RESEARCH ARTICLE

A fluid-dynamics study on ANURAG's parallel computer PACE-8

M. S. Ganagi, K. P. Singh*, G. Athithan and M. V. Atre

ANURAG (Advanced Numerical Research and Analysis Group), P. O. Kanchanbagh, Hyderabad 500 258, India
*Aeronautical Development Agency, P. B. No. 1718, Vimanapura Post, Bangalore 560 017, India

Fluid dynamics is of great importance in science and engineering. The entire dynamics, in all its complexity, is described by the Navier–Stokes equation, which is a set of coupled, nonlinear partial differential equations. They are extremely difficult to integrate, both analytically and numerically. Digital computing machines have been instrumental in numerically solving difficult problems of realistic flows around bodies of practical interest. A three-dimensional time-marching Euler code, which is a simplification of the Navier–Stokes equation, for flow around the forebody of an aircraft fuselage has been carried out on ANURAG's parallel computer PACE-8. We describe the parallelization of the problem, report results for varying numbers of grid points as well as different domain decompositions, and analyse and compare the theoretical and experimental speed-ups.

In recent years, numerical analysis of fluid flow around an aircraft has become a very important subject of study. It involves suitable integration schemes for partial differential equations, specialized time/space marching codes for transonic/supersonic regimes, grid generation, etc., and is called computational fluid dynamics (CFD). A typical CFD problem for an aircraft would involve about one million grid points with 25–40 variables at each point. Since about 1000 iterations are required for stable solutions, solving the problem in 2–3 hours requires a computing rate of over a hundred million floating-point operations per second (MFLOPS). Use of CFD in aircraft design is rapidly gaining importance because it is both faster and more economical than the conventional wind-tunnel studies.

For many years, advanced problems in CFD could only be studied using supercomputers in the Cray class. The advent of parallel computers offers an alternative and a more cost-effective approach. In fact CFD is the driving force behind many projects on parallel computers[1–3]. The parallel-computer project PACE in progress at ANURAG (which also has a similar origin) has been described in an earlier publication[4]. Here we describe a study of airflow around a standard fuselage, carried out on our eight-node parallel machine PACE-8 by numerically solving the appropriate Euler equations. We first describe the sequential algorithm based on the three-dimensional Euler equation. This is followed by discussions of the parallelization, domain decomposition, and the theoretical speed-up. The times taken for solving the Euler equation for different numbers of grid points and different domain decompositions are also presented.

Sequential algorithm

The Euler equation is a simplification of the Navier–Stokes equation where viscous effects are ignored. We have studied the flow around the forebody of an aircraft fuselage (Figure 1), which is symmetric about the vertical plane. This particular shape is a standard one prescribed by the US National Aeronautics and Space Administration (NASA)[5], for which experimental results have been obtained by wind-tunnel tests.

The boundary conditions are set up as follows (see Figure 2):

(i) The Euler equation is initially solved for a small region around the tip (ABB′) of the forebody assuming the tip to be a cylindrical cone. This cone solution along BD (B′D′) and free-stream values along DEFB

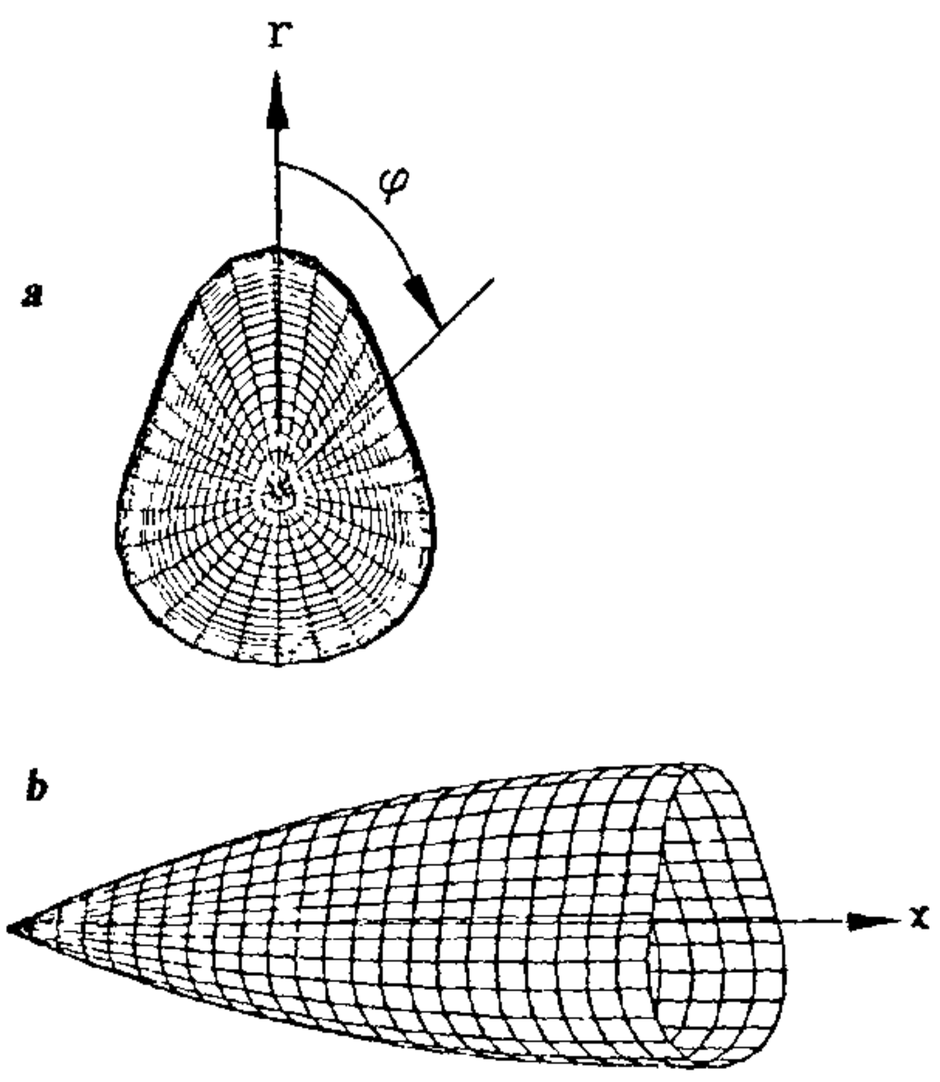


Figure 1. Forebody of the aircraft fuselage obtained from NASA Technical Memorandum 80062 (ref 5). (**a**), Front view; (**b**), side view.

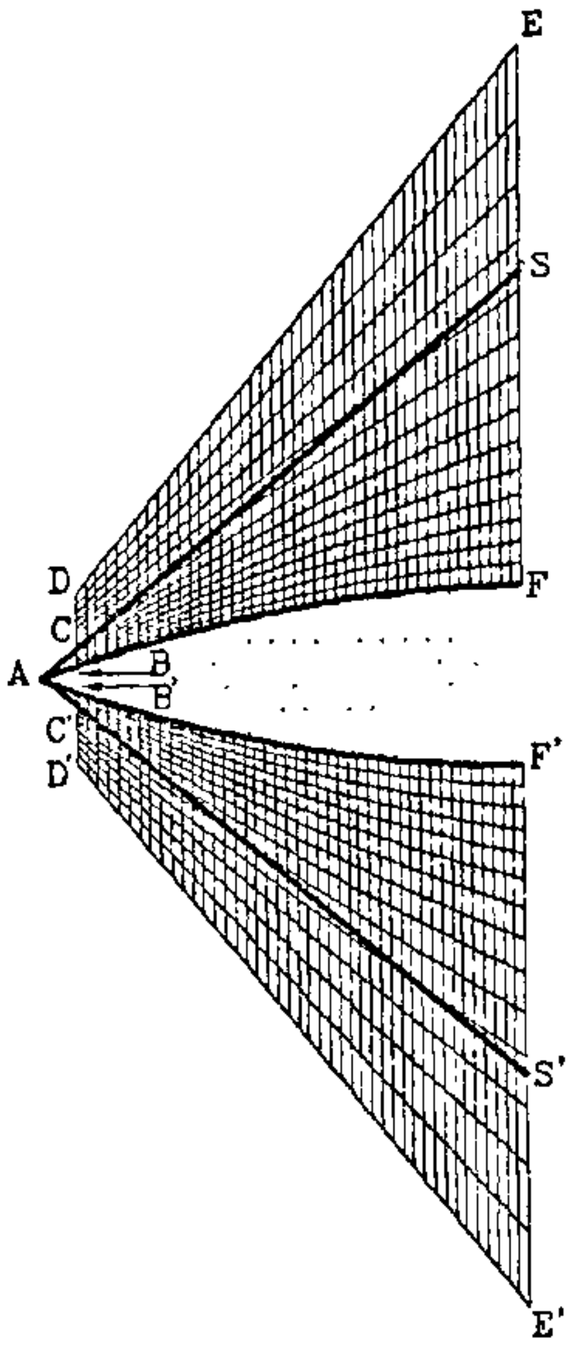


Figure 2. The domain of decomposition BCDEF (B'C'D'E'F') above (below) the forebody. The section is in the r, x plane intersecting the body at $\psi = 0°$ and $\psi = 180°$. ACS, AC'S' are the shock fronts.

(D'E'F'B') are now taken as boundary conditions for solving the Euler equation in the domain BCDEF (B'C'D'E'F').

(ii) Free-stream conditions are assigned initially at all other points within the domain of computation.

(iii) For subsequent iterations, free-stream conditions are imposed on CD (C'D') and DE (D'E'), and BC (B'C') is maintained at the initial boundary condition.

The iteration scheme used is the MacCormack's predictor–corrector method. The predictor step uses forward difference while the corrector step uses backward difference. For the outflow boundary (EF and E'F'), iterations are carried out using the values at two adjacent grid points inside the domain of integration. The momentum principle is used to carry out iterations on the surface of the body. Schuman filtering is used to damp oscillations in the field variables across the shock region after every iteration.

Parallelization of Euler code on PACE-8

To integrate a partial differential equation defined over some spatial domain D, the most natural scheme of parallelization is to decompose the domain D into as many regions as the number of processors such that each processor solves the partial differential equation for a particular subdomain. (This is in the spirit of SIMD machines.) The values of the field variables at the grid points on the boundaries between adjacent subdomains are communicated between adjacent processors for carrying out the iterations at points on the boundaries. In the present case, the domain decomposition is done along x and ψ directions (see Figure 1).

A detailed view of how to program in PACE-8's parallel environment may be found in ref. 6. Basically the entire code is split into two parts, of which the computation-intensive part is in the node program. The other part, namely the front-end-processor (FEP) part of the program, deals with downloading the initial parameters onto the nodes, finding the global minimal incremental time step for the current iteration, and global maximal residue for each iteration. It also collects the final solutions for the subdomains from each of the nodes after a specified number of iterations are carried out or when a specified accuracy is reached.

The parallelized program is generic in the sense that any kind of domain decomposition can be incorporated by setting parameters P_i (number of processors along $i = x, r, \psi$). As an illustration, if the grid size is chosen as $m \cdot n \cdot q$ (along x, r, ψ respectively), then, setting $P_x = a$, $P_r = 1$ and $P_\psi = b$ gives $m/a \cdot n \cdot q/b$ points for each subdomain. (If P is the total number of processors, then it is obvious that $a \cdot b = P$. For PACE-8, $P = 8$).

The FEP program reads the number of divisions along x, r and ψ directions, and finds the subdomain sizes for each node using the topology information a and b and sends them to each of the nodes, along with other necessary input parameters. Appropriate geometry information of each of the subdomains is also sent to the respective nodes. The nodes that have the first plane along the x-direction receive the boundary conditions from the FEP.

Each node assigns free-stream values to all the grid points as an initial solution and appropriate boundary conditions to its boundaries before beginning the iteration cycle. In the iteration cycle, the local incremental time step is found out by all the nodes and sent to the FEP to find the global minimal incremental time step for the current iteration. In each iteration the fluxes on the boundaries between adjacent subdomains are communicated between the nodes for the predictor and the corrector steps. Similarly, to carry out the iterations on the surface of the body and to carry out filtering, the field variables and switches along x and ψ directions are communicated. The squares of differences in pressure at current and previous iterations are determined locally and sent to the FEP by all nodes. The FEP finds the root-mean-square (RMS) residue of the pressure.

For a parallel computing system consisting of P processors the speed-up S is obtained using the generic formula[7]

$$S = \frac{P}{1+\eta}. \tag{1}$$

Here η stands for the parallelizing overhead as a fraction of the total sequential computing time. It is pertinent to mention here that the speed-up S increases as η (defined below) decreases. For a given mesh size (i.e. given m, n, q), the domain decomposition (i.e. a and b) should be chosen appropriately so that η is a minimum. Generally, the overhead η is evaluated using

$$\eta = \frac{\beta N_{\text{comm}} t_{\text{send}} + t_{\text{start}} R + T_{\text{li}}}{N_{\text{comp}} t_{\text{comp}}}, \tag{2}$$

where N_{comm} is the number of words sent in R communications, t_{send} the time for communicating one word by a single node, and t_{start} the start-up overhead involved during each communication. Similarly, N_{comp} is the number of computations carried out in the sequential program and t_{comp} the time for one floating-point operation. β is the degradation factor in t_{send} when all the eight nodes are communicating simultaneously. T_{li} is the total overhead due to load imbalance in node computations. Thus, from equation (2), we see that the parallelizing overhead is the sum of the overheads due to communications as well as load imbalance.

The details of the theoretical speed-up formula for the Euler equation, integrated by the MacCormack's method and parallelized as described above, are given in the appendix.

Solving the Euler equation

The Euler equation has been solved using both sequential and parallel codes. The sequential program was run on the FEP of PACE-8 (equivalent to one node) as well as various other machines accessible to us, and the parallel program was run on PACE-8. The times taken per iteration for different mesh sizes and different domain decompositions on the different machines are presented in Table 1.

The experimental speed-up is given in Table 2 for different grid sizes and topologies. The theoretical speed-up (as obtained from the appendix) is also given for comparison.

Table 1. Time per iteration for sequential and parallel algorithms on various machines.

Grid size	Total no. of points	No. of iterations	Processor decomposition	System	Linpack MFLOPS	CPU time (sec)	Time per iteration (sec)
10 × 10 × 10	1000	100	–			197.00	1.97
40 × 10 × 10	4000	100		Medha—		850.00	8.50
40 × 30 × 19	22,800	5	–	(Cyber 180–930)	0.58	371.60	74.32
96 × 16 × 16	24,576	5	–			439.10	87 82
96 × 16 × 17	26,112	2	–			173.32	86 66
192 × 32 × 17	104,448	2	–			761.3	380 65
10 × 10 × 10	1000	1	–	IBM-3090/150E	5.77	0 6	0 6
40 × 30 × 19	22.800	1	–			14 4	14 4
96 × 16 × 16	24,576	10	–			102.7	10 27
40 × 30 × 19	22.800	5	–	NEC-S-1000	2.8	84 64	16 93
96 × 16 × 16	24,576	5	–			91.55	18.31
96 × 16 × 17	26,112	5	–			100.89	20.18
10 × 10 × 10	1000	50	8 × 1 × 1			45	0.90
			4 × 1 × 2			45	0.90
			2 × 1 × 4			48	0.96
			1 × 1 × 8			67	1.36
81 × 10 × 10	8100	50	8 × 1 × 1			320	6 40
40 × 30 × 17	20,400	50	8 × 1 × 1			902	18.04
			4 × 1 × 2	PACE-8	1.84	900	18.00
			2 × 1 × 4			1120	22.40
			1 × 1 × 8			1270	25 40
96 × 16 × 16	24,576	50	8 × 1 × 1			768	15.36
			4 × 1 × 2			994	19.88
			2 × 1 × 4			1161	23 22
			1 × 1 × 8			1041	20 82
192 × 32 × 17	104,448	5	8 × 1 × 1			494	98 8
10 × 10 × 10	1000	10	–			51.80	5 18
17 × 10 × 16	2720	10	–	FEP of PACE	0.26	140.80	14 08
40 × 10 × 10	4000	10	–			206 00	20 60
81 × 10 × 10	8100	5	–			257.72	51.54
10 × 10 × 10	1000	10	–	Personal IRIS	0.9	13	1 3
40 × 30 × 17	20,400	10	–			306	30 6
96 × 16 × 16	24,576	10	–			364	36.4

Table 2. Speed-ups for different grid sizes and different topologies.

Grid size	Time per iteration on FEP t_1 (sec)	Parallel topology	Time per iteration on PACE-8, t_2 (sec)	Theoretical speed-up from eq. (1)	Experimental speed-up, t_1/t_2
96 × 16 × 16	142.78	8 × 1 × 1	15.36	7.97	9.30
		4 × 1 × 2	19.88	7.96	7.18
		2 × 1 × 4	23.22	7.46	6.15
		1 × 1 × 8	20.82	6.62	6.86

From the steady-state solutions, the pressure coefficient C_P defined by

$$C_P = \frac{P - P_0}{0.7 P_0 M^2} \tag{3}$$

(where P is the pressure, P_0 the free-stream pressure, and M the Mach number), is obtained as a function of x/L (where x is the distance from the first plane and L the distance between the first plane and the last plane along the x-direction). The values of C_P calculated on PACE-8 are plotted in Figure 3. These data were derived from the solutions for a grid size of 96 × 16 × 16, after 1000 iterations.

The steady-state solutions are obtained when the time variations of all the field variables reduce practically to zero (after about 1000 iterations in realistic CFD calculations). In particular, the RMS value of the time derivative of the pressure over all the grid points (also called the residue) decreases and approaches zero with increasing number of iterations. That this indeed happens in our study can be seen from Figure 4.

NASA has presented the experimental data for C_P for the fuselage studied by us[5]. However, no detailed comparisons are made here between experiments and our calculations for that is not the main objective of this paper. But suffice to mention here that the results for C_P obtained on PACE-8 agree with those given in ref. 5. While such a comparison would be of primary importance for a computational fluid dynamicist in terms of validating his code, our focus here is on highlighting the throughput gains achievable by parallelization. The present paper is thus in the same spirit as that of Weissbein *et al.*[8], who earlier reported an evaluation of Intel's iPSC system for CFD studies.

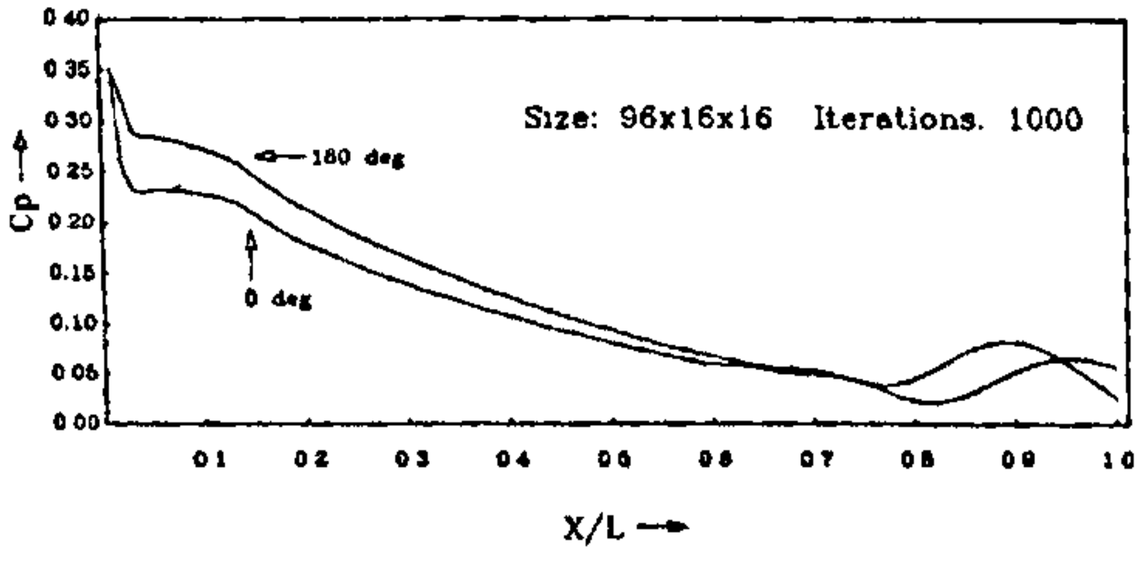


Figure 3. Variation of the pressure coefficient C_P along the fuselage for $\Psi = 0°$ and $\Psi = 180°$.

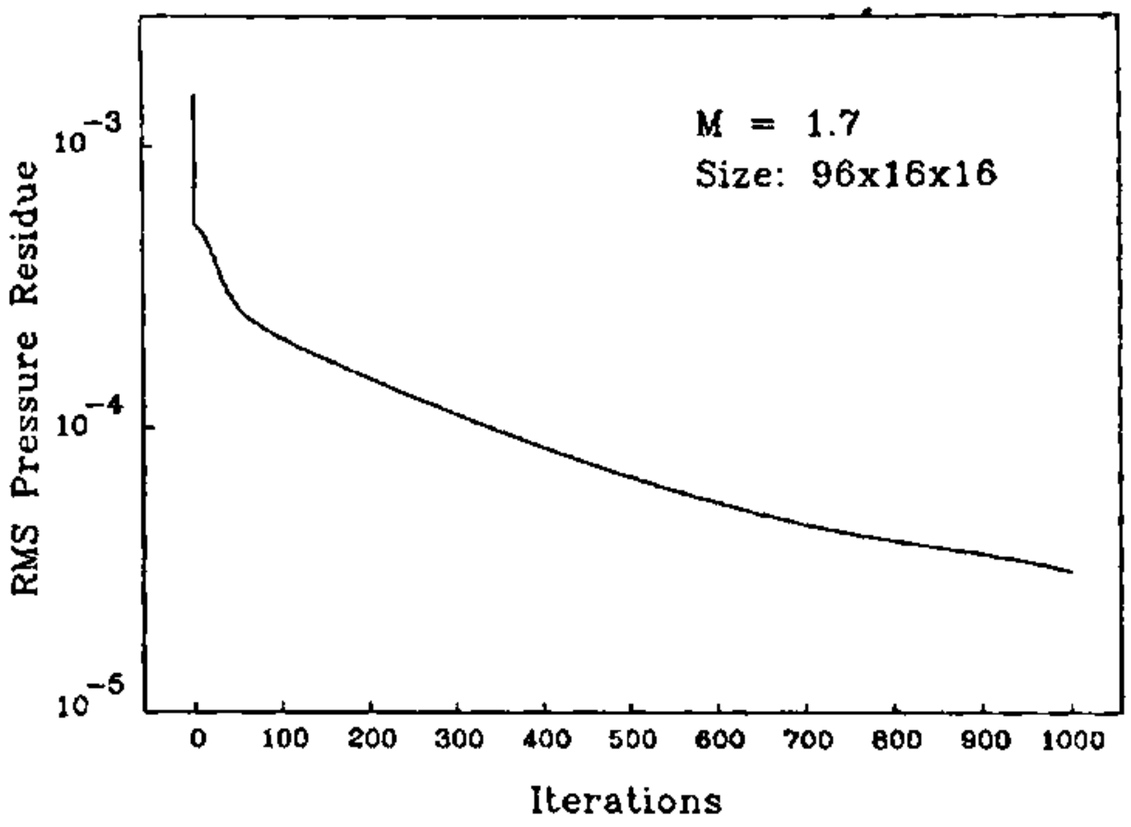


Figure 4. Variation of the RMS value of the time rate of change of C_P with increasing iterations. Note the semi-log plot. If the y-axis is represented on a linear scale, the residue curve would appear flat beyond about 1000 iterations.

Discussion

In the context of the remark just made, we now refer to some of our experiences in running this Euler code on various mainframes accessible to us (see also Table 1).

On Medha (Cyber 180–930) the problem with grid size 192 × 32 × 17 could be executed only in a single-user mode. However, in this case, all processes dealing with system administration could not run. Even in the single-user mode, the execution entailed a large amount of disk swapping, thus greatly increasing the elapsed time per iteration.

On NEC-S-1000, the problem with grid size 192 × 32 × 17 was run in a single-user environment. Even after an elapsed time of about 22 h, the machine could not carry out a single iteration to completion. Here again, owing to inadequate physical memory, heavy swapping between physical memory and secondary storage was involved. In fact more than 99% of the elapsed time of 22 h had been spent in input/output (I/O) operations. Though the situation was comparable to that on Medha, probably better disk-I/O speed of Medha made it possible to complete execution in a reasonable time limit.

On IBM-3090/150E the problem with grid size 192 × 32 × 17 could be run without any perceptible I/O

overheads. This is possible since the size of the physical memory available in the system is much higher than that required for the above problem. In fact the total physical memory is 64 megabytes (MB), though it should be mentioned that the FORTRAN compiler limits the data area to 16 MB. For more memory, the programmer would have to resort to 'dynamic-memory-allocation' technique. Even then, the virtual-storage management may not be required if the problem at hand does not need more than 60 MB of data space.

Turning next to our own machine PACE-8, with 3.5 MB of primary memory available to the user at each node, it is possible to tackle large problems without taking recourse to secondary memory devices. This also minimizes delay due to continual swapping of data. Thus a problem with grid size $192 \times 32 \times 17$ required only 98.8 sec per iteration compared with 380.65 sec per iterations on Medha (Cyber 180–930). The problem with grid size $96 \times 16 \times 16$ took about 15.36 sec on PACE-8 while a similar problem of the same size took around 21 sec on Intel's iPSC/2 SX with eight nodes[9]. However, the algorithm used to solve the Euler equation in the case of iPSC is based on finite-volume method whereas ours is based on finite-grid method.

It is worth mentioning that the delay due to swapping involved while accessing the secondary memory devices was also encountered when running the sequential program on the FEP. Thus the operation on the problem with grid size $96 \times 16 \times 16$ did not complete a single iteration even after 24 h.

Our experience highlights the fact that merely having serial machines with large computation speeds is not sufficient. It is also necessary to have large primary memory available to the user so that the effective time taken for solving any problem is minimized.

On PACE-8, the domain decomposition for a given grid size can be carried out in many ways and these yield different times per iteration (see Table 1). This can be explained as follows: A smaller time per iteration on PACE-8 corresponds to a larger speed-up S, which in turn implies a smaller value of η (eq. 1). Now the ratio of communication between subdomains to computation is related to the ratio of surface area to volume of a given subdomain. Hence we conclude that for a given grid size, a suitable decomposition with the least surface-to-volume ratio will have the least communication overhead. Thus it is desirable to have a subdomain that is as close to a cube as possible. In addition, it is necessary to have identical computational loads on the nodes so that the overhead due to load imbalance is a minimum (see eq. 2).

Using formula (1), we have calculated the theoretical speed-ups for various topologies, for a problem of grid size $96 \times 16 \times 16$. Since this problem could not be run on the FEP owing to excessive swapping, we estimated the time required for sequential run using the time-complexity function derived in the appendix. Independently, the time taken for solving the problem in a parallel manner was experimentally observed for various topologies and thus the experimental speed-up was also determined. The results are summarized in Table 2. It is clear that the matching between the theoretical and experimental speed-ups is not as good as one would desire. To explore the reasons for the discrepancies, we conducted several experiments, particularly to determine differences in computing times for problems of the same computational-time complexity but different data-type declarations. We found that the actual time taken depends very much on the array-size declarations. For instance, to solve a problem of size $96 \times 16 \times 2$ the sequential computation time was found to vary as much as 50 to 60% as one changes the array-size declaration from the minimum required $96 \times 16 \times 4$ to $98 \times 16 \times 6$. The specific mechanism in the overall system that makes the computing time depend on the array-size declarations is not yet clear to us and we are continuing our investigations. Tentatively we suspect that the lack of full agreement between the theoretical and experimental speed-ups is due to the excessive dependence of computing times on array-size declarations. Once we have a better understanding of this matter, we hope to take it into account in the theoretical speed-up formula to achieve a better correlation between the experimental and theoretical data.

To increase the computational speed beyond the levels currently achieved, ANURAG is designing a floating-point accelerator ANUCO, which is a memory-mapped peripheral and is expected to increase the Linpack rating of each node from 0.26 to 1.8 MFLOPS (from simulation studies).

As already reported[4], ANURAG is also engaged in assembling a 128-node computer PACE-128. Along with ANUCO, and the large memory available at each node, it would then be possible to tackle, in reasonable time on PACE-128, more difficult problems in CFD with larger numbers of grid points, on par with what is usually done on supercomputers.

Appendix

An evaluation of the theoretical speed-up requires the time complexity, i.e. the number of computations carried out as a function of the size of the problem being solved, for both the sequential and the corresponding parallel program. Time complexities are generally obtained for a representative type of operation (say floating-point multiplication or addition), though, in practice, one carries out a number of different types of operations. For instance, in our program to solve the Euler equation, one computes exponential and hyperbolic functions in a few places. However, floating-point multiplication has been taken as the standard operation here, as it was found to be the most frequently carried out operation in solving the Euler equation.

Table A1. Equivalent number of standard operations for various functions.

Function	Equivalent number of standard operations
sqrt	1.22
sinh	8.5
cosh	7.8
tanh	8.24
real exp	55.7

One floating-point addition is counted as equivalent to a floating-point multiplication as both these operations were found to take the same amount of processor time on PACE-8. To find out the equivalent number of standard operations for non-standard functions such as hyperbolic sine, cosine, etc., experiments were carried out, and the results are given in Table A1.

With a view to working out the time complexities of the sequential program and the parallel program (node part), a list of the different steps in these programs was made. Since the functional nature of the complexity for these steps in both the sequential and parallel programs turns out to be the same, the results are summarized together in Table A2. Taking $m \cdot n \cdot q$ as denoting the grid size of the problem and $a \cdot 1 \cdot b$ as the processor topology, the variables M, N and Q in the table take the values $m-1$, n and q respectively for the sequential program, but $(m-1)/a$, n and q/b in the case of the parallel program. If $(m-1)/a$ and q/b turn out to be non-integral, then some nodes compute on a subdomain of size $\lceil (m-1)/a \rceil \cdot n \cdot \lceil q/b \rceil$ while others compute on a subdomain of size $\lfloor (m-1)/a \rfloor \cdot n \cdot \lfloor q/b \rfloor$, so that the sums of the number of planes in each node along the three different directions work out to be m, n and q respectively. Using Tables A1 and A2 we find the time complexity of the sequential program is given by

$$N_{\text{comp}} = 731\, MNQ - 697\, MQ + 356\, \gamma MN + 178\, \gamma M,$$

where $\gamma = 2$. The time complexity of the parallel program for the interior nodes is obtained by using the value $\gamma = 0$ and that for the boundary nodes using $\gamma = 1$. Since the boundary nodes along the ψ-direction carry out extra computations, clearly there will be an imbalance of computational load among the nodes if b in the processor topology $a \cdot 1 \cdot b$ has a value other than one or two. This fact has a direct bearing on the speed-up formula.

Table A3 gives the details of the number of words communicated among the nodes for various processor topologies. The parallelizing overhead due to communication, T_{comm}, is evaluated as

$$T_{\text{comm}} = R\, t_{\text{start}} + \beta N_{\text{comm}}\, t_{\text{send}},$$

where R is the number of communications made by all the nodes and N_{comm} the total number of words communicated in R communications. Here t_{start} and t_{send} denote the times required to initiate a communication and send one word (4 bytes) to a randomly selected destination node respectively. For PACE-8, these times have been found to be 176 μsec and 5.6 μsec when only one node is communicating to another node. The factor β takes into account the degradation in communication speed when the communication channel, i.e. the bus, is being used by more than one node. From experiments on PACE-8 we find that β is approximately equal to 1.1 when all the eight nodes are trying to communicate simultaneously.

As mentioned earlier, if the number of divisions along the ψ direction is more than 2, then the boundary nodes perform more computations than the interior nodes. The exact amounts of extra computations are obtained by the differences in time complexities of parallel programs for the boundary nodes and that for the interior nodes and is given as

$$N_{\text{li}} = 356\, MN + 178\, M,$$

Table A2. Time complexities for various computational steps.

Computational steps	Time complexity
Find Δt for next iteration	$M\,(N-1)\,Q\,(43 + 2\,\text{sqrt})$
Predictor step	$81\, M\,(N-2)\,Q$
Calculating fluxes for corrector step	$M\,N\,(Q+\gamma^*)\,(136 + \text{sqrt} + 2\,\sinh + 2\,\cosh + \tanh)$
Corrector step	$91\, M\,(N-3)\,Q$
Calculations on body using momentum principle	$M\,Q\,(234 + 7\,\text{sqrt} + \sinh + \cosh + \text{real exp})$
Calculating fluxes on body	$M\,(Q+\gamma)\,(136 + \text{sqrt} + 2\,\sinh + 2\,\cosh + \tanh)$
Filtering step	$21\, M\,(N-2)\,Q + 132\, M\,(N-5)\,Q$
Calculating fluxes for next iteration	$M\,N\,(Q+\gamma)\,(136 + \text{sqrt} + 2\,\sinh + 2\,\cosh + \tanh)$
Finding residue	$3\, M\,(N-2)\,Q + 2\, M\,(N-1)\,Q$

*In the case of sequential, parallel (interior node), parallel (boundary node) programs, γ takes the values 2, 0, 1 respectively. (A node is considered as interior or boundary with reference to the ψ direction.)

Table A3. Number of communications for different topologies.

Topology $(a \cdot 1 \cdot b)$	N_{comm}	R
$8 \times 1 \times 1$	$q\,(a-1)(52n-4) + 2p^*$	$2p + 12(a-1)$
$4 \times 1 \times 2$	$q\,(a-1)(52n-4) + 2p + bm(a-1)(26n-2)/a$	$2p + 12(a-1) + 6(b-1)$
$2 \times 1 \times 4$	$m(b-1)(52n-4) + 2p + qa(b-1)(26n-2)/b$	$2p + 6(a-1) + 12(b-1)$
$1 \times 1 \times 8$	$m(b-1)(52n-4) + 2p$	$2p + 12(b-1)$

*p stands for the number of processors in the parallel computing system.

where the subscript represents load imbalance. If the total number of nodes is p, and q of them are boundary nodes, then $(p-q)\, N_{\text{li}}\, t_{\text{comp}}$ is the processing time lost owing to this load imbalance. Thus the load-imbalance overhead is given by

$$T_{\text{li}} = (p-q)\, N_{\text{li}}\, t_{\text{comp}}$$

The speed-up S for a parallel system of p processors is given as

$$S = \frac{P}{1 + \dfrac{T_{\text{comm}} + T_{\text{li}}}{T_{\text{seq}}}}.$$

Here $T_{\text{seq}} = N_{\text{comp}}\, t_{\text{comp}}$, where N_{comp} is the time complexity of the sequential program. The value of t_{comp} is 8 μsec in the case of a single node in PACE-8. Using appropriate formulas for T_{comm}, T_{seq} and T_{li} in the above speed-up formula, one can evaluate the speed-up for various processor topologies and various sizes of the problem as required.

1. Nosenchuck, D. M., Littman, M. G. and Flannery, W., *J. Sci. Computing*, 1986, **1**, 53.
2. Trottenberg, U., Guest Ed: Proc. of the 2nd International SUPRENUM Colloquium, Parallel Computing, 1988, vol. 7, no. 3.
3. Sinha, U. N., Deshpande, M. D. and Sarasamma, V. R., *Curr. Sci.*, 1988, **57**, 1277.
4. Neelakantan, K., Ghosh, P. P., Ganagi, M. S., Athithan, G., Atre, M. V. and Venkataraman, G., *Curr Sci.*, 1990, **59**, 982.
5. Townsend, J. C., Howell, D. T., Collins, I. K. and Hayes, C., NASA Technical Memorandum, No. 80062, Washington, DC, USA, 1979.

6. Ghosh, P. P., Ganagi, M. S. and Ashok, A., 'Simulator users manual', ANURAG Report ANU/PACE/90/02, Hyderabad, 1990.
7. Fox, G. C., Johnson, M. A., Lyzenga, G. A., Otto, S. W., Salman, J. K. and Walker, D. W., *Solving Problems on Concurrent Processors*, Prentice-Hall International, New York, 1988, vol. I.
8. Weissbein, D. A., Mangus, J. F. and George, M. W., in *Hypercube Concurrent Computers and Applications* (ed. Fox, G. E.), ACM Inc., Pasadena, USA, 1988, vol. II, p. 1127.
9. Intel Scientific Computers, CFD, June 1989 (technical literature distributed by Intel Corporation, Santa Clara, USA).

ACKNOWLEDGEMENTS. We are deeply grateful to Dr G. Venkataraman and Dr Kota Harinarayana for constant encouragement and support. We thank Mr R. Raghunathan (RCI) and Mr V. S. Sarma (NIC) for their assistance in running the Euler code on Medha and NEC-S-1000 respectively; and Dr K. Neelakantan and Mr P. P. Ghosh, among other colleagues, for helping us at various stages of the work reported here.

Received 12 April 1991, accepted 12 June 1991

RESEARCH COMMUNICATIONS

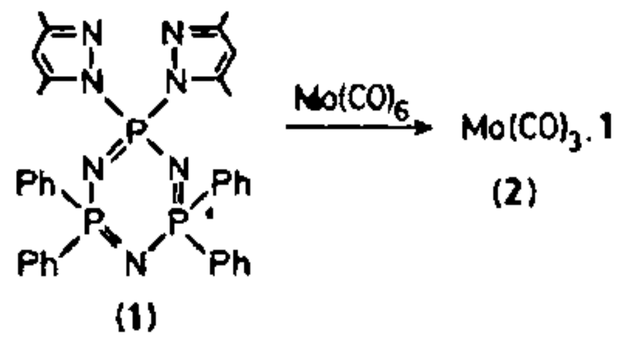
Novel coordination behaviour of *gem*-bis(pyrazolyl)cyclotriphosphazenes as tridentate NNN-donor ligands: The crystal structure of $[Mo(CO)_3\{N_3P_3Ph_4(3,5\text{-}Me_2C_3HN_2)_2\}]$

A. Chandrasekaran, S. S. Krishnamurthy and M. Nethaji

Department of Inorganic and Physical Chemistry, Indian Institute of Science, Bangalore 560 012, India

Reactions of group 6 metal carbonyls with bis(pyrazolyl) phosphazenes yield metal tricarbonyl complexes, $[M(CO)_3{\cdot}L]$ [$L = N_3P_3Ph_4(3,5\text{-}Me_2C_3HN_2)_2$ (1) or $N_3P_3(MeNCH_2CH_2O)_2(3,5\text{-}Me_2C_3HN_2)_2$ (4)]. The structure of the complex $[Mo(CO)_3{\cdot}1]$, determined by single-crystal X-ray analysis, shows that the (pyrazolyl) phosphazene acts as a tridentate ligand; the two pyridinic pyrazolyl nitrogen atoms and a phosphazene ring nitrogen atom are coordinated to the metal. A similar structure is proposed for the complexes $[M(CO)_3{\cdot}4]$ (M = Mo or W] on the basis of their spectroscopic data.

CURRENT interest in cyclophosphazenes is mainly focused on their organometallic chemistry[1,2]. Zerovalent metal complexes of cyclotriphosphazenes are sparse and have not been structurally characterized[1,3]. In this communication, we report the synthesis and structural study of the molybdenum tricarbonyl complex **2** of 2,2,4,4-tetraphenyl-6, 6-bis(3,5-dimethyl-1-pyrazolyl) cyclotriphosphazene (**1**) in which a phosphazene ring nitrogen atom is involved in coordination along with the two pyrazolyl pyridinic nitrogen atoms. We also report the synthesis and spectroscopic studies of a new bis(pyrazolyl) cyclotriphosphazene, viz. 2,2,4,4-bis (*N*-methylethanolamino)-6, 6-bis(3, 5-dimethyl-1-pyrazolyl) cyclotriphosphazene (**4**) and its metal tricarbonyl complexes, $[M(CO)_3{\cdot}4]$ (M = Mo (**5a**) or W (**5b**)). Complexes **2** and **5** are the first examples of structurally well-characterized systems in which a cyclotriphosphazene acts as a tridentate ligand. Pd(II) and Pt(II) complexes of pyrazolylphosphazenes are known but in these complexes there is no involvement of phosphazene ring nitrogen atom in coordination[4].



Compound **2** was prepared by heating[4] the ligand **1** (0.5 g) and molybdenum hexacarbonyl (0.2 g) (molar ratio 1:1) in 40 ml acetonitrile under reflux in an atmosphere of dry N_2 for 6 h. The product was precipitated as yellow crystals, which was washed with acetonitrile and dried under vacuum (yield: 76%). Single crystals were obtained by carrying out the reaction under appropriate dilute conditions and cooling the reaction mixture to 0°C. Compound **2** is not soluble in common organic solvents. Elemental analyses and infrared spectrum showed it to be a metal tricarbonyl derivative, $[Mo(CO)_3 \cdot 1]$. The structure of the complex was determined by single-crystal X-ray diffraction and a view of the molecule is shown in Figure 1. In addition to the pyridyl nitrogen atoms of the pyrazolyl groups, a nitrogen atom of the phosphazene ring is involved in coordination to the metal.

The geometry around the metal atom is distorted octahedral with short M–C bonds on one face and longer M–N bonds on the opposite face. The metallocycle is in a boat form and the heads of the boat (Pl and Mo) are bridged by a phosphazene ring nitrogen atom. The phosphazene ring is distinctly non-planar; the phosphorus atom (P3) and the adjascent nitrogen atom (N2) project upward (by 0.17 Å) from the plane formed by the other phosphazene ring atoms (N3, P1, N1, P2). The phosphorus-nitrogen bond distances are in the range 1.557(2)–1.637(2) Å with a mean value 1.600 Å. The phosphazene ring nitrogen–metal bond is longer (2.394(2) Å) than the pyrazolyl nitrogen–metal