Super-linear speed-up of a parallel multigrid Navier–Stokes solver on Flosolver

T. N. Venkatesh[1,*], V. R. Sarasamma[1], Rajalakshmy[1] S., Kirti Chandra Sahu[2] and Rama Govindarajan[2]

[1]Flosolver Unit, National Aerospace Laboratories, PB 1779, Bangalore 560 017, India
[2]Engineering Mechanics Unit, Jawaharlal Nehru Centre for Advanced Scientific Research, Jakkur, Bangalore 560 064, India

In parallel computing, scalability is an important issue and getting linear speed-ups is difficult for most codes. Super-linear speed-up has been achieved on an eight-processor Flosolver system for a multigrid Navier–Stokes code. The physical problem solved, the parallelization method, the speed-ups obtained and possible explanations for this result are discussed here.

PARALLEL computers are used nowadays for all non-trivial Computational Fluid Dynamics (CFD) calculations. The need for large computations makes parallel computers necessary. Also, since a majority of CFD codes use finite difference or finite volume methods, the domain decomposition technique can be used for parallelization, which results in good parallel efficiencies. Other codes that use spectral techniques where global communication is required are known to have moderate to poor scalability. The multigrid technique which is used to accelerate the convergence of Poisson solvers is generally considered to hamper parallel efficiency due to its global dependencies[1,2]. The most commonly used metric to measure scalability is the speed-up, which is defined as the ratio of the time taken by the sequential code to that of the parallel code. Speed-up is a function of the number of processors used. Parallel efficiency is defined as speed-up divided by the number of processors and in general, falls off as the number of processors increases.

At the Flosolver laboratory, we have achieved a super-linear speed-up of 11 on an eight-processor system using the FloSwitch[3] for communication on a multigrid laminar Navier–Stokes code written at the Jawaharlal Nehru Centre for Advanced Scientific Research, Bangalore. Recently, Dresden[4] reported a speed-up of nearly 140 with a 120-processor system on a finite element Navier–Stokes code. Here we document briefly the Flosolver experience.

Description of the problem and computational algorithm

The instability of spatially developing laminar flows, such as those through converging/diverging channels and pipes, is often fundamentally different from that of flows which do not vary downstream. The aim of the code DIVPIPE1S is to obtain flow profiles for spatially developing separated flows, where analytical solutions are not possible. The emphasis is on obtaining accurately the regions of separation, and the flow within and beyond such regions. The Navier–Stokes equations in the vorticity and streamfunction formulation have been solved using the full-multigrid algorithm (FMG) by the finite difference technique. It is found that the convergence in the FMG is about a hundred times faster than with a single grid, when using 128×128 grid points.

The geometry (shown in Figure 1) consists of a pipe which is parallel at the entry, followed by a divergent section, which in turn is followed by a much longer straight exit section. In the present case, the divergent section starts at $x = 9.375$ and ends at $x = 16$ with a 5° angle of divergence. The length of the pipe $L = 120$. All lengths are scaled by the radius R at the inlet. A long parallel exit section is necessary, because we prescribe the Neumann boundary condition ($\partial/\partial x = 0$) at the exit, which is found to be valid only far downstream. For the present case, at a Reynolds number of $Re = 400$ based on the inlet velocity and length scale, the Neumann condition is found to be valid only beyond $x \sim 98$. For this reason, even a laminar flow computation is extremely time-consuming, and a full multigrid algorithm is necessary. A sample steady-state result of an axisymmetric separated flow with reattachment for a divergent pipe is presented here.

The governing equations are

$$\frac{\partial w}{\partial t} + (u.\nabla)w = \frac{1}{Re}\nabla^2 w, \tag{1}$$

$$w = -\nabla^2 y. \tag{2}$$

These are non-dimensional equations with $Re = UR/n$, where U is the centreline velocity at the inlet, n is the kinematic viscosity, y is the streamfunction, w is the vorticity vector, and u is the velocity vector. The solution is facilitated by a transformation of coordinates given by

$$z = x,\ h = \frac{y}{f(x)},$$

where $f(x)$ is a function describing the boundary.

*For correspondence. (e-mail: tnv@flosolver.nal.res.in)

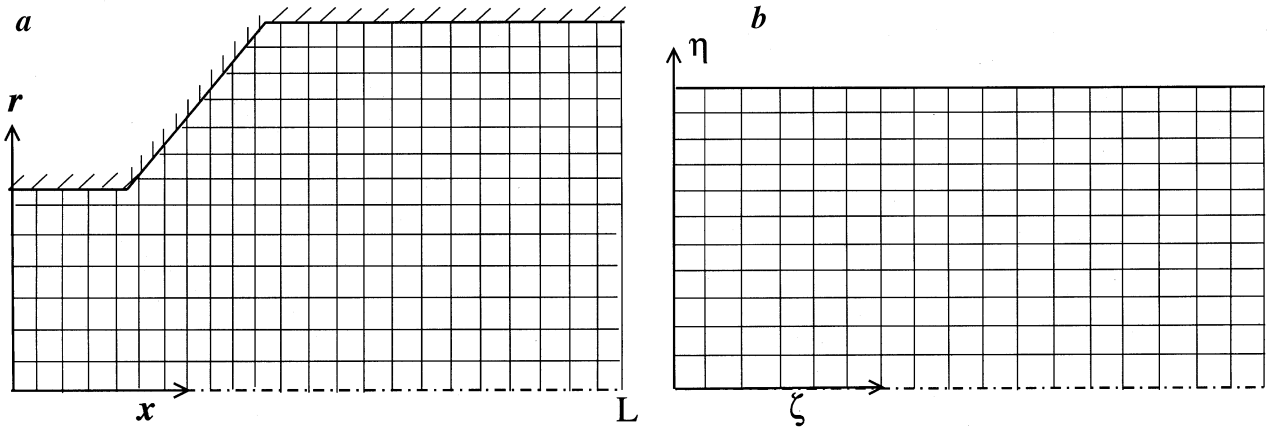

Figure 1. Schematic diagram of axisymmetric divergent pipe. (***a***) Physical domain and (***b***) computational domain.

Solution method

The boundary conditions at the centreline are $y = 0$; $w = 0$; $v = 0$ and u is a maximum; the last condition implying that y is linear at the centreline. No-slip and impermeable boundary conditions have been used at the wall. The functional forms of the streamfunction at the centreline and the vorticity at the wall are prescribed by employing fictitious points outside the domain. At the inlet, a parabolic velocity profile is prescribed, while at the outlet the Neumann boundary condition $(\partial y/\partial z = 0)$[5,6] is used, as mentioned earlier.

After setting up the initial and boundary conditions for the flow parameters, the vorticity at the new time step is calculated from eq. (1). Any standard time marching method may be used for this purpose. Then, using the vorticity at the new time step, stream-functions are computed from the Poisson (eq. (2)). This is the most time-consuming part in the program. If this kind of spatially developing flow is to be solved within realistic time-frames (a few days for each case), a convergence acceleration technique is essential. The multigrid technique (see later in the article) increases convergence rates by a large factor[5], and has been used here. In our case we find the speed-up to be a factor of hundred. Finally, the velocity components are calculated from the computed streamfunction. For a steady-state problem, these steps are repeated till the vorticity residual ($w_{\text{rsd}} \equiv \sum_{i=1}^{n}\sum_{j=1}^{n}|w_{i,j}^{r} - w_{i,j}^{r-1}|$) reduces to a value below a prescribed limit (= 10^{-10} in our case).

The multigrid method[7] provides algorithms which can be used to accelerate the rate of convergence of iterative methods, such as Jacobi or Gauss–Seidel, for solving elliptic partial differential equations. Iterative methods start with an approximate guess for the solution to the differential equation. The difference between the approximate solution and the exact solution is made smaller at every iteration. Algorithms like Jacobi or Gauss–Seidel are local because the new value for the solution at any lattice site depends only on the value of the previous iterate at neighbouring points. Such local algorithms are generally only efficient in reducing short-wavelength error components. In general, the error will be made up of components of many different wavelengths. The basic idea behind multigrid methods is to reduce long-wavelength error components by updating blocks of grid points. A six-level multigrid technique with simple V-cycle algorithm was used in the present case.

Sample result

The variation of streamwise velocity profiles for the pipe at different locations in the downstream is given in Figure 2 *a*. It can be seen that the flow separates at $x = 15.9$ and reattaches at $x \sim 18.3$, giving x independent parabolic profiles after $x = 98.4$. The corresponding normal velocity profiles are shown in Figure 2 *b*. These calculations were performed sequentially on a PC with Pentium IV processor at 2 GHz. The total runtime for convergence up to 1×10^{-10} was around 26 h.

Eight-processor Flosolver system

The eight-processor Flosolver system (Figure 3) consists of four Intel STL2 boards linked by the NAL FloSwitch. Each board has two Intel Pentium III processors with a clock speed of 1 GHz and 2 GB of shared RAM. The operating system on the nodes is Linux. The communication libraries include CCX[8], a small but efficient message-passing library and an in-house implementation of a subset of the MPI library.

Parallelization

The first step involved time-profiling of the sequential code. The time break-up is shown in Table 1. As can be seen from Table 1, a major portion of the computation time is taken by the Poisson solver. Break-up of computational time among the various subroutines in the Poisson solver is shown in Table 2. Here, the relaxation routines take around 75% of the time.

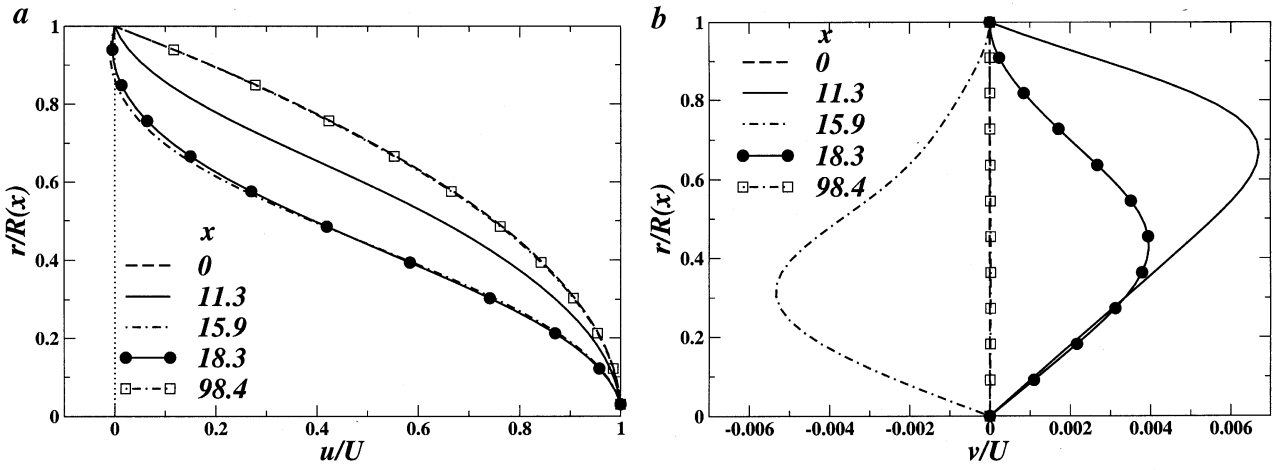

Figure 2. Flow in an axisymmetric divergent pipe, Re = 400 and angle of divergence = 5°. ***a***, Streamwise velocity profiles. ***b***, Normal velocity profiles at different downstream locations.

Table 1. Break-up of computational time among various subroutines of the code

Subroutine	Percentage time
Vorticity (w) update	0.42
Poisson solver	99.35
Velocity calculation	0.06
Iteration loop	99.83
Miscellaneous	0.17
Total	100.00

Table 2. Break-up of computational time of various subroutines in the Poisson solver

Subroutine	Percentage time
Relax (downleg)	38.37
Relax (upleg)	36.93
Residue calculation	10.37
Restriction	11.23
Prolongation	2.35
Miscellaneous	0.75
Poisson solver	100.00

Table 3. Speed-up obtained on Flosolver for DIVPIPE1P

Number of PEs	Boards	PEs per board	Speed-up
1	1	1	1.00
2	2	1	2.12
2	1	2	1.61
4	4	1	4.24
4	2	2	3.99
8	4	2	11.05

For parallelization, the domain decomposition technique was used. In the first stage, the subroutines of the Poisson solver were parallelized. Here the communication of streamfunction values at the boundaries of the subdomains is required. In the next stage, the velocity calculation and the vorticity update calculations were also parallelized. Even though these routines constitute a small fraction of the computation, their parallelization is important, since the communication of the streamfunction values is reduced and also the overall scalability improves.

There are issues to be addressed while parallelizing multigrid computations. Initially, we followed the full multigrid approximation, i.e. boundary information was communicated among the processors at all grid levels, so that the computation at each iteration step was identical to that of a sequential run. This is, however, not essential as there is a convergence criterion enforced at each level. In the next approach, the coarser grid calculations were done without passing the boundary data and communication was restricted to the finest grid. This has much lower message passing overhead, but could take a higher number of iterations to converge, as experienced by Alonso *et al.*[1]. In the present computations, there was no increase and in some cases (for eight processors) the number of iterations decreased. Consequently, the overall speed-up was much better than in the first approach.

Results

The parallel code DIVPIPE1P was run on up to eight processors on the Flosolver Mk 6 system. The speed-up obtained for the various cases is given in Table 3. Since the STL2 board has two processors, there is some flexibility in the number of actual processors chosen for a parallel run. For example, for a four-processor run, one could choose (i) one processor from each of the four boards or (ii) two processors each from two boards. There are differences in the run times and consequently the speed-up between the two combinations. For case (i), each individual process has the complete resources of the board and the communication is through the

FloSwitch. For case (ii), the communication between processors on the same board is fast as shared memory on the same board is used, but there are other overheads since common resources are used.

Figure 3. Eight-processor Flosolver system.

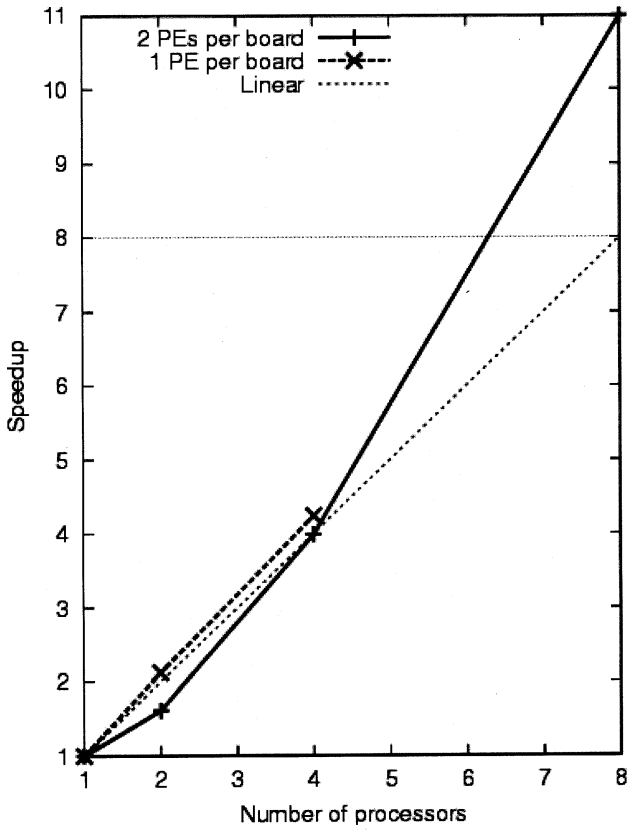

Figure 4. Speed-up of multigrid code on Flosolver.

The graph of speed-up versus the number of processors (Figure 4) clearly shows that there is significant superlinear speed-up beyond four processors. On eight processors, we get a speed-up of 11, i.e. a superlinear speed-up of 37.5%. For comparison with superlinear speed-ups reported on similar problems in the literature, Stiller *et al.*[4] obtained a speed-up of 140 on 120 processors (15% superlinearity). It should be noted though that the problem size also determines when the maximum speed-up is obtained. A general trend is that the number of processors at which the maximum speed-up is obtained increases with the problem size.

Possible explanation of superlinear speed-up

There are many possible reasons for the superlinear speed-up achieved. Both the hardware architecture and the parallel algorithm are likely to play a role. A preliminary study suggests that due to the manner of parallelization, the number of iterations needed for convergence in the sub-domains decreases on increasing the number of processors. The average number of iterations for the different grids is shown in Table 4. The average was taken over the number of processors and the total number of time steps. One can see that the average number of iterations comes down on increasing the number of processors which would result in a decrease in the total amount of work done when more processors are used, and this contributes to the enhanced speed-up. The gain from this is around 1.297 for eight processors. It is generally argued that in principle, appropriate changes can be made to the sequential algorithm to realize a similar gain and this should be the base for calculating speed-up. However, it should be noted that the manner of parallelization is important in this case. In practice, making the necessary changes in the sequential algorithm is not easy or even obvious for complex CFD codes.

The hardware contribution to the superlinearity is likely to be because of both the efficiency of communication and cache effects. The cache effects come into play when the size of the sub-domain becomes small and the variables accessed frequently fit into the cache. The speed-up on eight processors which can be attributed to cache effects is around 8.481

Table 4. Average number of iterations for convergence of the Poisson solver for different grids as a function of the number of processors. The average is taken over all time steps and all processors. Fractional values are obtained for some parallel runs because the number of iterations differs on different processors

Grid	Sequential	2 PEs	4 PEs	8 PEs
1	7	7.000	7.000	6.000
2	5	4.500	2.750	1.876
3	4	3.500	2.250	1.625
4	3	2.500	1.750	1.375
5	3	3.000	1.500	1.125

(11/1.297). This is also greater than the number of processors. A more detailed analysis into the role of various effects is required. Investigations are currently being done and the results will be reported later.

Conclusion

Scalability is an important issue in parallel computing and for most problems, enormous effort goes into making a code come close to achieving linear speed-ups; a superlinear speed-up is extremely rare. Improvements in the machine architecture and use of better parallel algorithms have contributed to superlinear speed-ups on the multigrid problem, which was previously considered difficult to parallelize efficiently. The amount of parallel superlinear speed-up achieved (37.5%) is also, we believe, higher than what has been achieved elsewhere for this class of problems.

1. Alonso, J. J., Mitty, T. J., Martinelli, L. and Jameson, A., *Parallel Computational Fluid Dynamics: New Algorithms and Applications* (eds Satofuka, N., Periaux, J. and Ecer, A.), Proceedings of Parallel CFD '94, Kyoto, May 1994, Elsevier, 1995.
2. Tai, C. H. and Zhao, Y., *J. Comput. Phys.*, 2003, **192**, 277–311.
3. Venkatesh, T. N., Nanjundiah, R. S. and Sinha, U. N., *Developments in Terracomputing*, Proceedings of the Ninth ECMWF Workshop on the Use of High Performance Computing in Meteorology, Reading, England November 2000, World Scientific, 2001.
4. Stiller, J., Frana, K., Grundmann, R., Fladrich, U. and Nagel, W. E., SFB-Preprint SFB609-05-2004.
5. Fletcher, C. A. J., *Computational Techniques for Fluid Dynamics*, Springer, Heidelberg, 1991, vol. 1, 2nd edn.
6. Fletcher, C. A. J., *Computational Techniques for Fluid Dynamics*, Springer, Heidelberg, 1991, vol. II, 2nd edn.
7. Sahu, K. C., Numerical computations of spatially developing flow by full multigrid technique, MS thesis, JNCASR, Bangalore, 2003.
8. Sinha, U. N., Deshpande, M. D. and Sarasamma, V. R., *Supercomputer*, 1989, **6**, 4.

ACKNOWLEDGEMENTS. This work was undertaken with funding from the Defence R&D Organization, Government of India. We thank Dr V. Siddartha for encouragement and Dr U. N. Sinha, NAL, Bangalore for support. Prof. R. Narasimha's comments during the course of the project proved helpful. Thanks are due to Prof. S. P. Vanka for advice on multigrid techniques.

Received 9 July 2004; revised accepted 26 November 2004