PARALLELIZATION OF 3D SOLVER FOR INCOMPRESSIBLE FLOWS IN A CYLINDER

Oleg Bessonov
Institute for Problems in Mechanics
of the Russian Academy of Sciences
101, Vernadsky ave., 117526 Moscow
RUSSIA

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

1 Introduction

Modern distributed memory parallel computers are characterized by very high computational potential. Therefore they are very attractive for the solution of time-consuming non-steady 3D CFD problems. 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 RISC 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 (rectangular or cylindrical), the Finite Difference (FDM) and Finite Volume (FVM) methods have proved to be very efficient [1]. 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 after completion of every timestep.

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 (e.g. low Prandtl melt flow in crystal growth applications). The implicit methods should be incorporated, that involve solving 3-diagonal linear systems in every spatial direction (for the economical ADI approach). Another numerical difficulty of incompressible flow simulation arises from the physical nature of 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 [2, 3]. However, these algorithms are either much less accurate than necessary, or less economical than the Fourier method (they would need, for example, $O(N^2)$ operations vs. $O(N\log(N))$ for 1D transform).

The present work is based on the previous effort on parallelization of 3D CFD problem [4] where one-dimensional decomposition of a computational domain was considered. Now, the analysis has been extended to multidimensional decomposition, with the consideration of all arising questions. 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 [5]. As a result, the algorithmically and numerically economical implementation has been obtained for the number of processors up to 16.

The new code has been used as a benchmark for analyzing parallel performance of several distributed memory machines – massive parallel computers (MPP) and multiprocessor node (SMP) clusters. Some previous work has been performed in this area [6]. The 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 Description of the Numerical Method

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 [7].

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 normally 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" [4].
- 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.

3 Parallelization of the Algorithm

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).

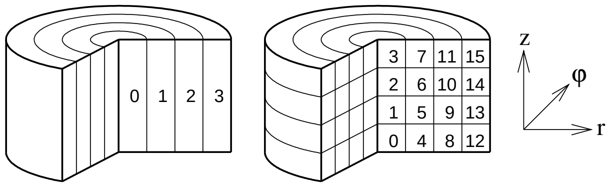

Figure 1: 1D and 2D decompositions of a computational domain

Consider first the parallelization method for 1-dimensional splitting.

- 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 FFTs 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 [8, 4] for 4 or more processors. These methods employ more complicated way of Gauss elimination procedure, that can be done simultaneously in all subdomains. These 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).

As a result, the parallelized numerical method is 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 [4, 6]. 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.

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 transfer is to split a computational domain in the last spatial direction (φ) and rearrange data for this operation. Figure 2 illustrates this rearrangement (blocked transposition) for 4 processors, when $3/4$ of all data are involved into an exchange.

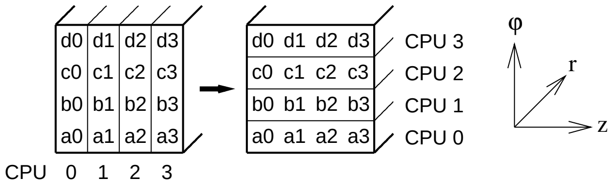

Figure 2: Blocked transposition for parallelization of FFT in the direction z

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)

4 New Parallelized Method for Solving Poisson Equation

The described procedure requires an exchange of 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, the 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 [5] 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 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 by the following way:

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

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 backsubstitution 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 repetition 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.

5 Some Technological Aspects of Parallelization

Parallel computational code expressed in high level language (Fortran in our case) has much more complicated structure than 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. 3). 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.

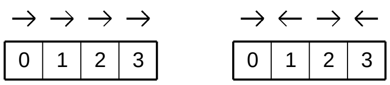

Figure 3: Standard (left) and alternating (right) numbering schemes

Another improvement of the algorithm concerns the solution of 3-diagonal linear systems with constant coefficients, that happens in the discretized pressure Poison equation and some other cases. For this sort of systems, the LU matrix decomposition can be performed in advance, thus reducing the computational work and eliminating data exchanges of matrix elements in parallel solution.

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.

This approach also allows to accomplish some specific optimizations of data exchanges without modification of application code, such as splitting blocks to be transferred into smaller parts, 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 etc) to the structure of an algorithm.

Up to now, the intermediate communication routines have been adapted to the following protocols: NX (Intel i860), Parix (Parsytec), PVM, MPL (IBM SP2), SHMEM (Cray T3E, SGI) and MPI, the latter in different incompatible implementations.

6 Evaluating Parallel Performance of Several Computers

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 internode 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. Main characteristics of all analyzed computers and some results of this investigation are presented in Table 1.

Table 1: Characteristics of 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 SP2-375 Colony switch	16	8M	1500 297	80-175 –	0.27-0.59 –
IBM SP2-120 SP switch	1	128K	480 120	– 28	– 0.23
Alpha 21264 Myrinet	2	4M	1333 347	90 41-73	0.26 0.12-0.20
PC PIII-550 2×Ethernet100	2	512K	550 84.5	39 5.8-10	0.46 0.07-0.12
SGI O2000 shared memory	256	8M	600 125	30-45 –	0.24-0.36 –

The MFLOPS rates were measured by the single-processor version of the presented code, for the problem size 70 MB ($128 \times 64 \times 92$). In order to represent the real life situation for SMP nodes, and to account shared memory conflicts, the appropriate number of copies of this program were running simultaneously.

The communication speed was measured by transferring large arrays of data ($32 - 64$ KB) in duplex mode. Both intra-node and internode exchanges are shown (top and bottom, respectively). When appropriate, measurements were performed in two regimes: heavy, when all CPU pairs exchange simultaneously, and light, when only one pair communicates without conflicts (shown as ranges).

Table 2 and Fig. 4 present the parallelization efficiency results for 2 problems: of the fixed size (70 MB), and the scalable one (70 MB per processor).

For the fixed size problem, a superlinear speed-up can be seen for SP2-375 due to big L2-cache. When the subproblem's size becomes comparable with the size of a cache, most data arrays fit into it entirely, and the computational speed increases sharply, compensating (fully or partially) the parallelization overhead.

Table 2: Parallelization efficiency (%) for two problems

parallel platform	fixed size problem				scalable problem			
	2	4	8	16	2	4	8	16
SP2-375	98.9	98.0	105.5	102.3	96.2	86.4	79.6	71.5
SP2-120	94.1	87.0	82.2	74.7	95.2	91.4	87.9	84.9
Alpha clus.	91.8	82.4	83.5	75.2	90.5	82.3	82.2	71.8
PC cluster	89.0	82.0	78.6	66.4	90.8	86.2	78.8	74.5
O2000	–	–	–	–	94.0	85.3	77.8	57.0

In general, these results illustrate a good correlation between communication-to-computation speed ratio and parallelization efficiency. However, for a big number of processors (16, and sometimes 8) there is some unexpected drop in efficiency for the scalable problem (IBM SP2-375, Alpha cluster, SGI O2000).

For the Alpha cluster, this can be explained by differences in processor node's speed exceeding 5 % in some cases, and also by non-uniform (in time) behaviour of communication network. The IBM SP2-375 is supposed to suffer from multi-user environment when user processes migrate between SMP-nodes during their runs. The most unstable computer happens to be the 256-processor O2000. Due to its NUMA memory organization, even the performance of a single-processor job would vary by 50 % or more in different runs. For this reason, it was impossible to obtain any reliable results for this machine for the fixed size problem.

The classical and well-balanced IBM SP2-120 with single-processor nodes demonstrates very uniform and monotonic behaviour in all regimes.

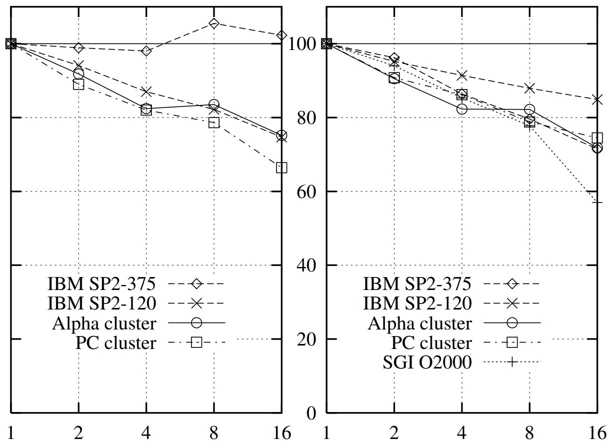

Figure 4: Efficiency (%) for the fixed (left) and scalable (right) problems

The most interesting observations have been obtained for the dual Ethernet PC cluster. Despite its communication network is less balanced than that of the Alpha cluster (Table 1), the PC cluster demonstrates comparable level of parallelization efficiency due to better intra-node exchanges.

With a proper organization of parallel algorithms, this sort of systems may compete to Myrinet-based PC clusters [9] with much more expensive communication hardware. More advances implementations of PC clusters, build upon dual PIII and Athlon platforms (1 GHz or more) and interconnected by gigabit Ethernet, look as promising and economical solutions for coming years.

7 Conclusion

The method presented in the paper 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 correlation between communication speed and parallelization efficiency.

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 (grant RFBR-01-01-00745). The access to parallel computers was given by CINES, France, and JSCC (Joint SuperComputer Center), Russia.

Bibliography

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

[2] F. Marino, E. Swartzlander. Parallel implementation of multidimensional transforms without interprocessor communication. IEEE Transactions on Computers, **48** (9) (1999) 951–961.

[3] L. Borges, P. Daripa. A fast parallel algorithm for the Poisson equation on a disk. J. Comput. Phys. **169** (2001) 151–192.

[4] O. Bessonov, V. Brailovskaya, V. Polezhaev, B. Roux. Parallelization of the solution of 3D Navier-Stokes equations for fluid flow in a cavity with moving covers. Proceedings / PaCT-95, Lecture Notes in Computer Science, **964** (1995) 385–399.

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

[6] O. Bessonov, B. Roux. Optimization techniques and performance analysis for different serial and parallel RISC-based computers. Proceedings / PaCT-97, Lecture Notes in Computer Science, **1277** (1997) 168–174.

[7] V. Polezhaev, O. Bessonov, N. Nikitin, S. Nikitin. Convective interaction and instabilities in GaAs Czochralski model. J. Crystal Growth (2001), to appear.

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

[9] F. Cappello, O. Richard, D. Etiemble. Performance of the NAS benchmarks on a cluster of SMP PCs using a parallelization of the MPI programs with OpenMP. Proceedings / PaCT-99, Lecture Notes in Computer Science, **1662** (1999) 339–350.