Using a Parallel CFD Code for Evaluation of Clusters and MPPs

Oleg Bessonov
Institute for Problems in Mechanics
of the Russian Academy of Sciences
101, Vernadsky ave., 119526 Moscow, Russia
bess@ipmnet.ru

Dominique Fougère, Bernard Roux
Laboratoire de Modélisation et Simulation
Numérique en Mécanique, L3M–IMT
La Jetée, Technopôle de Château-Gombert
13451 Marseille Cedex 20, France
{fougere,broux}@l3m.univ-mrs.fr

Abstract

We consider some methods of evaluating parallel performance on clusters and MPP platforms. These methods are based on using a parallel code for the numerical simulation of 3D incompressible viscous flow in a cylindrical domain. Details of the efficient parallelization of this code are discussed for distributed memory computers with relatively slow communication links. Results of measurements of parallel performance are presented for several computers of different architectures, with the number of processors used from 1 to 16. Additionally, direct measurements of basic performance indicators are considered, such as serial performance and speed of interprocessor communications. Finally, the comparative performance analysis is presented for these two sorts of measurements.

1. Introduction

Correct measurement of computer performance is a very important task. For traditional serial computers, there exist some standardized benchmarks. The most known of them (at least for scientific and engineering applications) is the SPEC CPU2000 benchmark suite [1]. However, this suite is a licensed product and therefore not available for typical benchmarkers. Also, it consists of several "heavy" benchmarks (e.g. 14 separate programs in SPECfp_base2000 set) that complicates a measurement exercise. For this reason, using some application program as an "approximation" seems to be more attractive. A good sort of such approximating benchmark can be the 3D CFD code considered here. The previous experience of measurements with this code on many computer systems demonstrates the reasonable correlation with published SPECfp_base2000 results.

The natural extension of a benchmarking methodology for parallel computers is to use the same code in the parallelized form. The considered code is realistic and not easily parallelizable, therefore it can serve as a sensitive indicator of parallel performance of distributed memory computers with relatively slow communication links. At last, grid dimensions of discretized equations can be varied in order to represent different problem sizes, as well as to evaluate scalable problems. It should be noted also, that there is no general-purpose parallel benchmark in the SPEC collection yet.

In order to be more representative, the considered code must be implemented in the most efficient manner, both in serial and parallel forms. Due to high computational potential of modern microprocessors and quality of compilers, optimizing the single-processor performance is not a serious problem now. In contrast, the communication speed of interconnection networks is usually much lower than necessary to exploit fully the intrinsic parallelism of numerical algorithms. With the rapid development of superscalar microprocessors, the gap between computational speed and interconnection capacity becomes even wider. Therefore, much attention should be paid on the development of numerical methods and parallelization algorithms that are economical from the point of view of data exchanges.

For simulations of flows in 3D regular domains, the Finite Difference (FDM) and Finite Volume (FVM) methods have proved to be very efficient [2]. Straightforward implementations of these methods normally use a substantial fraction of "explicit" time integration codes, that don't need data exchanges during the computational steps. Only a small part of data, the separator (boundary) planes between subdomains, belonging to different computational nodes, need to be transferred.

Unfortunately, a realistic simulation of incompressible viscous flows can't be performed by pure explicit code due to timestep constraints, especially for flows with highly diffusive processes. The implicit methods should be incorporated, that involve solving 3-diagonal linear systems in every spatial direction. Another numerical difficulty of incompressible flow simulation arises from the physical nature of

0-7695-1926-1/03/$17.00 (C) 2003 IEEE

pressure. The pressure Poisson equation must be solved globally in the entire domain on every timestep. In order to avoid expensive iterative methods, the direct Fourier method is often used that involves Fast Fourier transfer (FFT) steps and 3-diagonal sweeps. Parallelization of FFT requires full data exchange between nodes and is therefore very uneconomical. In order to reduce amount of data exchanges, several approaches have been suggested by different authors [3, 4]. However, these algorithms are either much less accurate than necessary, or less economical than the Fourier method.

The algorithm considered here is based on the previous work [5] with the extension to multidimensional decomposition of a computational domain. To avoid excessive data exchanges, a new method for solving Poisson equation has been developed, based on a cyclic reduction of arising linear systems in frame of the FACR approach [6, 7]. As a result, the algorithmically and numerically economical implementation has been obtained for the number of processors up to 16.

The developed parallel code has been used for benchmarking several distributed memory machines – massive parallel computers (MPP) and multiprocessor node (SMP) clusters. These tests have been complemented with direct measurements of basic performance indicators – single-processor computational performance and speed of interprocessor communications. Some previous work has been performed in this area [8]. Current analysis is based on the evaluation of parallelization efficiency of the presented code for different number of processors (2, 4, 8, 16) and problem sizes in comparison with the measured computational and communication characteristics.

2. Numerical method and basic parallelization of the algorithm

The numerical problem considered here is the solution of 3D non-stationary Navier-Stokes equations in Boussinesq approximation for incompressible viscous flow in a cylindrical domain. This sort of simulation is used in crystal growth applications, like semiconductor melt flows in Czochralski apparatus [9] or processes above the surface of a growing aqueous-soluble crystal, and in modeling of natural convection in space experiments [10].

The velocity-pressure formulation and FVM discretization are employed, with the decoupled solution of momentum, pressure and temperature equations using the Fractional step (pressure correction) method. The time integration scheme is partially implicit, with the implicit treatment of the most critical terms using ADI (Alternating directions implicit) approach. The pressure Poisson equation is traditionally solved by efficient Fourier method, that involves FFTs in two spatial directions and 3-diagonal systems solutions in the last direction. This numerical method is fully direct and doesn't involve costly iterative steps.

From the point of view of data processing, the computations are organized by the following way:

- The cylindrical computational domain is considered as a 3-dimensional array (φ, z, r). All computations are performed in the most efficient manner, using the 1-st index of array as the innermost one in Fortran loops. An iteration of the outer loop can be considered as computations in a plane (φ, z), that is moving in the direction r as a "frontal plane of computations" [5].
- All explicit parts of the algorithm are trivial and simply form 2D loops within this plane of computations.
- The implicit part is split into solving 3-diagonal linear systems in all 3 directions (φ, z, r), each consisting of 2 sweeps (forward and backward) in corresponding direction. All sweeps in the directions φ and z involve processing of data located within 2D plane of computations. Sweeps in the direction r look like a slow motion of this plane in forward or backward direction.
- The Fourier method comprises FFTs in the directions φ and z, that again involve processing within a plane of computations, and solving 3-diagonal systems in the direction r, implemented as for the implicit step.

The parallelization method is based on the splitting a computational domain in the last 2 directions, r and z. The current implementation includes the following variants: 1×1, 2×1, 4×1, 4×2 and 4×4 (Fig. 1), from 1 to 16 CPUs (with a possible extension to 8×4 for 32 CPUs).

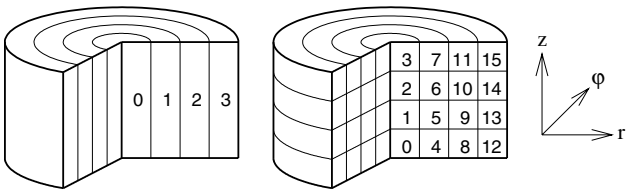

Figure 1. 1D and 2D decompositions of a computational domain.

Consider first the parallelization method for 1-dimensional splitting (Fig. 1, left).

- Computational domains are overlapped, with one neighbour's plane (2D array of data) stored in a node for each boundary. This is necessary for calculation of some terms in discretized equations.
- All parts of the numerical algorithm involving calculations only within a plane of computations (φ, z) are

processed independently in each node and don't need data exchanges. These parts include all explicit steps, implicit sweeps in the directions φ and z, and FFT transforms in these directions. Data exchanges between adjacent processor nodes are performed only between these steps (when necessary), transmitting full 2D arrays of data.

- Sweeps in the direction r can't be parallelized in frame of the conventional 3-diagonal solver. Instead, the twisted factorization is used for 2 processors, or two-way parallel partition method [11, 5] for 4 or more processors (Fig. 2). These methods employ more complicated way of Gauss elimination procedure, that can be done simultaneously in all subdomains. The modified sweeps are performed as frontal planes of computations, with exchange of full 2D data arrays between adjacent nodes when necessary. Parallel solution of 3-diagonal system on 4 processors requires 3 to 6 such exchanges (depending on the sort of 3-diagonal matrix).

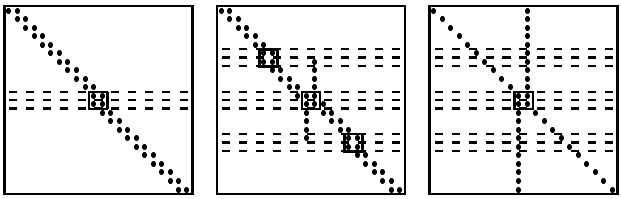

Figure 2. Illustration of the 2-way method of parallelization for 2 (left) and 4 (central, right) processors.

As a result, the parallelized numerical method remains algebraically identical to the sequential one. This is different from the iterative domain decomposition approach, when the efficiency depends on convergence properties of the algorithm and can be violated by the splitting.

The above method has demonstrated the good parallelization efficiency [5, 8]. However, the increased complexity of solving 3-diagonal systems limits the number of processors by 4, at most 8. The natural way to overcome this limitation is to extend decomposition into the 2-nd spatial direction (z). This would increase the limitation to $4 \times 4 = 16$ processors (Fig. 1, right)..

The parallelization procedure and data distribution for the direction z are similar to those of the direction r for almost all steps of the algorithm. However, FFTs in the direction z can't be efficiently parallelized because multiple transmissions of all processed data are required. The way to reduce the number of data transfers is to split a computational domain in the remaining spatial direction (φ) and rearrange data for this operation. Figure 3 illustrates this rearrangement (blocked transposition) for 4 subdomains, when $3/4$ of all data are involved into an exchange.

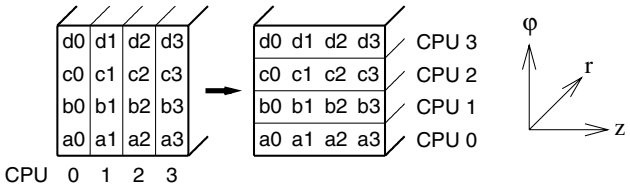

Figure 3. Blocked transposition for parallelization of FFT in the direction z.

The following steps of the algorithm – FFT in the direction z, 3-diagonal sweeps in the direction r, and inverse FFT in z – are performed on rearranged data independently in each processor. Finally, another transposition is required in order to return resulting data into the initial distribution. As a result, the parallelized procedure for the Fourier method would look as follows:

FFT(φ), transposition,
FFT(z), 3-diag(r), FFT(z),
transposition, FFT(φ)

3. Extension of the parallelization method and techniques

The described procedure requires an exchange of full 3D data arrays between processors, while for the other steps of the algorithm only 2D boundary planes must be transferred. Since the speed of interprocessor communications of modern parallel computers is much lower than their computational performance, this step would involve long delays and dramatically reduce the efficiency of parallelization.

In order to lower the required amount of data transfer, a new method for solving pressure Poisson equation has been developed. The method is employed to 2D linear systems obtained after performing FFTs in the direction φ. It is based on the FACR (Fourier analysis with cyclic reduction) approach [6, 7] and consists of 3 stages: cyclic reduction of the original matrix, solution of the reduced linear system by the Fourier method, and substitution of results into the remaining equations.

The method of cyclic reduction itself is used for simplifying 3-diagonal and blocked 3-diagonal linear systems. One iteration of this method halves the number of equations in a system using the way illustrated below. Let us consider a 3-diagonal system:

$$
\begin{aligned}
x_{i-2} + A\,x_{i-1} + \quad x_i \qquad\qquad\quad &= y_{i-1} \\
x_{i-1} + A\,x_i + \quad x_{i+1} \qquad\quad &= y_i \\
x_i + A\,x_{i+1} + x_{i+2} \quad &= y_{i+1}
\end{aligned}
$$

If we multiply every second equation (i-th in this case) by $-A$ and add two adjacent equations to it, we obtain the reduced linear system:

$$x_{i-2} + (2 - A^2)\,x_i + x_{i+2} = y_{i-1} - A\,y_i + y_{i+1}$$

Substituting $A^{(1)} = 2 - A^2$ and $y_i^{(1)} = y_{i-1} - A\,y_i + y_{i+1}$, we obtain the system of equations of the same type and can therefore employ the cyclic reduction procedure again. After several iterations, the resulting system can be solved by any convenient method, with the following back-substitution steps in order to find the remaining unknowns.

In our case, the blocked 3-diagonal system is solved, where A is a 3-diagonal matrix itself. As a result, the new matrices $A^{(1)}$, $A^{(2)}$ etc are no more 3-diagonal. However, they can be factored into simple 3-diagonal matrices, and the resulting systems can be resolved by several repetitions of 3-diagonal algorithm.

Every iteration of the cyclic reduction increases the complexity of the numerical algorithm and sophisticates the data exchange pattern. As a compromise, the 2-step cyclic reduction scheme has been chosen, with the 4-fold reduction of matrix size and amount of data exchanges in Fourier method. Despite the slight increase of data transfers in another parts of the algorithm, the resulting amount of transmissions is now on the reasonable level and doesn't influence so much the efficiency of parallelization.

Now consider some technological improvements of parallelization methods. First of them concerns the structural complexity of the parallel code (in comparison with the sequential one). This complexity arises in particular from the increased number of sorts of subdomains with a variety of boundary conditions: external (physical), and internal (between subdomains). In order to simplify the code flow, the alternating numbering scheme is proposed. If, for example, the domain is split into 4 subdomains in some direction, data elements (data points) in this direction are numbered in alternating order (Fig. 4). Due to this, codes in every two adjacent nodes (0 and 1, 2 and 3) become more unified, and the total number of different boundary conditions is reduced. More important, all data exchanges are performed uniformly in all nodes. Additionally, the alternating numbering scheme naturally corresponds to the two-way parallel partition method for solving 3-diagonal linear systems.

The next point is a choice of communication library. The most standard one is the MPI. Unfortunately, some parallel systems may lack a MPI implementation at all, or may offer more efficient option like SHMEM, GM or MPL. For this reason, the library-independent approach has been chosen, with a set of intermediate data exchange routines used instead of MPI. All library-specific calls are encapsulated within these routines. As a result, a parallel application program becomes system-independent. In order to adapt to any new communication protocol, only a small set of routines must be rewritten. Sometimes, there exist incompatibilities in different implementations of the same library, or some compiler problems, and the library-independent approach is useful in this case.

Figure 4. Standard (left) and alternating (right) numbering schemes.

This approach also allows to accomplish some specific optimizations of data exchanges without modification of application code, such as splitting blocks to be transferred, or regulating duplex mode of transmission by some way. Another thing necessary for parallel optimization is the renumbering (remap) of allocated processor nodes, that can be important for better adaptation of a parallel computer topology (SMP-nodes, 2D-grids, NUMA, heterogeneous architectures etc) to the structure of an algorithm. For example, in case of two-dimensional decomposition on Fig. 1 (right) processors in a bi-processor SMP-node may be assigned either to two horizontally adjacent subdomains (e.g. subdomains 0 and 4, 1 and 5, etc), or to two vertically adjacent ones (0 and 1, 4 and 5, etc), depending on the resulting efficiency.

Up to now, the intermediate communication routines have been adapted to the following protocols: NX (Intel i860), Parix (Parsytec), PVM, MPL (IBM SP), SHMEM (Cray T3E, SGI) and MPI in different incompatible implementations (MPIch, LAM etc).

4. Evaluation of basic computational and communication performance

Apparently, the efficiency of parallelization depends primarily on the single-processor performance of computational code, as well as on the speed of interprocessor communications. Generally, the ratio of the computational performance to the communication speed determines the important parameter, that monotonically influences this efficiency.

For measuring the computational performance, we use the single-processor variant of the same 3D CFD code, with the problem size 70 MB (grid size $128 \times 64 \times 92$). The

MFLOPS rate (for 64-bit arithmetic) achieved by this code is used as a performance indicator. All measurements on SMP nodes are performed in single-program and multiple-program runs, with the number of copies corresponding to the number of processors. The latter measurements account shared memory conflicts and represent the real-life situation when SMP nodes are fully loaded with processes.

The results of these measurements for several typical platforms and different compilers are presented in Table 1 and Fig. 5 for single-program and dual-program runs. These platforms are widely used for building inexpensive PC clusters.

Table 1. Program execution speed in single-program and dual-program runs (MFLOPS).

processor	compiler	MHz	1-prog	2-prog	ratio
Pentium-4-1800	Intel	1800	575.0	–	–
Athlon-MP1800+	PGI	1525	419.5	316.2	75.4 %
Athlon-MP1800+	GNU	1525	362.7	294.1	81.1 %
Athlon-MP1200	GNU	1200	309.2	266.6	86.2 %
Pentium-III-1000	GNU	1000	128.1	102.4	80.0 %
Pentium-III-550	GNU	550	97.5	84.5	86.7 %

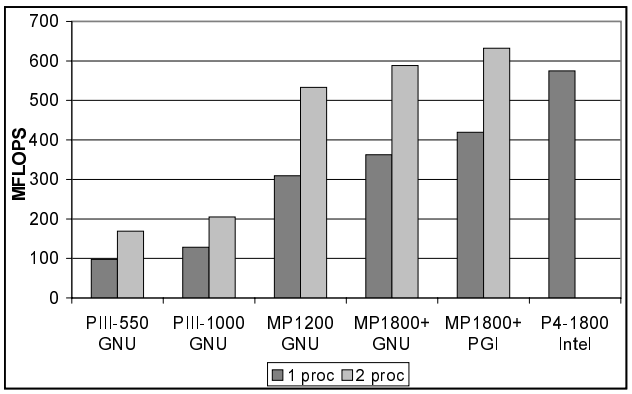

Figure 5. Serial performance of several typical platforms.

It can be seen that the execution speed of each process in a dual-program run is degraded on faster configurations (relative to a single-program run) due to the memory throughput saturation. Because of this, the performance superiority of the PGI compiler over the GNU (on the MP1800+) becomes lower, from 15.7 % for the single-program run to 7.5 % for the dual-program run. Similarly, the difference between the MP1800+/1525 and MP1200 (using the same inefficient GNU compiler) is reduced from 17.3 % to 10.3 % (while the frequency difference is 27.1 %).

Comparisons with Intel processors show that the old dual Pentium-III server platforms (considered before as candidates for building clusters) demonstrate disappointing performance level. On the other hand, the new Pentium-4 processor with RAMBUS memory happened to be 37 % faster than the Athlon-MP. Comparing to the dual-program run of the process on the Athlon-MP, it is 82 % faster, giving for the single Pentium-4 some 91 % of the throughput of the dual Athlon-MP system on such sort of problems. Partly this result was achieved by using the new SSE2 floating point unit supported by the Intel compiler. Without the SSE2 option, the Pentium-4 overruns the Athlon-MP CPU by only 15 %. Note however that the Pentium-4 performance is often non-stable. It depends on many factors and is very sensitive to the compiler's quality and programmer's experience. For some portion of the reference CFD code, the observed speed of the Pentium-4 processor happened to be lower than that of the Athlon-MP.

The communication speed was measured by transferring large arrays of data (32 – 64 KB) between processors in one-way and duplex modes. This corresponds to the typical size of 2D boundary planes to be exchanged between subdomains for most cases. For such block size, the near-asymptotical data transfer speed is usually achieved for all considered communication environments.

Two sorts of test programs have been used: the public domain "MPI Performance Test Suite" from `http://parallel.ru/ftp/tests/` (programs "`transf1`" and "`transf2`"), and the custom test program. The latter employs more realistic method when arrays to be transferred are shifted in memory at every iteration in order to avoid misleading cache effects. For this reason, their results are sometimes lower than that of the first test method, especially in intra-node shared memory measurements.

In order to simulate real-life situation when every CPU in a node communicates with its counterpart in another node, twin inter-node communications have been tested also (in duplex transfer mode).

The results of these measurements (in MB/s) on dual Athlon-MP cluster for different sorts of interprocessor communications are presented in Table 2 and Fig. 6. Data for duplex and twin-duplex modes are given in the table for every single transfer flow, therefore the total throughput of these transfers is two and four times higher (respectively).

Results for 100-Mbit Ethernet auxiliary network are given for the comparison and evaluation of driver quality. It can be seen that efficiency of 100-Mbit implementation is high: 86 % for one-way transfers, and 76 % for duplex

Table 2. Speed of communication between processors (in MB/s).

program	mode of transfer	intra-node (shared memory)	inter-node Dolphin SCI	inter-node Gigabit Ethernet	inter-node 100 Mbit Ethernet
"transf1"	one-way	500-600	200	56	10.8
"transf2"	duplex	140	120	36	9-10
custom	one-way	240	160	53.5	10.8
custom	duplex	120	105	34.5	9.5
custom	twin-dplx	–	88	28	–

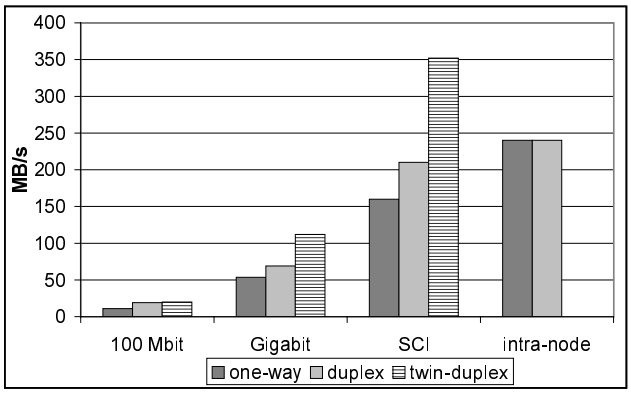

Figure 6. Communication throughput of different interconnects in MPI.

transfers (relative to 2×100 Mbit/s theoretical peak owing to hardware duplex support).

For Gigabit Ethernet, efficiency is lower and varies from about 45 % (one-way) to 29 % (duplex, relative to 2×1000 Mbit/s peak). For twin-duplex transfer, total throughput achieves $4 \times 28.0 = 112$ MB/s, i.e. 45 % of the $2 \times 1000/8 = 250$ MB/s peak value.

These moderate results can be explained partly by a poor implementation of software levels (driver and MPI routines), supposedly involving redundant data movements between buffers. Some limitation can be imposed by the very nature of Gigabit Ethernet protocol (headers, block lengths, delays etc), TCP/IP inefficiency, and also by the limited throughput of the PCI64/33 port to which an Ethernet adapter is connected (266 MB/s peak rate).

The best inter-node results are obtained for Dolphin/Scali/SCI links plugged into the advanced PCI64/66 ports with the extended throughput (533 MB/s peak rate). For twin-duplex transfers, total throughput achieves $4 \times 88 = 350$ MB/s. This benefits from the hardware duplex feature of the SCI interconnects, and reaches about 65 % of the peak rate of the PCI64/66 port.

Intra-node transfers seem to be not efficient as they could be. The main reason of this inefficiency is seen from the comparison of the results for "transf1" and custom (one-way) test programs: in the last case data are transmitted between 2 processors through the shared memory, while in the first case they can arrive to the interprocessor bus directly from the processor caches. Also, a software level (MPI) implementation could be the reason. Another implementations of MPI libraries will be evaluated for intra-node communications in a future.

5. Evaluation of parallel performance

During the last years, many new parallel machines have appeared, including the novel class – multiprocessor node (SMP) parallel computers and clusters. Combining several processors in a single node with common shared memories allows to isolate traffic between neighbour processors within this node, thus reducing inter-node communications. Also, the speed of intra-node exchanges is usually several times higher due to "directcopy" transfer in memory.

The presented parallel code has been used for evaluating parallel performance of several computers, mainly of this new class, in order to reveal their communication behaviour and applicability to this class of numerical problems. Additionally, an investigation of single processor performance and communication network characteristics has been performed, using the methodology described in the previous section. Main characteristics of all analyzed computers and some results of this investigation are presented in Table 3.

The MFLOPS results are presented for the problem size 70 MB (grid $128 \times 64 \times 92$) in multiple-program runs. The communication speed was measured in duplex mode. Both intra-node and inter-node exchanges are shown (top and bottom numbers, respectively). When appropriate, measurements were performed in two regimes: heavy (multi-duplex), when every CPU in a node communicates with its counterpart in another node, and light (simple duplex), when only one pair of processors communicates without competition (these results are shown as ranges).

The communication-to-computation speed ratios are also presented as "invariant" characteristics of compared parallel systems.

Table 4 and Fig. 7 present the parallelization efficiency results for 2 problems: of the fixed size (70 MB total), and the scalable one (70 MB per processor). The size of the biggest 16-processor scalable problem exceeds 1 GB (grid $256 \times 256 \times 184$).

Table 3. Characteristics of the analyzed parallel computers.

parallel platform and interconnect	CPUs per node	CPU cache size	theor. vs. real MFLOPS	comm. duplex MB/s	ratio MB/s to MFLOPS
IBM SP3-375 Colony switch	16	8M	1500 297	80-175 –	0.27-0.59 –
Alpha 21264-667 Myrinet	2	4M	1333 347	90 41-73	0.26 0.12-0.20
Intel PIII-550 2×Ethernet100	2	512K	550 84.5	39 5.8-10	0.46 0.07-0.12
AMD Athlon-MP Ethernet1000	2	256K	3050 308	120 28-35	0.39 0.09-0.11
AMD Athlon-MP Dolphin/SCI	2	256K	3050 308	120 88-105	0.39 0.29-0.34

Table 4. Parallelization efficiency (%) for the fixed and scalable problems.

parallel platform	fixed size problem				scalable problem			
	2	4	8	16	2	4	8	16
IBM SP3-375	98.9	98.0	105.5	102.3	96.2	86.4	79.6	71.5
Alpha cluster	91.8	82.4	83.5	75.2	90.5	82.3	82.2	71.8
Intel cluster	89.0	82.0	78.6	66.4	90.8	86.2	78.8	74.5
AMD cluster Gig	96.9	83.5	88.2	73.2	95.2	88.2	80.4	70.8
AMD cluster SCI	98.0	89.6	104.0	–	96.3	90.3	85.9	–

For the fixed size problem, it can be seen that the IBM SP3 and AMD-SCI machines demonstrate better efficiency than other clusters. The main reason of it is much higher communication-to-computation speed ratio. Also these two machines demonstrate a superlinear speed-up for 8 and more processors due to cache effects. On the IBM SP3, the size of L2-cache is big enough to fully contain most necessary arrays of a subdomain. In this case the computational speed increases sharply, compensating (fully or partially) the parallelization overhead. On the AMD-SCI, its fast 256 KB L2-cache becomes sufficient to hold several 2D-planes of data arrays that are processed simultaneously, with the similar performance effect.

The remaining 3 clusters (Alpha, Intel and AMD-Gigabit) demonstrate very close behaviour, because their communication-to-computation speed ratios are not much different from each other. For example, the higher ratio of

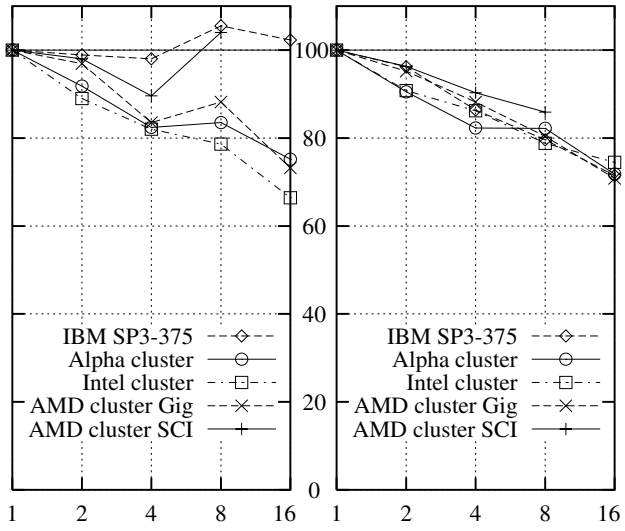

Figure 7. Efficiency (%) for the fixed (left) and scalable (right) problems.

inter-node transfers of the Alpha cluster is compensated by the better intra-node transfers of its competitors. For all three machines, we also see some positive cache effects on 8-processor configurations.

For the scalable problem, the correlation of the parallelization efficiency with the communication-to-computation speed ratio is less clear, partly because the bigger problem is less sensitive to this ratio. Besides, there may be additional individual reasons for every platform. For example, the Athlon and Alpha clusters suffer from differences in processor node's speeds exceeding 2–3 % sometimes. In this case the overall performance is limited by the speed of the slowest processor. The IBM SP3 is supposed to suffer from multi-user environment when user processes may migrate between SMP-nodes during their runs.

Note that the parallelization involves some algorithmic overhead due to the increased computational work in the solution of 3-diagonal systems. Also there is a disbalance between the innermost and outermost subdomains. For this reason, measured performance results must be compared with some monotonically decreasing line representing parallel efficiency of the infinitely fast communications.

Measurements of the AMD-SCI performance will be continued in the future, when 8 SMP nodes with 16 processors become available.

The presented comparison shows that modern inexpensive PC clusters built upon Gigabit Ethernet or, better, upon Dolphin/SCI interface demonstrate very competitive results both in parallelization efficiency and absolute performance. For this reason they can be considered as a promising and economical solution for coming years.

6. Conclusion

We illustrate in this paper that a parallel CFD code can be successfully used as an adequate and sensitive measurement tool for evaluating parallel computers. This code can be easily ported to any platform with standard software tools (Fortran compiler, MPI or another appropriate library, etc). The code was used for choosing the best parallel platform for a cluster at L3M, Marseille (France).

The presented method allows to parallelize 3D CFD codes for simulation of incompressible flows in regular domains. Despite the partially implicit nature of such codes and relatively low communication speed of modern computers' interconnects, this method ensures a reasonable level of parallelization efficiency. The method follows SPMD model and can be easily adapted to different architectures.

The comparative performance analysis of several computers performed with the new code reveals their important characteristics and illustrates the reasonable correlation between communication speed and parallelization efficiency.

7. Acknowledgements

This work was partially supported by the program "Réseau de coopération universitaire et scientifique Franco-Germano-Russe" of the French Ministry of National Education, and by the Russian Foundation for Basic Research (grants RFBR-01-01-00745 and RFBR-02-01-00210).

Concerning the performance evaluation on the AMD biAthlon Cluster at L3M, the authors wish to acknowledge the firm Linagora SA (France), integrator of the L3M cluster, who granted a part of this investigation with their partners AMD, Dolphin and Scali.

The access to other parallel computers was given by CINES, France, and JSCC (Joint SuperComputer Center), Russia.

References

[1] J. Henning, "SPEC CPU2000: Measuring CPU performance in the new Millennium", *Computer*, July 2000, pp. 28–35.

[2] "Numerical simulation of 3-D incompressible unsteady viscous laminar flows: a GAMM workshop" (ed. by M. Deville et al.), *Notes on Numerical Fluid Mechanics*, vol. 36, Vieweg, 1992.

[3] F. Marino and E. Swartzlander, "Parallel implementation of multidimensional transforms without interprocessor communication", *IEEE Transactions on Computers*, vol. 48, no. 9, pp. 951–961, 1999.

[4] L. Borges and P. Daripa, "A fast parallel algorithm for the Poisson equation on a disk", *J. Comput. Phys.*, vol. 169, pp. 151–192, 2001.

[5] O. Bessonov, V. Brailovskaya, V. Polezhaev, and B. Roux, "Parallelization of the solution of 3D Navier-Stokes equations for fluid flow in a cavity with moving covers", *Lecture Notes in Computer Science*, vol. 964, pp. 385–399, 1995.

[6] C. Temperton, "Direct methods for the solution of the discrete Poisson equation: some comparisons", *J. Comput. Phys.*, vol. 31, pp. 1–20, 1979.

[7] C. Temperton, "On the FACR(l) algorithm for the discrete Poisson equation", *J. Comput. Phys.*, vol. 34, pp. 314–329, 1980.

[8] O. Bessonov and B. Roux, "Optimization techniques and performance analysis for different serial and parallel RISC-based computers", *Lecture Notes in Computer Science*, vol. 1277, pp. 168–174, 1997.

[9] V. Polezhaev, O. Bessonov, N. Nikitin, and S. Nikitin, "Convective interaction and instabilities in GaAs Czochralski model", *J. Crystal Growth*, vol. 230, pp. 40–47, 2001.

[10] O. Bessonov and V. Polezhaev, "Mathematical modeling of convection in the DACON sensor under conditions of real space flight", *Cosmic Research*, vol. 39, no. 2, pp. 159–166, 2001.

[11] C. Walshaw and S.J. Farr, "A two-way parallel partition method for solving tridiagonal systems", *School of Computer Studies Research Report Series*, Report 93.25, University of Leeds, U.K., 1993.

[12] F. Cappello, O. Richard, and D. Etiemble, "Performance of the NAS benchmarks on a cluster of SMP PCs using a parallelization of the MPI programs with OpenMP", *Lecture Notes in Computer Science*, vol. 1662, pp. 339–350, 1999.