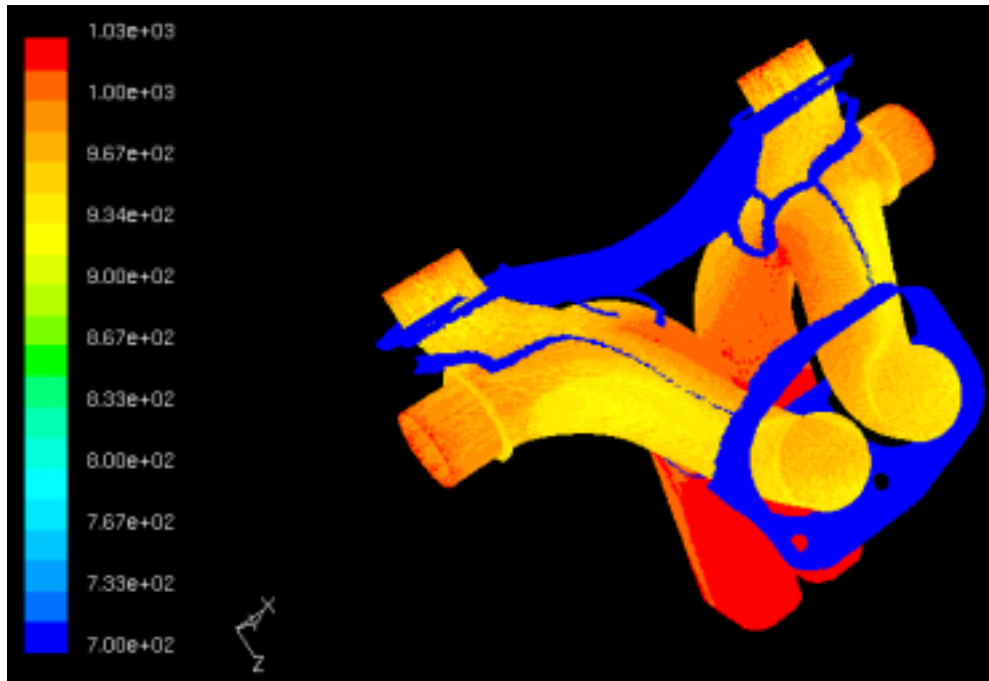
### 4.1 Exhaust manifold in a truck engine

The Core Technologies group in Caterpillar Inc. has tested a steady-state conjugate heat transfer model of a heavy-duty truck engine exhaust manifold. The exhaust gas is assumed to obey the ideal gas law, and turbulence is accounted for using the standard k-epsilon model. The operating conditions simulated are those specified by one of Caterpillar's internal engine test procedures, which is used to assess the long-term durability of the manifold. The boundary conditions are computed using Caterpillar's engine cycle-simulation code. The objective of this simulation is to obtain the temperature distribution in the metal, which is subsequently used in a thermo-mechanical stress analysis of the component. The simulation results show metal temperature ranges from 640K to 950K, while gas temperature can be higher than 1000K (Figure 1). In the fluid the peak Mach number is approximately 0.27 corresponding to a velocity of 201 m/s (Figure 2).

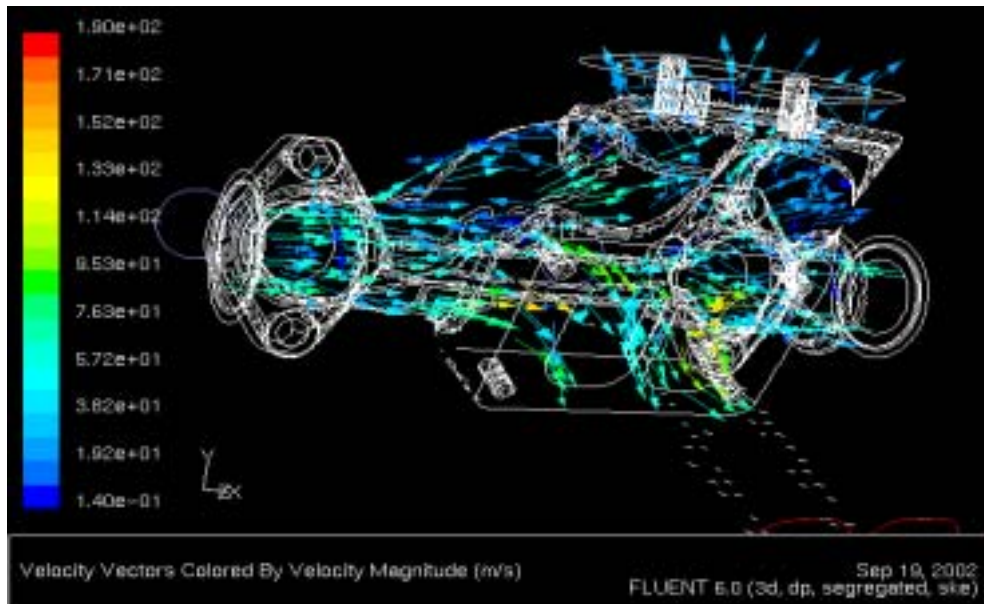
The computational grid consists of about 1.9 million tetrahedral cells. A fine mesh was used to capture the flow accurately in the complex geometry (Figure 3). The size of the data file is about 180 MB.

### 4.2 Turbulent flow in a duct

Fluent, Inc provides the turbulent duct flow simulation problem. The cross-section of the duct changes from a circle at the inlet to a rectangle at the outflow boundary. Although a simpler geometry compared with the exhaust manifold, the chaotic nature of the flow requires intensive computational resources. Simulation results show that much turbulence and swirling occurs at the change of cross-section (Figures 4 and 5). The computation mesh consists of 9.8 million hexahedral cells. The data file has a size of 1GB. The Reynolds-stress model is used for computing fluid turbulence. A segregated implicit solver is used.



**Fig. 1.** Temperature distribution in a truck-engine exhaust manifold. Only part of the manifold is shown. (Courtesy of Caterpillar Inc.)



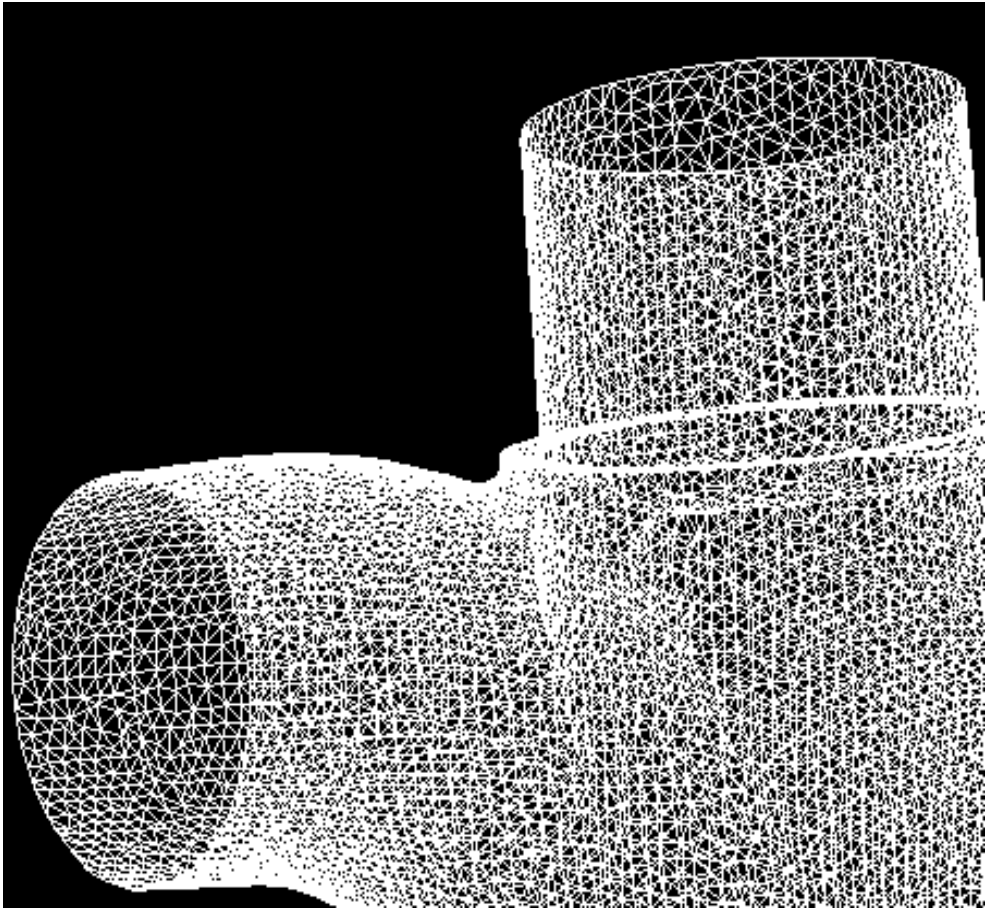
**Fig. 2.** Plot of velocity vectors in a truck-engine exhaust manifold. (Courtesy of Caterpillar Inc.)

## 5 Performance and scalability analysis

The exhaust manifold problem is solved using a number of processors from 2 to 32. Both computer platforms are tested. On the IA-32 Linux cluster, FLUENT provides three connection modes. The socket and network MPI connection modes contain built-in parallel libraries, but make use of the slower 100 Mbit interconnect. The MPICH-GM connection mode utilizes the fast Myrinet connection.

The time required for advancement of flow by 250 time-steps and input/output is shown in Table 1 and Figure 6. Both computing platforms benefit from use of multiple processors, as can be seen in the continuous drop in time when more processors are used. The IA-32 Linux cluster performs about 2.7 times faster compared with the SGI Origin2000, primarily due to the faster processors. Among the different connection modes, the MPICH-GM connection mode brings forth the best performance for large number of processors. The superior performance of Myrinet interconnect (denoted gmpi) can be seen more clearly in Figure 7, where the same data is presented on a log-log scale.

Another measurement of interest is speed-up shown in Figure 8. Speed-up is the ratio of wall-clock time required to complete a calculation using 1 processor to that of an equivalent calculation using a higher number of processors. Due to the limited size of memory in one processor, the simulation requires at least



**Fig. 3.** Computational grid at a joint in the exhaust manifold. (Courtesy of Caterpillar Inc.)

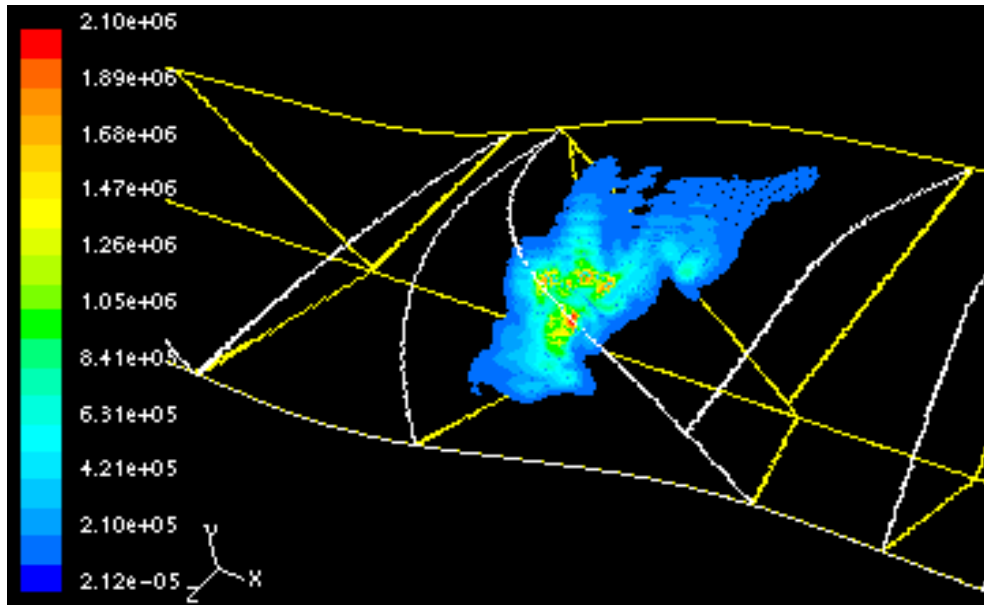


Fig. 4. Turbulent dissipation at the change of cross-section. (Courtesy of Fluent, Inc.)

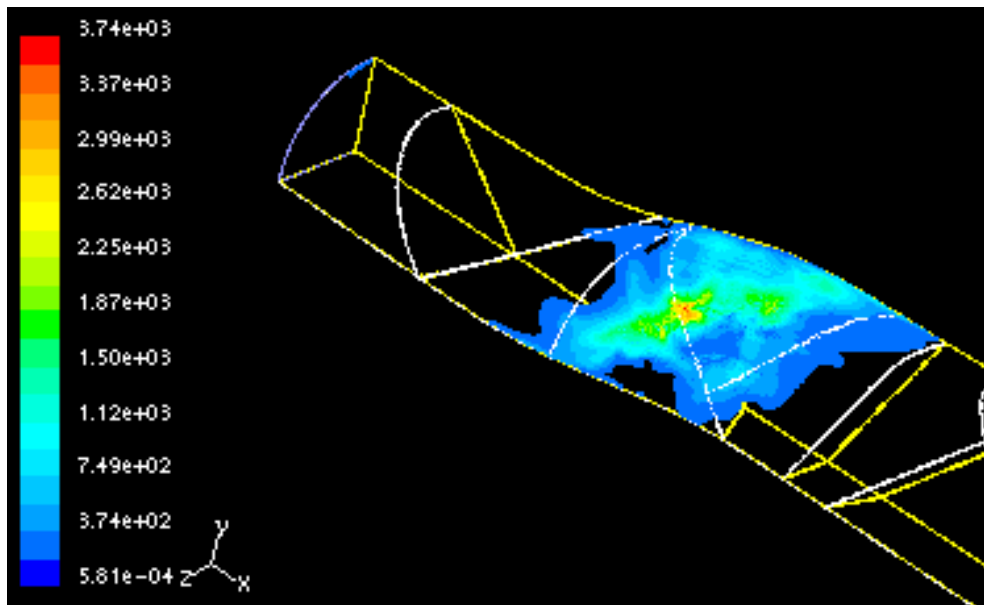


Fig. 5. Magnitude of vorticity at the change of cross-section. (Courtesy of Fluent, Inc.)

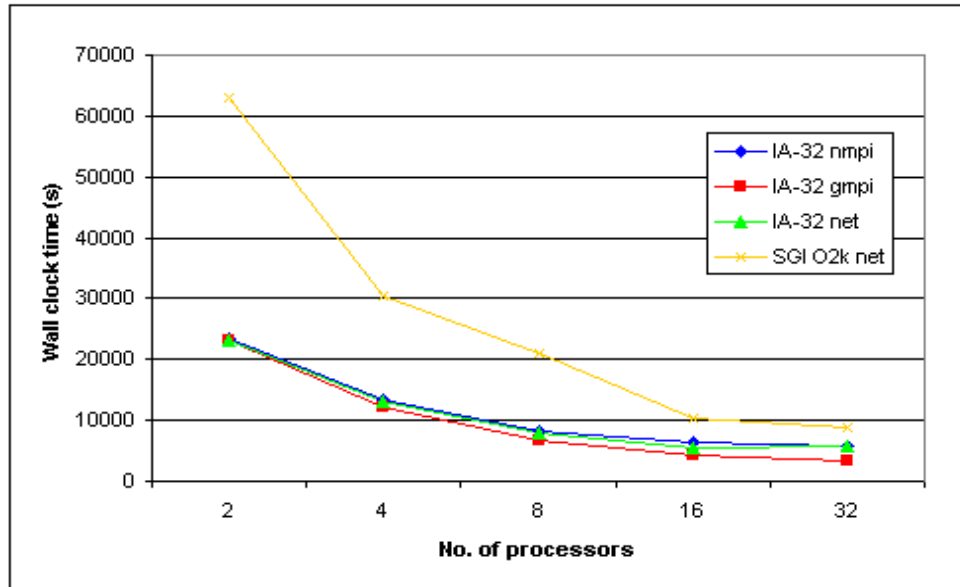
two processors. The wall clock time for the 1-processor run is estimated as two times the wall clock time for the 2-processor run. Using  $P$  processors, the value of speed-up ranges from 0 to  $P$ , where  $P$  is the linear speed-up value assuming no communication overhead among processors. As the number of processors increases, the speed-up deviates more from the linear value due to heavier communication overhead. Both the SGI Origin2000 and the IA-32 Linux cluster with Myrinet connection achieves a speed-up of 14 when 32 processors are used.

Performance and scalability results from the turbulent duct flow problem are shown in Table 2 and Figures 9 to 11. This problem is about 5 times bigger than the exhaust manifold problem, in terms of both the number of cells and the size of the data file. As a result, more processors are used, with a maximum of 64 processors tested on the IA-32 Linux cluster. For a larger problem, the IA-32 Linux cluster is performing three times faster than the SGI Origin2000 (Figure 9). However, the performance scaling drops as the number of processors increases. Doubling the number of processors from 32 to 64 on the IA-32 Linux cluster, the time drops only 23% from 13,000 seconds to 10,000 seconds, far from an ideal value of 50%.

To understand the communication overhead, the simulation time is divided into two parts: time-advancement solver and input/output. The time-advancement solver almost achieves linear scaling. Time per time-step drops 3.9 times from 93 to 24 seconds, when number of processors increases 4 times from 16 to 64. (Figure 10) On the other hand, the input/output has a negative impact on the scaling performance. The time spent on input/output increases as more processors are used. Combined with the fact that the time spent on the actual computation drops, the portion of time spent on input/output increases drastically (Figure 11). For the run with 64 processors, more than 50% of time is spent on input/output. This result is not surprising, as FLUENT uses only one processor for input/output. At the beginning and end of the simulation, all the processors have to communicate with this single processor. This input/output mechanism is a bottleneck for scalable performance. Therefore, for any given simulation, there is an optimum number of processors that strikes a balance between faster advancement but slower input/output. A larger problem in which time-advancement computation dominates can benefit from use of more processors.

**Table 1.** Simulation time in seconds of 250 time-steps and input/output for exhaust manifold problem.

Number of processors	IA-32 nmpi	IA-32 gmpi	IA-32 net	SGI O2k net
2	23,548	22,983	23,195	63,055
4	13,430	12,063	12,939	30,294
8	8,125	6,765	7,778	20,889
16	6,468	4,175	5,596	10,300
32	5,906	3,274	5,741	8,782



**Fig. 6.** Simulation time of 250 iterations and input/output for exhaust manifold problem.

**Table 2.** Simulation time in seconds of 200 time-steps and input/output for turbulent duct flow problem. For the IA-32 Linux cluster, the superior MPICH-GM Myrinet connection mode is used.

Number of processors	IA-32 gmpi	SGI O2k net
16	22,363	68,182
32	13,015	38,255
64	9,874	

## 6 Conclusions

A leading commercial CFD software FLUENT is ported to the NCSA IA-32 Linux cluster. Parallel-computing capability of FLUENT is tested on this platform with two test problems: an exhaust manifold simulation from Caterpillar Inc. and a turbulent duct flow simulation from Fluent, Inc.

Parallel computing is shown to be critical in both tests, as the problems can only be accommodated in multiple processors. A maximum of 64 processors are

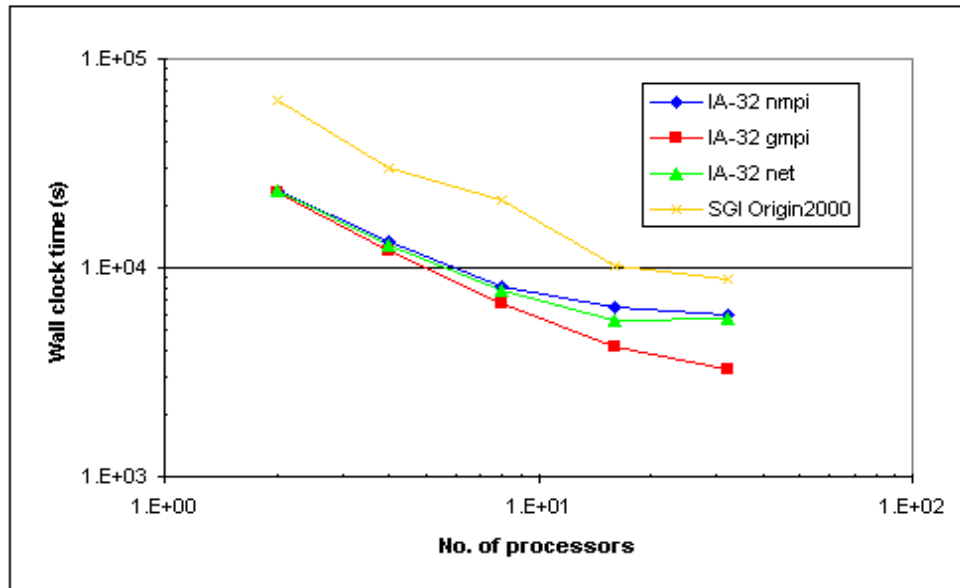


Fig. 7. Simulation time data in Figure 6, presented on a log-log scale.

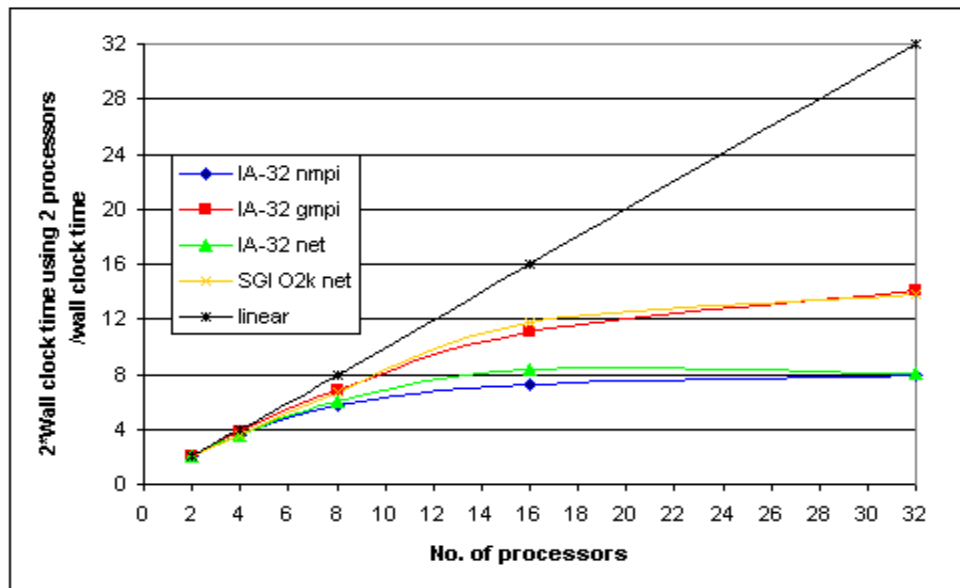
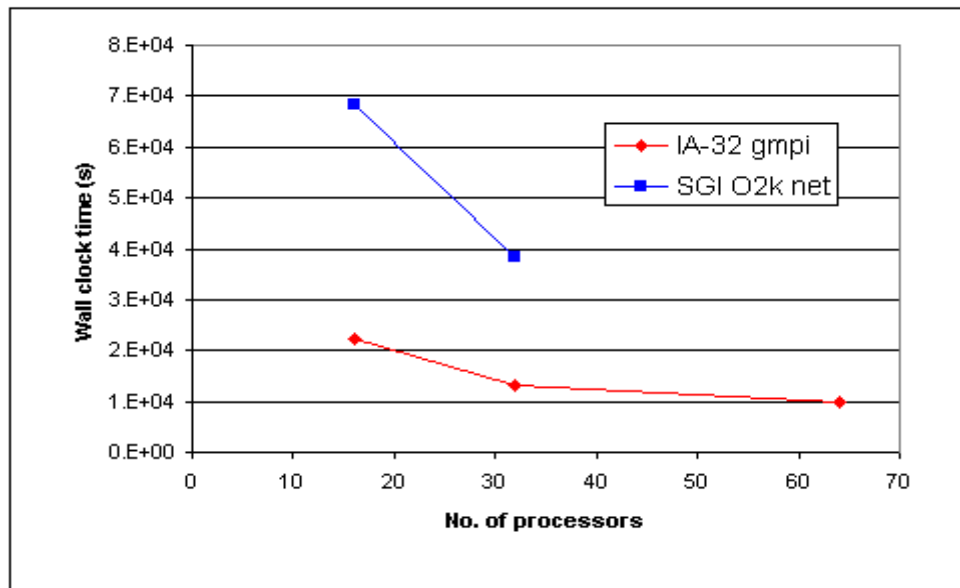


Fig. 8. Speed-up based on ratio of simulation time using two processors to that using more processors. Simulations include 250 iterations and input/output.



**Fig. 9.** Simulation time of 200 iterations and input/output for turbulent duct flow problem.

used. The IA-32 Linux cluster computes more than 2.5 times faster than the SGI Origin2000, mainly due to the faster processors. Speed-up of 7 is observed when 32 processors are used instead of 2. Time profiles of simulations suggest that the computation scales very well, and the main source of parallel computing overhead is the file input/output. This high performance of Linux cluster, combined with the relatively low cost of building the machine and the open-source nature of operating system, provides a promising solution to large CFD simulations at an affordable cost.

## 7 Acknowledgements

The author would like to thank Robert McDavid at Caterpillar Inc. for supplying and explaining the truck-engine exhaust manifold problem. Tom Tysinger and Richard LaRoche at Fluent, Inc. provided assistance in setting up the parallel version of FLUENT on the IA-32 Linux cluster. The efforts of the following people in NCSA are also acknowledged. Brian Kucic and Jay Alameda coordinated our cooperation with Caterpillar Inc. and FLUENT, Inc. Galen Arnold and Karen Fernsler provided vital help in installation of the software as well as making FLUENT work with the Myrinet connection and PBS scheduling system. The consulting office provided a small Linux cluster for installation testing.

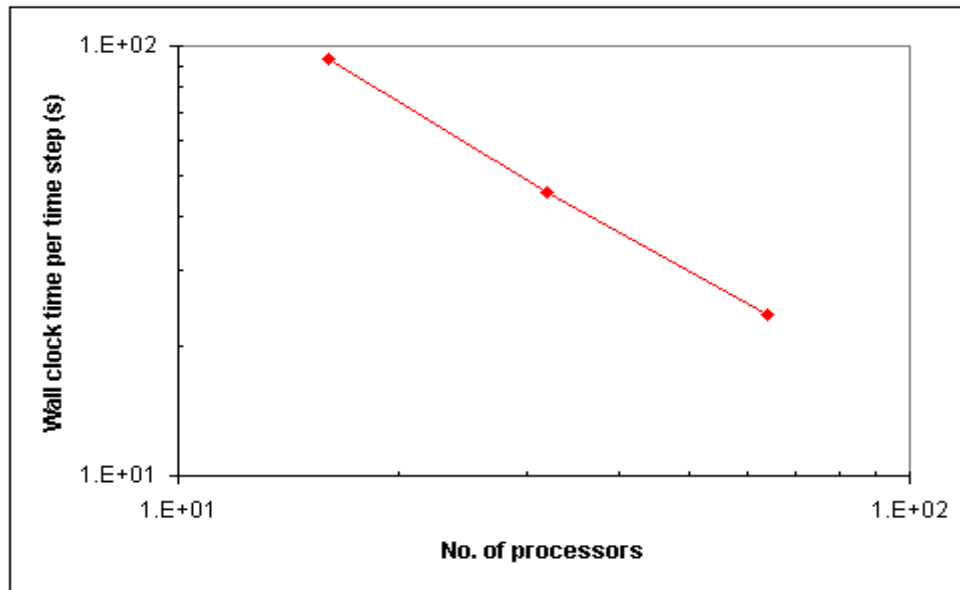


Fig. 10. Scaling of time-advancement computation on the IA-32 Linux cluster, plotted on a log-log scale. Time spent on input/output is excluded.

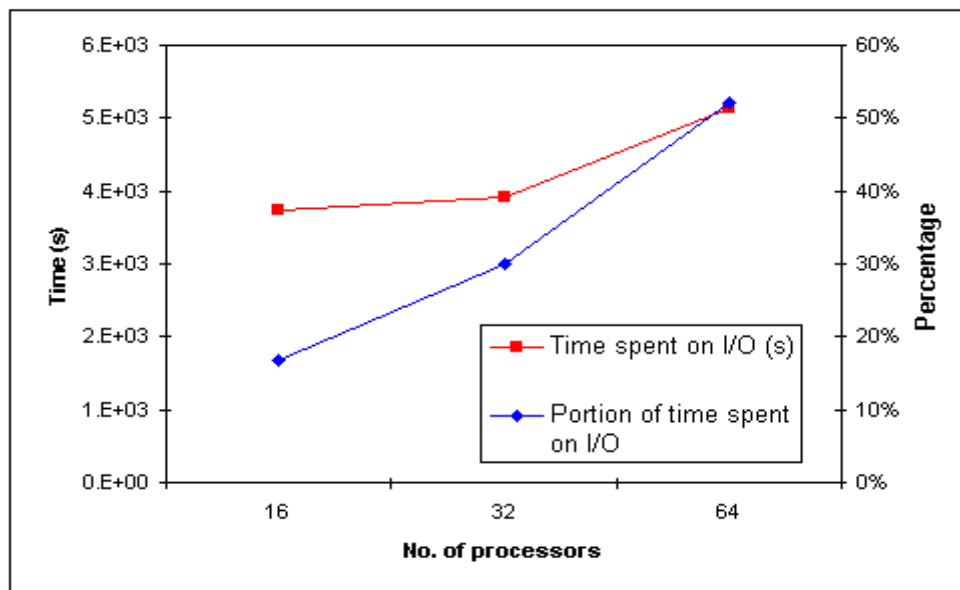


Fig. 11. Scaling of input/output on the IA-32 Linux cluster.

Finally, the author had constructive discussions with Faisal Saied at the initial stage of learning how to run parallel jobs with FLUENT.

## References

1. Fluent Inc. homepage. <http://www.fluent.com>. 2003.
2. NCSA homepage. <http://www.ncsa.uiuc.edu>. 2003.

## A Appendix: Scripts for generation of machine file and submission of batch jobs

A Perl script `makeconf.pl` is written by Galen Arnold to create the machine file that FLUENT parallel jobs need:

```
#!/usr/bin/perl

$pbs_nodefile= $ARGV[0];
$ppn= $ARGV[1];

$nodecount= `cat $pbs_nodefile | wc -l`;
#
# coerce to numeric
$nodecount *= 1;

$nodes= `cat $pbs_nodefile| sort|uniq`;
(@nodes)= split /\n/, $nodes;

printf "$nodecount\n";

foreach $node (@nodes)
{
    printf "$node 4\n";
    if ($ppn == 2)
    {
        printf "$node 5\n";
    }
}
```

The script generates machine files `/.gmpi/conf` with the format

```
2
cn213 4
cn214 4
and
```

14

4

cn213 4

cn213 5

cn214 4

cn214 5

for 1-processor-per-node and 2-processors-per-node cases, respectively. In both files, the number of processors is indicated in the first line. Subsequent lines point out the nodes and the GM port numbers. The script is run right before the simulation starts, as indicated in the following script for batch job submission.

```
#!/bin/sh
#PBS -l walltime=10:00:00,nodes=2:ppn=1:prod
#PBS -q ncsa
#PBS -V
#PBS -N fluentgmpi
#PBS -o /u/ncsa/kwok/fluent_files/caterpillar_problem/log_dir
#PBS -j oe
#PBS -A acp

# use storage node and NFS, create temporary work directory
export SCR='set_SCR'
if [ "$SCR" != "" ]; then
    cd $SCR
fi

# copy relevant files from home directory and mass storage to
# temporary work directory
cp /fluent_files/caterpillar_problem/* .
msscnd "cd fluent6_benchmarks/caterpillar_problem, mget c12*";

# use local scratch instead of storage node and NFS
cd /scratch/$PBS_JOBID
vmirun "cp /fluent_files/caterpillar_problem/* ."
vmirun "msscnd "cd fluent6_stuff/caterpillar_problem, mget c12*""";

# use GPFS instead of storage node and NFS
#cd /gpfs/$PBS_JOBID
#cp /fluent_files/caterpillar_problem/* .
#msscnd "cd fluent6_benchmarks/caterpillar_problem, mget c12*";

np='wc -l < $PBS_NODEFILE' # number of lines = number of procs
np='expr $np' # remove spaces
nodes='sort -u $PBS_NODEFILE | wc -l' # unique lines = number of nodes
nodes='expr $nodes' # remove spaces
```

```
ppn='expr $np \/ $nodes' # calculate procs per node

# set limit for virtual memory
ulimit -v '/usr/local/sbin/memory_limit.pl $ppn 0'
vmem='ulimit -v'
echo "virtual memory limited to $vmem Kbytes"

# create /.gmpi/conf that contains information about nodes allocated.
# Fluent reads this file.
rm -f /.gmpi/conf
./makeconf.pl $PBS_NODEFILE 1 > /.gmpi/conf

# Run 3-dimensional simulation
fluent 3ddp -g -t2 -pgmpi -i input >& output

rm -rf *.gz

# tar output files up and store it in mass storage
tar cvf c12egr_cfd_5.$PBS_JOBID.tar *
msscmd "cd fluent6_benchmarks/pt_results, put c12egr_cfd_5.$PBS_JOBID.tar"
```