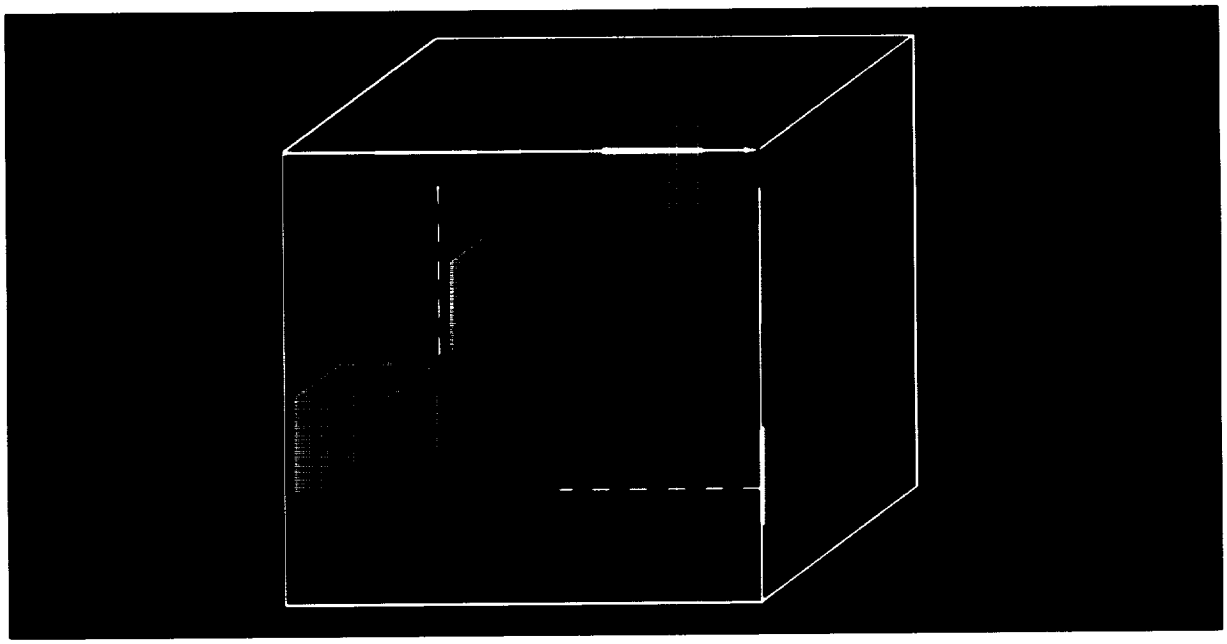
MCAT Institute
Progress Report
94-06

Multi-partitioning for ADI-schemes on message passing architectures

Rob F. Van der Wijngaart



July 1994

NCC2-752

MCAT Institute
3933 Blue Gum Drive
San Jose, CA 95127

Multi-partitioning for ADI-schemes on message-passing architectures

Rob F. Van der Wijngaart

1 Introduction

In order to simulate the effects of the impingement of hot exhaust jets of High Performance Aircraft on landing surfaces a multi-disciplinary computation coupling flow dynamics to heat conduction in the runway needs to be carried out. Within each of the disciplines one or more partial differential equations need to be solved. Straightforward discretization and linearization of these equations leads to large sparse systems of linear equations that cannot be solved efficiently using direct methods. However, if the three-dimensional spatial discretization operators are approximated by products of one-dimensional operators, they can be inverted at a reasonable cost. This kind of discrete-operator splitting is called Alternating Direction Implicit (ADI).

Because of its computational attractiveness, ADI has become a major work horse for solving compressible fluid flow problems (e.g. [1]). Consequently, a lot can be gained from an efficient parallel implementation. But the most popular paradigm for parallel computations—that of an array of processors that have access only to local data and that communicate through so-called messages—suffers from the fact that ADI creates data dependencies in all three coordinate directions in the successive stages of the inversions of the respective one-dimensional operators. These dependencies can result in severe load imbalances, or in heavy message traffic, depending on how the total problem is decomposed into tasks to be solved by the individual processors.

The work described in this report relates to the study of decomposition techniques that minimize load imbalance and message-passing frequency.

2 Decompositions

In the field of finite-element, -volume, and -difference methods for partial differential equations the amount of computational work per grid point is usually constant. Therefore, a uniform distribution of work among processors can be obtained by assigning equal numbers of points to each processor. This is called domain decomposition.

3 Results and conclusions

Of the three most viable techniques examined in [2] the so-called multi-partitioning strategy was found to be most efficient on the Intel iPSC/860 and Paragon for the NAS Scalar Penta-diagonal Parallel Benchmark (SP) [3]. This efficiency derives largely from the coarse granularity of the strategy, which reduces latencies and allows overlap of communication and computation.

Consequently, multi-partitioning was also used for the implementation of the SP and a modification of a full-fledged flow solver [4] on a network of workstations using the Parallel Virtual Machine (PVM) package from Oak Ridge National Laboratory [5]. This implementation has met with only moderate success so far, mostly because of the very limited communication properties of Ethernet connecting the workstations. An upgrade of the switching network is expected to improve performance significantly.

The results of the above investigations are described in references [2] and [4], which are attached to this report as appendices.

References

[1] T.H. Pulliam, D.S. Chaussee, *A diagonal form of an implicit approximate factorization algorithm*, Journal of Computational Physics, Vol. 29, p. 1037, 1975

[2] R.F. Van der Wijngaart, T. Phung, E. Barszcz, *Three implementations of the NAS scalar penta-diagonal benchmark*, submitted for presentation at Supercomputing '94, November 1994

[3] D. Bailey, J. Barton, T. Lasinski, H. Simon, *The NAS parallel benchmarks*, NASA Ames Report RNR-91-002 Revision 2, 1991

[4] M.H. Smith, R.F. Van der Wijngaart, *Granularity and the parallel efficiency of flow solution on distributed computer systems*, 25^{th} AIAA Fluid Dynamics Conference, Colorado Springs, CO, June 20-23, 1994

[5] A. Geist et al., *PVM 3 User's guide and reference manual*, ORNL/TM-12187, Oak Ridge, Tennessee, May 1993

N/S

AIAA-94-2260

Granularity and the Parallel Efficiency of Flow Solution on Distributed Computer Systems

Merritt H. Smith
NASA Ames Research Center
Moffett Field, CA

Rob F. Van der Wijngaart
MCAT Institute
Moffett Field, CA

25th AIAA Fluid Dynamics Conference
June 20-23, 1994 / Colorado Springs, CO

GRANULARITY AND THE PARALLEL EFFICIENCY OF FLOW SOLUTION ON DISTRIBUTED COMPUTER SYSTEMS

Merritt H. Smith[†] and Rob F. Van der Wijngaart[‡]
NASA Ames Research Center
Moffett Field, California

Abstract

Distributed parallel computer systems present a unique challenge to the designer of flow solution algorithms. Message latencies on the order of milliseconds, poor connectivity, and very low network bandwidths stand as obstacles to parallel efficiency. A parallel method, developed in this work, for solving the Reynolds-averaged Navier-Stokes equations achieves reasonable efficiency (>50%) on clusters of from one to eight moderately fast workstation processors connected by Ethernet. A variable-grain multi-partition algorithm provides useful parallelism while limiting the number and volume of messages. We describe a new parameter which, while incomplete, begins to coalesce the performance of different algorithms on different clusters into a single curve.

Introduction

Combining multiple workstations into a network-based parallel computer is an attractive alternative to spending millions of dollars on a conventional supercomputer or a dedicated multi-processor. In many instances the hardware is already in place. A workstation cluster is easily and inexpensively expandable, and the workstations need not be co-opted from their original intended purposes. It becomes a matter of careful programming to produce useful applications on the distributed system. Therein lies the challenge.

It is tempting to think of a cluster of workstations as just another variety of distributed memory multi-processor. This thinking is accurate in the sense that there are multiple processing units, each with its own local memory, and all connected by a communications network. The difficulty with this view is that it suggests that a parallel computational strategy, which runs efficiently on a dedicated multiprocessor, will also do well on a cluster. Typically, this is not true.

Inter-Processor Communications

The connectivity within a dedicated multiprocessor is generally far better than that of a cluster, which might be connected only by Ethernet. While both systems provide processor interconnectivity, the internal connections of a closely coupled machine are specifically designed for parallel computing. The external network traditionally has been used for infrequent file transfers, often over long distances. The requirements for these two tasks are different, and using an external network for connectivity in parallel computing involves some additional costs.

Two primary factors describe the communication performance of a connectivity network. The first is message latency -- the time required to prepare a message for delivery. The second is the data transfer rate, or bandwidth.

Mattson, et. al.[1] have made a comparison of several widely available distributed-computing communication packages. They have compiled a list of round-trip transfer times for a number of different message sizes on a pair of Sun SPARCstation 1 workstations connected by Ethernet. The focus here will be on the data from the Parallel Virtual Machine (PVM)[2] package from Oak Ridge National Laboratory, but the data for most other packages is similar. They report that PVM 2.4.1 has a message latency of approximately 2.5 milliseconds and a bandwidth of 583 KBytes/second. Experience with PVM 2.4.1 and the more

---

[†]Research Scientist, Member AIAA
[‡]Research Scientist, MCAT Institute, Member AIAA

recent release, PVM 3.2.3, on the NAS Distributed Computing Facility at NASA Ames Research Center suggests that latency and bandwidth are similar for later versions of PVM.

When compared to the message-passing speed of a dedicated multiprocessor like the Intel iPSC-860, the communication performance of PVM on Ethernet is quite poor. Bokhari[3] reports a latency of 0.095 milliseconds for the Intel machine, and a transfer rate of approximately 2.5 Mbyte/second. For larger messages the transfer rate is the dominant factor, and the iPSC-860 is over four times faster than PVM on an uncontested Ethernet. For the smallest messages latency dominates, and the iPSC-860 is over twenty-five times faster. Thus, the message latency and bandwidth of PVM communications over Ethernet can conspire to render strategies imported from dedicated multiprocessors ineffective.

Other features of PVM can cause further degradation of communication performance on any network[4]. PVM version 3.2.3 and earlier require explicit packing of message buffers before data may be sent out over the network, and unpacking once the message arrives. Packing and unpacking are basically byte copying operations. On some systems, this operation can be quite slow, on the order of 10MByte/second. If the PVMDEFAULT option is selected for a particular message, a conversion to the XDR device independent data format is required, slowing the process still more.

A formula similar to that for conductances in a wire describes the effect of message buffering speed (B_{pack}) and network bandwidth (B_{net}) on the total communication performance (B_{com}):

$$B_{com} = \frac{1}{1/B_{net} + 2/B_{pack}},$$

Assume that the full Ethernet bandwidth of 1.25MByte/second can be attained between two Silicon Graphics, Inc. R3000 processor based workstations, and that packing and unpacking of XDR encoded data proceeds at 2.2MByte/second. The actual communication bandwidth which may be attained is then 0.59MByte/second. If the Ethernet is replaced with FDDI operating at 12.5MBytes/second, the maximum attainable bandwidth is 1.01MByte/second. Thus, most of the gain from changing to FDDI is lost to the packing and unpacking of XDR encoded data.

One immediate improvement can be gained by communicating only raw binary data (PVMRAW). In this case, packing and unpacking speeds can increase by a factor of four or five. Still, increasing the unpacking speed to 7.1MByte/second holds FDDI to a peak of 2.8Mbytes/second.

Communication should be looked upon as a penalty to parallel computation; a comparable serial algorithm has no need for it. The best route to parallel efficiency on a cluster or any other system is to maximize the real computational work in relation to the losses induced by communication.

At this point we define two types of granularity used in this work. The first, Latency Granularity, is the ratio of the number of floating point operations to the number of messages sent between processors of the cluster ($G_l = N_{flop}/N_{msg}$). The Bandwidth Granularity is the ratio of the number of floating point operations to the total message volume ($G_b = N_{flop}/N_{WT}$). In general, algorithms with larger granularity are better suited to cluster computation.

CFD on Clusters

In deference to poor latency and bandwidth, appropriate solution strategies must be found if flow solutions are to be efficiently computed on clustered systems. One such strategy has already been demonstrated. Multiple grids provide an easily exploited parallel structure which has been used to advantage on workstation clusters[5]. Multiple-grid systems used in Computational Fluid Dynamics (CFD) simplify the description of complex geometries[6,7]. Typically, the flow equations are solved on each grid independently, with information exchanged only on inter-grid interface surfaces. A coarse-grained parallelism (i.e. large G_l and G_b) may be extracted by placing each grid on a different processor, without changing the basic solution process used by a serial computer. The method needs only two messages per grid per iteration, and the message volume per iteration is, at maximum, double the total boundary surface area of the mesh system. Both granularities increase if the boundary surfaces need not be updated after

every iteration. Parallel efficiencies in excess of 95 percent have been achieved using this strategy for a relatively small number of processors (fewer than 12).

One difficulty with this kind of very coarse parallelism is load imbalance, which can cause enormous losses. Imbalance occurs when the workload assigned to each processor is not commensurate with its performance. Processors which finish early will sit idle until the slowest or most heavily loaded processor finishes.

The second, and possibly key disadvantage of the grid-by-grid method is that the degree of parallelism is limited to the number of grids in the problem. Of course, grids can be divided into sub-blocks to provide work for additional processors, or to improve the load balance. For a strictly explicit solver, this is a reasonable choice. For (semi-)implicit methods the data dependency of the solution at each point of the grid extends beyond the boundaries of the local sub-block. If the boundaries of the sub-block are explicitly updated, the convergence rate may be decreased. As total time to solution is the factor which must be minimized, what is needed is a scheme which will maintain the implicitness of solution on an individual grid while efficiently distributing the work over multiple processors.

Numerical Method

As outlined above, a successful parallel algorithm on a distributed sytem must have two key features. With the associated challenge in parentheses, they are:

1. Minimum number of messages (Latency).
2. Minimum message volume (Bandwidth).

Additionally, two objectives are common to all parallel (CFD) applications.

3. Maintain or improve the convergence characteristics of the original serial algorithm.
4. Keep all processors busy doing useful work.

The grid-by-grid parallel algorithm described above and in Ref. 5 meets nearly all of these criteria. Only two messages per grid per iteration are needed. The total amount of message traffic for a given grid system is a minimum, and the original serial algorithm is used in an unmodified form. It is the fourth criterion which this method may fail to meet. If there are more processors available than grids in the problem, the extra processors go idle.

It is the goal of this work to find and implement a parallelization strategy which meets all of these objectives, and test it in a distributed computing environment. The strategy selected is a variation on the three-dimensional multi-partition method described by Naik, et. al.[8] The modifications to the method follow the work of Van der Wijngaart[9], who has used this method to solve the heat equation on an Intel iPSC-860.

A complete single-grid flow solver, called MEDUSA-MP, and a version of the NAS SP Parallel Benchmark which use this multi-partition method are developed and discussed in the next section.

Flow Solution Algorithm

MEDUSA-MP solves the Reynolds-Averaged Navier-Stokes equations, cast in strong conservation law form, in generalized coordinates using the diagonalized Beam-Warming[10] algorithm as implemented in the OVERFLOW[11] code. The original Beam-Warming algorithm may be represented as:

$$[I + h\delta_\xi A^n]\,[I + h\delta_\eta B^n]\,[I + h\delta_\zeta C\,]\,\Delta Q^n = -h\,(\delta_\xi E^n + \delta_\eta F^n + \delta_\zeta G^n)$$

where E, F, and G are the combined viscous and inviscid fluxes, A, B, and C are the combined viscous and inviscid flux Jacobians, and ΔQ^n is an increment of the conserved quantities. Implicit second-order dissipation is added to the left side of the equation, and mixed second and fourth-order dissipation is added to the right-hand-side (RHS) to enhance stability. The system is solved by successively inverting the matrix operators (from left to right) along lines in each of the coordinate directions.

Using the eigenvectors of the inviscid flux Jacobians, the equations may be approximately diagonalized:

$$T_\xi\,[I + h\delta_\xi \Lambda_\xi]\,N\,[I + h\delta_\eta \Lambda_\eta]\,P \cdot [I + h\delta_\zeta \Lambda_\zeta]\,T_\zeta^{-1}\,\Delta Q^n = RHS^n$$

where RHS^n is identical to the right-hand-side of the block Beam-Warming scheme, Λ_x is a diagonal matrix of the eigenvalues, and T_ξ, N, P, and T_ζ^{-1} are block-diagonal. While it is not true that the inviscid and viscous flux Jacobians are simultaneously diagonalizable, an estimate for the viscous contribution to the eigenvalues may be made and added to the appropriate implicit operators.

The steady-state solutions for the block and diagonal schemes are identical, but the diagonalized scheme is non-conservative in time, and the time-accuracy in general is lowered from second to first-order[12]. A complete description of this algorithm may be found in Ref. 10.

The advantage of the diagonalized scheme over the original algorithm is that the matrix systems which must be inverted have gone from 5x5 block-tridiagonal to either scalar-tridiagonal or scalar-pentadiagonal, depending on whether second-order or mixed second and fourth-order implicit dissipation is added. In either case, a dramatic decrease in the number of arithmetic operations results. The change will also reduce the total volume of inter-processor communication in a parallel implementation, as will be shown later.

NAS SP Parallel Benchmark

The NAS Scalar Pentadiagonal (SP) Parallel Benchmark code was ported from the Intel Paragon XP/S-15 to the cluster without changes, except for message passing syntax and process spawning, which accounted for less than one percent of the source code. SP is a simplified version of the diagonalized Beam-Warming scheme with fourth order dissipation discussed above. It is described in detail in Ref. 13.

The main simplifying features of the benchmark versus a true engineering application are:

- Solution takes place on a Cartesian grid, thus obviating the need to compute and store metric terms.
- No boundary conditions are applied; the boundary values are initialized and never changed during the course of the computation.
- No turbulence model is used.
- The added fourth order smoothing is not solution dependent.

Although these simplifications render the benchmark invalid as a flow solver, they do retain the main computational features in terms of data dependencies and communication requirements. In fact, the benchmark is a rather severe test case for any parallel implementation, since the simplifying assumptions reduce the computational load while leaving the communication requirements virtually unchanged.

Parallelization

A single grid is subdivided into a three-dimensional array of $N \times N \times N$ grid blocks, or partitions, which are evenly distributed among N^2 processors; each processor receives N whole blocks. The partitions must be distributed to the processors such that any coordinate plane will pass through exactly one partition held by each processor. This will ensure a balanced load across equivalent systems.

An example partitioning scheme for a system to be solved by 4^2 processors is shown in Fig. 1. Assume (K,L,M)-indexing in the partition array. In the K=1 partition plane the numbering is canonical. As K increases by one the location of the partition assigned to processor 1 is increased by one in both L and M. Thus, processor 1 owns the body diagonal of the partition array, hence the name "3D diagonal multi-partitioning" used by Naik, et. al. Looking at the family of K-partition planes, as K increases, processor numbers which "fall off" on the right or top re-appear on the next K-plane on the opposite edge. More concisely, each plane family is periodic in the two included coordinate directions. Thus, every diagonal of the partition array is held by a single processor. The reasons for this careful partitioning will be explained shortly.

Solution of the system proceeds as follows. The right-hand-side vector is computed in all N partitions assigned to each processor. Inversion of the matrix operators proceeds in two steps using a wavefront technique. If ξ runs parallel to K in Fig. 1, each processor advances the forward sweep of the Thomas algorithm for all ξ-line segments in the K=1 partition plane. Upon completion of the forward sweep for K=1, a message is created on each processor containing essential elements of the matrix super-diagonals and the modified right-hand-side vector from all ξ-line segments in the partition. This message is passed to the processor

containing the partition with identical L and M coordinates at K=2, where it is unpacked. For instance, processor 1 would send the message to processor 16. The forward sweep is continued with all processors operating at K=2. The forward sweep ends when the process is complete in the ξ direction (K=4).

Backward substitution proceeds in a similar manner, but in the reverse direction. Upon completion of the backward substitution at each K level, a message is created on each processor containing elements of the solution vector from all ξ-line segments in the partition. This message is passed back to the processor containing the preceding K partition, and the process is repeated until the inversion is complete at K=1.

The other two factors are treated similarly, except that the wavefront will move in each of the remaining coordinate directions.

Three-dimensional multi-partitioning has some distinct advantages. The method is almost perfectly load balanced. If properly applied, the number of grid points can vary by at most one across the partitions in each direction, and every processor has exactly one partition at every K, L, and M level. Thus, for all three factors, each processor will have very nearly the same amount of work as any other, and no processor will sit completely idle. The load balance also benefits from each processor having exactly one partition on each boundary of the grid when applying boundary conditions. As a result of the careful partitioning described above, the communication pattern for each processor is invariant; each processor "slides" up and down its diagonal in the partition array while keeping the same six neighboring processors.

In general, there are some costs for the 3D multi-partition method. There are more processor interface points, and thus a greater message volume, than would be found if the partitioning scheme allocated a single partition to each processor Memory must be managed dynamically to maintain flexibility, increasing coding

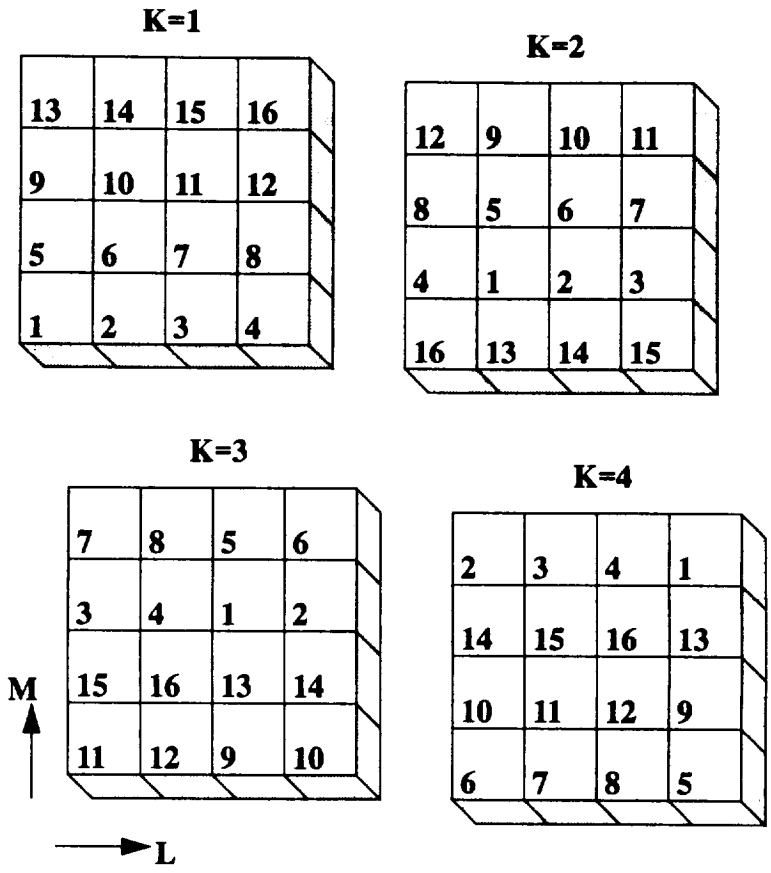


Figure 1: 3D diagonal partitioning scheme for a 4^2 processor system. The number on the partition indicates to which processor it is assigned.

complexity, and lowering run-time efficiency on a per-processor basis. And while it has not been stated explicitly, it is necessary to use an integer-squared number of processors to achieve a balanced work load.

The Virtual Processor

The integer-squared limitation is really only a problem on closely coupled machines which do not allow multiple user processes per processor. On a workstation, the number of processes is not so severely limited. By starting multiple processes on each processor of the system and treating each process as if it were an independent "virtual processor," a non-square number of processors may be accommodated. The simplest method of assigning processes to processors is to start n_p processes on each of n_p machines resulting in n_p^2 virtual processors, satisfying the limitations of the method. It is clear that as the number of machines increases, the size of the partitions on a particular grid will fall rapidly, resulting in increased message traffic and decreased efficiency.

Instead, MEDUSA-MP is designed to allocate virtual processors such that the number of virtual processors per processor never exceeds a preset limit (*mvp*). An integer-squared number is searched for between one and $mvp \times n_p$, such that the total work is most nearly balanced across the processors. All processors may not have the same number of processes running, leading to idle time, but the reduction in message traffic can offset this loss.

It is possible, even likely, that all of the cluster processors will not be the same. If an equal number of partitions are given to each processor, regardless of processor speed and memory, idle time will result. The concept of the virtual processor can help ease this problem. By distributing more virtual processors to faster machines, a better load balance can be achieved on a heterogeneous system.

Partition Boundary Treatments

A modification to the diagonalized Beam-Warming scheme is made in MEDUSA-MP to reduce significantly the message volume and the per-processor memory requirements. The dissipation scheme switches at each partition boundary from mixed second and fourth-order to pure second-order in the direction normal to the boundary.

This change allows a layer of overlap cells to be removed from each partition boundary, saving memory. Reduced communication volume comes from two sources. The extra overlap cells no longer need updating, and the matrix system may be dropped from scalar-pentadiagonal to scalar-tridiagonal at the boundary faces. Table 1 summarizes the total communication volume per interior partition per iteration for both diagonalized schemes, with data for the block-tridiagonal scheme for reference.

The penalty for this improvement in memory utilization and message volume is that the steady-state solution will no longer be identical to one produced by the block Beam-Warming scheme. This difference does not

Scheme	Partition Boundary Dissipation	Forward Sweep (words/line/ partition)	Backward Substitution (words/line/ partition)	Three Factor Total (words/ partition)
Block Beam-Warming	Mixed 2nd and 4th-order	25 matrix + 10 RHS	10 solution	$45(S_\xi+S_\eta+S_\zeta)$
Diag. Beam-Warming	Mixed 2nd and 4th-order	12 matrix+ 10 RHS	10 solution	$32(S_\xi+S_\eta+S_\zeta)$
Diag. Beam-Warming	2nd-order	3 matrix+ 5 RHS	5 solution	$13(S_\xi+S_\eta+S_\zeta)$

Table 1: Inter-partition communication summary for selected solution schemes. S_ξ, S_η, and S_ζ represent the number of grid points on a plane normal to ξ, η, and ζ, respectively.

necessarily represent a loss in accuracy; only the form of the artificial dissipation has been modified. It is recognized, however, that the modified scheme is more dissipative. This effect will be magnified as the number of processors increases, suggesting that this modification may be inappropriate for use on massively parallel processors. However, the number of processors used in a distributed system is usually orders of magnitude less, meaning that the increased dissipation will occur only on a few grid-planes through the solution. Further, the better connectivity of a dedicated multi-processor could make this modification unnecessary. The modified dissipation's effect on the stability of the scheme has not yet been determined.

Computing Environment

Measuring the performance of PVM jobs on multi-user systems is a somewhat tricky business. CFD computations on structured grids should use static load balancing, since this has the greatest potential for speed-up. But statically balanced loads can be skewed dramatically if multiple users are allowed on a cluster simultaneously. A dedicated cluster ought to be used, which schedules one distributed job after another, and does not allow additional (interactive) use.

Due to practical limitations, such a dedicated cluster was not available, so a special strategy had to be adopted which runs counter to common engineering practice. Rather than running very long simulations (many iterations) in order to average out system fluctuations, very short runs are used. Long runs provide a good view of the statistical average of jobs run on the cluster with a typical user load, which is exactly what is *not* wanted. Short runs may hit a period of high load, but will occasionally take place in a time window when not much else is happening, thus providing a good view of what a dedicated system can deliver. Moreover, the ensemble *minimum* of wall clock time spent is used, rather than the ensemble average over many simulations. Again, this is because the average predicts loaded-system performance, while the minimum shows dedicated-system performance.

Timing runs were made on two clusters at NASA Ames. The RFA cluster consists of 22 Silicon Graphics, Inc. workstations with either the R4000 or R4400 processor running at 50MHz and 75MHz respectively. Each machine has at least 64MB of RAM. The workstations are connected by an Ethernet consisting of three subnets. As depicted in Fig. 2, two of the subnets, 34 and 51, are essentially independent while the third, subnet 49, reaches the FDDI backbone by way of subnet 51.

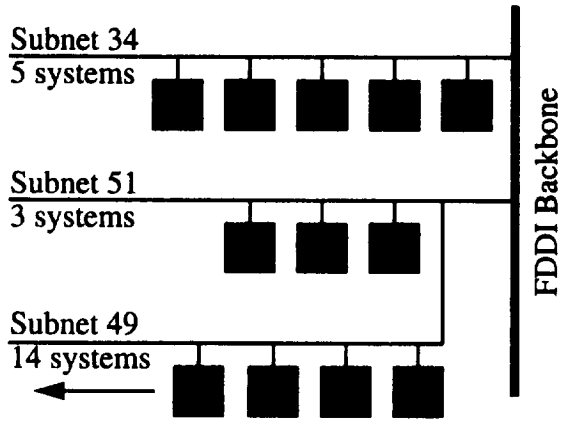


Figure 2: Network topology of the RFA Cluster.

No effort at all has been expended to achieve an optimal cluster arrangement. The cluster is built from existing systems, and the placement of a system on a particular subnet is a function of the geographic location of the processor.

The second cluster, referred to as the SPS cluster, consists of four Silicon Graphics, Inc. 4D/380S workstations, each containing eight R3000 processors running at 33MHz. Although the eight processors per machine use a single shared memory, PVM 3.2.3 does not take advantage of this hardware feature and passes all communications through the local PVM daemon anyway. Consequently, for communication purposes the cluster consists of 32 processors, although some messages will be carried on internal lines. The SPS cluster provides both Ethernet and FDDI networks.

Results

Flow Solver Performance

The test problem for this work is the subsonic flow over a flat plate. This two-dimensional flow is expanded in the third coordinate direction by copying the grid as many times as necessary for a reasonable sized computation.

It must be emphasized that the focus of this work is on the computational performance of the flow solver. And while this is essentially a

two-dimensional geometry, the flow solver is fully three-dimensional. It was felt that this simple geometry provides a sufficient test for timing purposes. The flat plate grid has 61 points in the streamwise direction, and 51 in the surface normal direction. Multi-partition computations use a $61 \times 51 \times 64$ point grid. Due to memory restrictions on the RFA cluster, single partition computations have been carried out on grids of $61 \times 51 \times 25$ points. Computations on machines identical to those of the RFA cluster, but with larger memory, indicate that little or no loss in baseline processor performance is incurred by going to the larger grid.

To demonstrate the effects of the modified boundary smoothing, otherwise identical runs are made using MEDUSA-MP and OVERFLOW-PVM. Four processors are used by MEDUSA-MP, while one is used by OVERFLOW-PVM. Figure 3 shows computed skin

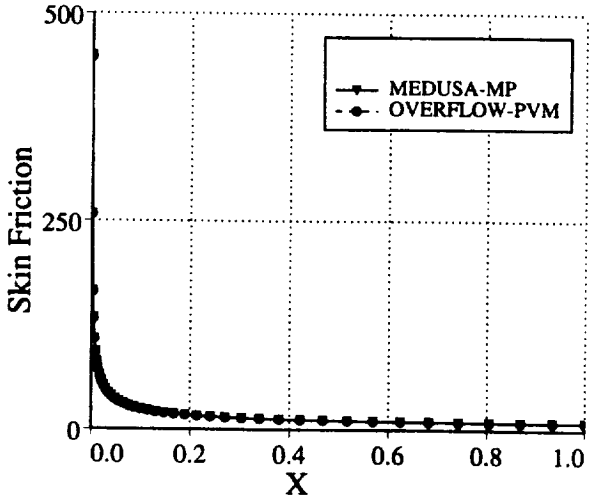


Figure 3: Non-dimensional skin friction as a function of streamwise station for laminar flat plate flow. $M_{f.s.}$=0.2, Re=10,000. Comparisons between solutions from MEDUSA-MP and OVERFLOW-PVM.

frictions from both codes. While there are slight differences in the solutions, the difference between the skin friction values does not exceed 0.3%.

Figure 4 is a plot of performance, measured in Mflops (Million floating-point operations per second), versus the number of processors used in the computation. Mflops were derived from the number of operations performed on interior grid points (2105op./pt./iter.) and the elapsed wall-clock time. Boundary point updates involve relatively few operations, and are ignored. MEDUSA-MP on the RFA cluster (inverted triangles) shows nearly linear speed-up with increasing number of processors up to four. Above this, losses due to Ethernet and PVM begin to be reflected in the performance, eventually reducing incremental speed-up to near zero.

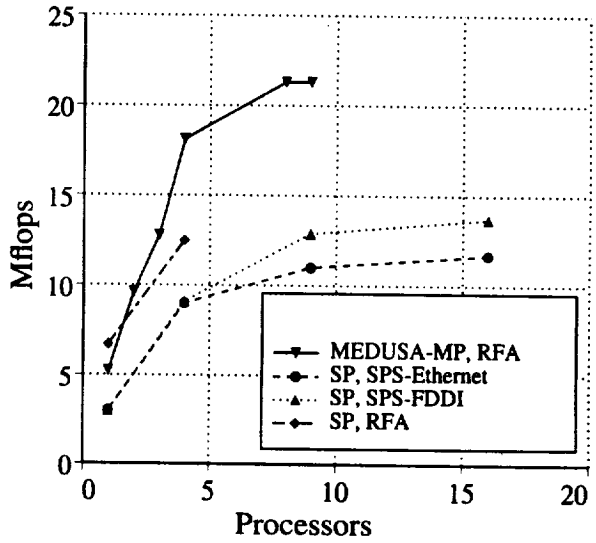


Figure 4: Cluster performance as a function of processor count.

The effect of Bandwidth Granularity (G_b) is seen in Fig. 5. As anticipated, increasing G_b by reducing the number of virtual processors generally improves efficiency, but not always. It must be noted that due to the virtual processor allocation method in MEDUSA-MP the granularity of a two processor case is identical to that of four processors. Similarly, three and nine processors have the same granularity. Clearly the efficiencies are not the same. The computation of G_b does not take into account the actual number of processors involved in the computation, only the number of virtual processors. The communication rate between two processes on a single processor usually exceeds that of Ethernet, so efficiency is not truly a function of Bandwidth Granularity. While efficiency generally is a function of the number of actual processors, it can vary widely depending upon the performance of an individual system relative to the speed of the network.

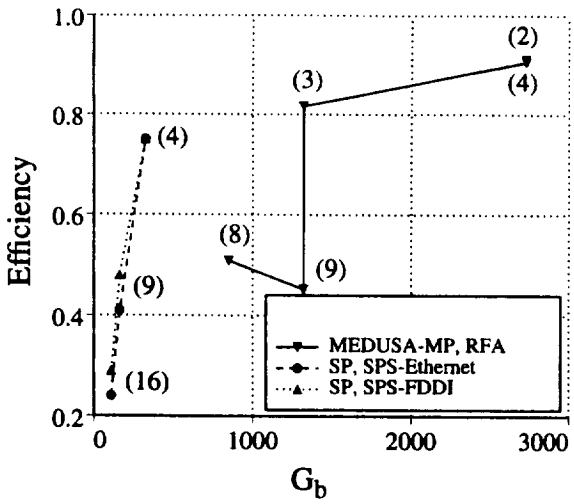


Figure 5: Efficiency vs. Bandwidth Granularity for MEDUSA-MP and the SP parallel benchmark. Numbers in parentheses indicate number of processors.

NAS SP Parallel Benchmark

To compare the performance of the SP benchmark with MEDUSA-MP, a grid size of $61 \times 51 \times 64$ points has been chosen. This represents a medium-sized single grid for most CFD computations. As these dimensions do not divide nicely among most of the configurations of small numbers of processes, a slight load imbalance is introduced. Performance is affected only minimally.

All computation rates indicated in Fig. 4 were collected using runs of 0, 1 and 5 iterations. Like MEDUSA-MP, the Mflops rating for the SP benchmark is derived from the number of operations per iteration (835) for interior grid points. Some operations are also needed for boundary points, despite the trivial boundary conditions. These operations are ignored. Numbers of processes on the SPS cluster larger than 16 have not been attempted since the incremental performance gain is obviously becoming negative.

Single-processor baseline computations on the RFA cluster were done on a $61 \times 51 \times 45$ point grid in order to avoid swapping due to the limited amount of memory on the machines, which would inordinately disadvantage the single-processor performance. It was found that the baseline processing speed is nearly independent of grid size for a wide range of grids containing 32^3 points or more. As in the SPS cluster case, computations on multiple processors were carried out on a grid containing $61 \times 51 \times 64$ points. No more than four machines of the RFA cluster were employed successfully in any of the computations of the SP benchmark. The failure of larger jobs appears to be hardware dependent.

One unexpected result which must be pointed out is the relatively small gain in performance achieved by using FDDI on the SPS cluster. For four processors, there is no improvement at all, and only slight improvement over Ethernet for nine and sixteen processors.

Analysis

At first glance the performance information given in Figs. 4 and 5 might suggest that MEDUSA-MP is a more efficient code than the SP benchmark. In reality, much more time was spent optimizing the communication of the SP code. But, because of the greater number of operations per point per iteration in MEDUSA-MP, it spends relatively less time communicating. Another variable in the problem is the cluster on which each code was run. The communication bandwidth for Ethernet, relative to processor speed, is greater on the SPS cluster than on the RFA cluster. Message packing and unpacking speeds are also very different.

A parameter which can account for algorithmic differences, cluster differences, and implementation differences is clearly needed. One possiblity is the optimal computation to communication time ratio (φ) defined:

$$\varphi = \frac{G_b B_{com}}{M_w \sum_{i=1}^{n_p} S_{p_i}} = \frac{t_{comp}}{t_{comm}}$$

where M_w is the size of the computational word in bytes, and S_p is the speed of an individual processor. It is assumed that communication is not masked by computation, and that there is no contention for network resources.

The behavior of φ for MEDUSA-MP and the SP benchmark running on the RFA cluster and the SPS cluster are shown in Fig. 6. The jagged appearance of the MEDUSA-MP curve reflects the virtual processor allocation algo-

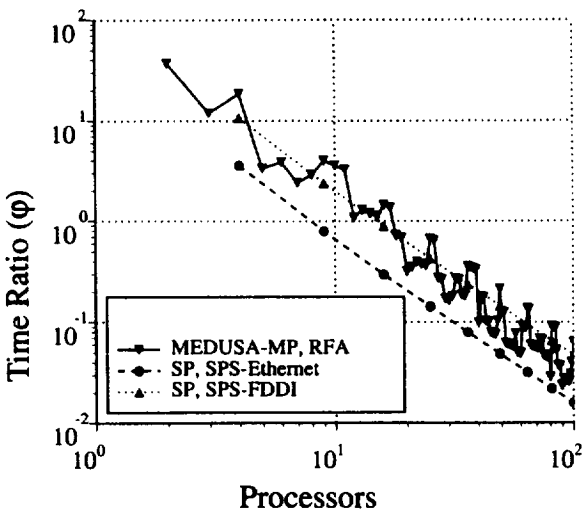


Figure 6: Computation to Communication time ratio for MEDUSA-MP and the SP benchmark.

rithm and its effect on the granularity of the partitioning.

For equivalent numbers of processors, the time ratio for the SP benchmark on the SPS cluster is nearly an order-of-magnitude less than that for MEDUSA-MP on the RFA cluster. The SP-SPS combination, even in an optimal implementation, is less able to use all available computational resources. In this light, the efficiency curves shown in Fig. 5 make some sense. Indeed, if the parallel efficiency is plotted versus φ (Fig. 7), a more clear

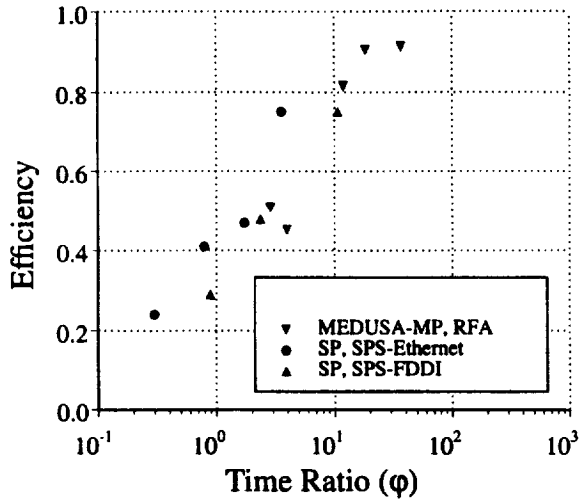


Figure 7: Parallel efficiency as a function of the time ratio (φ) for MEDUSA-MP and the SP benchmark.

comparison between the codes may be made. It would appear that on the SPS cluster using Ethernet, the SP code is more nearly optimal.

This is not true when the same code and cluster uses FDDI. The SP code is designed such that much of the network overhead is masked by computation. For four processors, the code is able to mask most all of the network costs. All of the communication penalty is coming from the packing and unpacking of messages. As the processor count climbs, there is less computation to mask the network costs. The system will incur idle time using Ethernet earlier (fewer processors) than with FDDI, so efficiency improvements will only appear after a processor count threshold is exceeded. It is clear that some correction must be made to the formula for φ to account for masked communication.

Another anomaly which cannot be explained by differences in φ may be seen in Fig. 5. When running MEDUSA-MP on eight processors, though G_b and φ are less, one achieves better efficiency than when running on nine. The behavior may be explained partially by noting that there are two processes on each processor in the eight processor case. This allows intra-processor communication, and overlap of communication and computation. While one process is waiting for a message, the second can be executing, reducing idle time on the processor and improving efficiency.

The time ratio is not completely successful in coalescing the efficiency data. It does not fully represent implementation details like communication masking and virtual processors. Work in this area remains.

One use of the time ratio is for predictive purposes. For instance, if FDDI is installed on the RFA cluster, it is anticipated that more processors could be used efficiently. From the plot, the time ratio at which MEDUSA-MP should operate at 50% efficiency is approximately 5. If one concentrates on the perfect-square processor counts, G_b may be approximated as:

$$G_b = 6912 n_p^{-3/4}.$$

Re-arranging the formula for φ, and assuming the packing and unpacking bandwidths are the same, gives the relationship:

$$n_p = \left[\frac{6912}{\varphi M_w S_p} \cdot \frac{B_{net} B_{pack}}{B_{net} + 2B_{pack}} \right]^{4/7}.$$

The R4000 processor of the RFA cluster is capable of packing data at a rate of 25MByte/second and runs a single partition solution of MEDUSA-MP at 5.24Mflop using 8Byte words. FDDI has a peak bandwidth of 12.5MByte/second, so one could reasonably assume that twenty-one processors could be effectively used with FDDI. As the approximation for G_b only used perfect-square processor counts, sixteen would be more practical, and eighteen might also work.

Conclusions

It has been established that a multi-partition flow solution algorithm can run efficiently on small clusters of moderately powerful workstations using Ethernet. This method will be useful for improving parallelism and load balancing in existing grid-level parallel codes. A reduction of the order of the artificial dissipation at partition boundaries has little effect on solution accuracy for small numbers of processors, and greatly reduces the volume of communication.

Communication is quite clearly the bottleneck in the implementations discussed here. A large portion of the bottleneck occurs on the processor in the form of byte-copying overhead. Better communication packages are needed.

A time ratio parameter has been introduced which, with additional work, may be useful in comparing the implementation efficiency of parallel solution algorithms. Additionally it could be used to predict the performance of a particular code on an untested computer system.

References

[1]Mattson, T. G., Douglas, C. C., Schultz, M. H., "A Comparison of CPS, LINDA, P4, POSYBL, PVM, and TCGMSG: Two Node Communication Times," YALEU/DCS/TR-975, Yale University, New Haven, Connecticut, May 1993.

[2]Geist, A., Beguelin, A., Dongarra, J., Weicheng, J., Manchek, R., Sunderam, V., "PVM 3 User's Guide and Reference Manual," ORNL/TM-12187, Oak Ridge, Tennessee, May 1993.

[3]Bokhari, S. H., "Complete Exchange on the iPSC-860," ICASE Technical Report 91-4, NASA Langley Research Center, Hampton, Virginia, 1991.

[4]Saphir, William C., "Message Buffering and its Effect on the Communication Performance of Parallel Computers," NASA Ames Technical Report RNS-94-004, April 1994.

[5]Smith, M. H., Chawla, K., and Van Dalsem, W. R., "Numerical Simulation of a Complete STOVL Aircraft in Ground Effect," AIAA-91-3293, Baltimore, Maryland, September 1991.

[6]Rizk, Y. M., and Gee, K., "Numerical Prediction of the Unsteady Flowfield Around the F-18 Aircraft at Large Incidence," AIAA-91-0020, Baltimore, Maryland, September 1991.

[7]Smith, M. H., Pallis, J. M., "MEDUSA - An Overset Grid Flow Solver for Network-Based Parallel Computer Systems," AIAA-93-3312CP, Orlando, Florida, July 1993.

[8]Naik, N. H., Naik, V. K., Nicoules, M., "Parallelization of a Class of Implicit Finite Difference Schemes in Computational Fluid Dynamics," *International Journal of High Speed Computing*, Volume 5, Number 1, 1993.

[9]Van der Wijngaart, R. F., "Efficient Implementation of a 3-Dimensional ADI Method on the iPSC/860," Supercomputing '93, Portland, Oregon, November 1993.

[10]Pulliam, T. H., and Chaussee, D. S., "A Diagonal Form of an Implicit Approximate-Factorization Algorithm," *Journal of Computational Physics*, Volume 39, Number 2, February 1981.

[11]Buning, P. G., and Chan, W. M., "OVERFLOW/F3D User's Manual, Version 1.6," NASA Ames Research Center, March 1991.

[12]Chaderjian, N. M., and Guruswamy, G. P., "Unsteady Transonic Navier-Stokes Computations for an Oscillating Wing Using Single and Multiple Zones," AIAA-90-0313, Reno, Nevada, January 1990.

[13]Bailey, D., Barton, J., Lasinski, T., Simon, H., "The NAS Parallel Benchmarks," NASA Ames Technical Report RNR-91-002, August 1991.

NAS

Three Implementations of the NAS Scalar Penta-diagonal Benchmark

Rob F. Van der Wijngaart *, Thanh Phung †, Eric Barszcz ‡
NASA Ames Research Center, Moffett Field, CA 94035

Abstract

Three viable strategies for the implementation of the NAS Scalar Penta-diagonal Parallel Benchmark on MIMD systems are investigated, namely transposition, pipelined Gaussian elimination, and multi-partitioning. Hardware platforms considered are the Intel Paragon XP/S-15, and a cluster of SGI workstations on Ethernet, communicating through PVM. It is found that the multi-partitioning strategy offers the kind of coarse granularity that allows scaling up to hundreds of processors on a massively parallel machine. Moreover, efficiency is retained when the code is ported verbatim (save message passing syntax) to a PVM environment on a modest size cluster of workstations.

1 Introduction

Parallel benchmark performance results often bear little relation to realistic applications. The reasons for that are manifold, even when taking into account that implementors are sometimes willing to go to extremes to tune their codes, and will often squander space in order to save a few extra operations. Perhaps the single most important factor in reducing the value of most benchmark problems is the absence of data distribution incompatibilities. To remedy this deficiency the NAS Parallel Benchmarks [1] include a set of problems derived from realistic applications from the area of computational fluid dynamics. These original applications have been pruned so that they can be described completely in just a few pages. It may appear that this reduction of complexity does not affect the characteristics of the applications; roughly the same amount and type of communications is required by both applications and benchmarks. But this is exactly what causes the problem. Arithmetic operations are eliminated while virtually all data dependencies are retained, which places a heavier emphasis on the communication properties

*Employee of MCAT Institute
†Employee of Intel Corporation
‡Employee of NASA Ames, RNR branch

of the scheme used to implement the benchmark than on that for the application. Just reducing the data dependencies at the same rate as the overall arithmetic complexity does not necessarily increase the realism of the benchmarks. Different parts of the application are impacted differently by a change in arithmetic and communication loads. And even when the relative granularity (ratio of computation over communication volume) for all parts of the benchmark are the same as that of the application, latency will still skew the performance of the benchmark.

Hence, the value of the conclusions one can draw from the implementation of the NAS parallel benchmarks should not be overestimated. But we maintain that these model problems are among the best available, and interesting lessons can be learned from them.

2 Description of SP benchmark problem

The problem selected for this paper is the Scalar Penta-diagonal (SP) benchmark, which is derived from the diagonalized Beam-Warming approximate factorization scheme for the computation of three-dimensional viscous fluid flow using a structured discretization mesh. SP is a variation on the class of alternating direction implicit (ADI) schemes. A precise description of the problem is provided in [1], including the various simplifying assumptions made. We will outline the benchmark to identify the major computation and communication tasks.

The form of the discretization equations is:

$$\mathbf{B}_1 \{\mathbf{I} - \mathbf{L}_{x_3}\} \mathbf{B}_2 \{\mathbf{I} - \mathbf{L}_{x_2}\} \mathbf{B}_3 \{\mathbf{I} - \mathbf{L}_{x_1}\} \mathbf{B}_4 \Delta U = \mathbf{RHS}(U) \tag{1}$$

This system is solved for the update vector ΔU, which consists of five values for every grid point. The known value of U itself is the result from the previous time step. $\mathbf{I}$ is the identity, and the matrices $\mathbf{L}_{x_i}$ are one-dimensional discrete operators of a special structure, derived from the diagonalization of the Beam-Warming scheme [5]; they comprise difference stencils of size five in the x_i-direction, but do not mix the different components of the ΔU vector. Consequently, inversion of each of the factors $\{\mathbf{I} - \mathbf{L}_{x_i}\}$ results in the solution of a set of five independent families of—again—independent scalar penta-diagonal equations, one for each grid line in the x_i-direction. Among these five families there are three that have the exact same difference formulation due to a coincidence of three eigenvalues resulting from the diagonalization, so the number of families to be inverted per x_i-direction is actually only three. The $\mathbf{B}_i$ are block-diagonal matrices that do not involve any spatial differencing, but that couple all five components of the ΔU vector. Finally, the **RHS** matrix is a three-dimensional difference operator of size five in all coordinate directions. It also mixes all five components of the ΔU vector, but it does not have to be inverted since U is known. It should be noted that the coefficient matrices $\mathbf{B}_i$, $\mathbf{L}_{x_i}$, and **RHS** all depend on U, so they have to be (largely) recomputed for every time step.

Boundary values are kept constant (Dirichlet conditions), so that system (1) is solved only for interior grid points. The amount of computational work per interior point involved in the different parts of the system depends on the particular implementation. The transpose (TR) and multi-partition (MP) methods described in section 4 have the same operation count, but the pipelined Gaussian elimination (PGE) differs in a few places due to fewer stored intermediate results. Operation counts are shown in Table 1.

	$\pm$		$\times$		/		$\sqrt{}$	total	
	MP	PGE	MP	PGE	MP	PGE	all	MP	PGE
$\mathbf{B}_1^{-1}$	11	12	17	19	0	2	0	28	33
$\mathbf{B}_2^{-1}$	5	6	3	3	0	0	0	8	9
$\mathbf{B}_3^{-1}$	5	6	3	3	0	0	0	8	9
$\mathbf{B}_4^{-1}$	10	12	17	19	0	1	0	27	32
$\Sigma_{i=1}^{3}\{\mathbf{I}-\mathbf{L}_{x_i}\}$	48	87	36	51	0	0	0	84	138
$\Sigma_{i=1}^{3}\{\mathbf{I}-\mathbf{L}_{x_i}\}^{-1}$	96	96	129	129	9	9	0	234	234
RHS	225	230	213	273	2	1	1	441	505
$U+\Delta U$	5	5	0	0	0	0	0	5	5
total	305	454	418	497	11	13	1	835	965

Table 1: Operations per interior point for MP/TR and PGE

Clearly, most computational work is done forming the right hand side vector (**RHS**), followed by inversion ($\{\mathbf{I}-\mathbf{L}_{x_i}\}^{-1}$) and construction ($\{\mathbf{I}-\mathbf{L}_{x_i}\}$) of the one-dimensional difference operators. The block-diagonal inverses $\mathbf{B}_i^{-1}$ can be generated directly without constructing the $\mathbf{B}_i$ explicitly. Note that each phase of the SP can be executed for all grid points independently, except for the inversion of the scalar penta-diagonal systems. During inversion points on the same grid line are coupled, and a certain sequential dependence is incurred.

As an aside, we remark that a minor error appears in the NAS Parallel Benchmark document [1], which is also contained in the SP sample code available from NAS. The initialization formulas (3.8) and (3.9) in [1] do not constitute a transfinite trilinear interpolation of the boundary values, which can be verified most easily by substituting a constant value of C for $u_0^{(m)}$ on the boundary. The resulting initialization formula is: $u_0^{(m)} \equiv 3C-3C^2+C^3$, which only reduces to C if $C = 0$, 1, or 2. The correct interpolation formulas are:

$$P_{\xi_i}\,\phi = (1-\xi_i)\,\phi\,|_{\xi_i=0} + \xi_i\,\phi\,|_{\xi_i=1} \tag{2}$$

and

$$u_0^{(m)}(\xi,\eta,\zeta) = (P_\xi + P_\eta + P_\zeta - P_\xi P_\eta - P_\xi P_\zeta - P_\eta P_\zeta + P_\xi P_\eta P_\zeta)\,u_0^{(m)}\,. \tag{3}$$

It should be noted that the verification values for residual norms and solution errors supplied in the NAS sample code are based on the erroneous initialization formulas. Obviously, computational performance is not affected.

3 Hardware platforms and communication protocols

Efficiency of programs run on sequential computers is often strongly influenced by data regularity and locality. Good programmers do not hesitate to change loop orderings to take better advantage of vectorization, or to better utilize cache; it is deemed acceptable to take (partial) responsibility for data management within the computer in order to improve performance, although ideally one would like to leave such matters up to compilers. In the case of parallel distributed-memory computers, management is further complicated due to the fact that local memory has much faster access than remote memory. Certain highly specialized architectures can provide reasonably fast access to remote memory, but at the expense of the rigidity of regular data sizes and lay-out on the set of processors available, while still requiring expensive dedicated switching networks. An example is the execution of data-parallel programs on SIMD (Single Instruction/Multiple Data) machines. SIMD machines require some directions from the programmer how to distribute data, but take care of any data transport themselves, just like conventional computers.

Good results have been reported for certain applications on those machines. But if the problem at hand features little regularity, or if the hardware available has slow interconnections, more user interference is required. The MIMD (Multiple Date/Multiple Instruction) computing model on distributed-memory machines with explicit message passing (we will refer to this situation as the MIMD model, for brevity) gives the user complete responsibility for the data lay-out and inter-processor data movement. The benefit of assuming this responsibility is flexibility and control; the price is an increased programming burden, and usually relatively slow communication. Therefore, algorithms designed to run on MIMD machines need to minimize communications, both in number and volume. Once this fact is understood, portability of MIMD programs is eased, since message passing syntaxes are very similar on most current platforms.

In this paper we investigate three different strategies for implementing the SP on different MIMD architectures. The main machine used for comparison of these methods is an Intel Paragon XP/S-15 with 208 nodes of 32MB RAM and a 16KB 4-way associative data cache each. The communication protocol used is NX, running under the operating system OSF/1. Typical latencies and bandwidths on this machine are 112 μs and 35 MB/s, respectively, regardless of the distance traveled between nodes. The Paragon i860XP chip features two vector pipelines for addition and multiplication, which can be controlled using compiler flags.

From this comparison we draw conclusions regarding the viability of each of these methods on very-high-latency networks, and select the most promising for implementation on a network of workstations. The configuration chosen here consists of 4 Silicon Graphics

workstations with 8 MIPS R3000 33MHz processors each. Main memory is 256 MB of shared memory for each machine, with a primary data cache of size 64 KB and a secondary data cache of size 256 KB. Connectivity is established through Ethernet. The syntactic communication protocol is PVM 3.2 (Parallel Virtual Machine), which features typical latencies and bandwidths of 2500 μs and 0.58 MB/s, respectively. No vectorization is available on the workstations. Interestingly, the relative latency on the network of workstations (i.e. the product of latency and bandwidth) is lower than that on the Paragon.

4 Algorithms

In the context of distributed parallel computing, an algorithm does not simply constitute a list of arithmetic operations to be carried out plus an ordering, but also a description of the data lay-out and the communication strategy. The three algorithms considered here are—highly tuned—variations of schemes that were compared earlier in [8] for another application.

4.1 Transpose method

In the current implementation of the transpose (TR) method (also called dynamic block-Cartesian [8], or global exchange), each processor gets assigned a single plane of data of the whole grid during each of the phases of the solution of system (1). Consequently, the number of processors p must equal the number of planes in the grid in all directions. Since the SP specifies cubic grids of linear dimensions of 64 and 102, respectively, the number of processors employed must be 64 or 102 also. If more processors are to be used, then they must be an integral multiple—called a— of the number of planes in the grid, and each processor receives a contiguous cut from one whole plane that stretches the grid completely in one coordinate direction. Again, the data space owned by each processor has a thickness of just one point. Whereas the particular orientation of such a (partial) plane is immaterial during most parts of the SP, it is essential that it be aligned with the discretization operator direction during the inversion of the scalar penta-diagonal systems. Since these systems are solved in three directions, the data planes have to be rotated—*transposed*—in between these inversion phases. On a grid of size n^3 this takes place by letting every processor send $n-1$ lines of data to the $n-1$ processors that need them, while also receiving $n-1$ lines from the other processors.

A complication of TR is caused by the non-linearity of the problem. It is not sufficient simply to transpose the intermediate solutions of system (1) as the left-hand-side operators are inverted successively from left to right. Since the operators depend on U themselves, U has to be transposed along, doubling the communication load.

Some savings can be obtained if the number of processors matches the grid size exactly ($a=1$). First, each plane contains whole grid lines in *two* directions, obviating the need

for transposition between at least one pair of inversions. Second, if two time steps are combined, another transposition can be saved. The solution directions in two time steps are as follows: $z \rightarrow y \rightarrow x \rightarrow z \rightarrow y \rightarrow x \rightarrow \cdots$. These directions can be kept in-processor if the chain of planes plus transpositions is defined as follows (T signifies a transpose): $zy \xrightarrow{T} zx \xrightarrow{T} xy \xrightarrow{T} \cdots$, requiring only three flips of U and two of the intermediate result for every two iterations. Flips of U and the intermediate results can actually be combined to reduce latency, for a total of three per two iterations. Also, for computational efficieny the reciprocal of the first element of the U vector is transposed along with U (the divide operator is very expensive). If the number of processors exceeds the number of planes, then additional transposes within planes are needed to bring the 'broken' dimension in processor.

The communication requirement per processor per iteration is relatively easy to determine. The only data transfers are during the flips, and before assembling the right hand side vector, where each processor receives and sends four data planes (two from/to each side). Assuming a processor owning an interior plane (with neighbors on each side), then the total number of messages is:

$$Nmsg^{TR} = 3(n + a - 2)/2 + 4\,, \tag{4}$$

while the total number of bytes transferred (1 word = 8 bytes) is:

$$Vmsg^{TR} = \frac{n^2}{a}\left(252 + 60a + 60(a-1)\frac{a+2}{a}\right) - 60\frac{n}{a}(a+1)\,. \tag{5}$$

4.2 Pipelined Gaussian elimination

In the current version of the pipelined Gaussian elimination (PGE) method (also called static block-Cartesian [8]), each processor receives a single grid block whose dimensions are as close to cubical as possible. Since this generally means that no single grid line is fully contained within a block, some way of keeping processors busy while a grid line is 'swept out of the block' is needed. The method adopted here is two-way pipelined Gaussian elimination. The inversion procedure for a grid line is essentially Gaussian elimination for a banded system of width five.

In order to reduce message overhead grid lines in each of the three coordinate directions are grouped into pencils that are treated as units. Each processor contains segments of a number of pencils. The two-way pipeline is started on the same pencil at each end by different processors on opposite sides of the grid. Once each has finished its segment of the pencil, it passes interface information to the neighboring processor and starts at the extreme end again of the next pencil. Interior processors behave similarly, except that they also have to receive information for each pencil. Some special coding is required to combine results where the two ends of the pipeline meet. The optimum pencil size depends on the amount of work on a grid line, and consequently the size for the forward elimination (G_f) will be different from that for the backsubstitution (G_b). This effect

is described in detail in, e.g., [4]. The benefit of two-way PGE, as opposed to one-way PGE, is that the start-up cost of the pipeline is halved. Moreover, if only two processors split a coordinate direction, then each need contain just one (segment of a) pencil without incurring a load imbalance.

Although the communication volume of PGE is the smallest possible, a disadvantage is that many messages need to be sent, one at the end of each pencil segment. For a processor assigned to a completely interior grid block the maximum number of messages sent per iteration on an n^3 grid with p processors is:

$$Nmsg^{PGE} = 3(1/G_f + 1/G_b)n^2/\sqrt[3]{p^2} + 6\,. \tag{6}$$

while the maximum number of bytes transferred is:

$$Vmsg^{PGE} = 1248n^2/\sqrt[3]{p^2}\,. \tag{7}$$

Typical figures for pencil sizes for simulations on the Paragon are $G_f = 8$ and $G_b = 16$, resulting in $Nmsg^{PGE} = 9n^2/(16\sqrt[3]{p^2}) + 6$.

4.3 Multi-partition

In the current version of the multi-partition (MP) method (also called Bruno-Cappello multi-cell [8]), each processor receives $\sqrt{p}$ grid blocks—cells—that effectively line up along a (logical) body diagonal of the grid, save periodic effects. This is called the 3D diagonal scheme [4]. Cells are positioned such that every coordinate plane that cuts the grid intersects with exactly one cell of each processor. Thus, there is always work to do for every processor during the sequence of penta-diagonal inversions. No transposition is needed, and the pipeline that results is very coarse grained and perfectly load balanced. Since in MP all processors have the same number of boundary and interior cells, they all send per iteration the exact same number of messages:

$$Nmsg^{MP} = 6\sqrt{p}\,, \tag{8}$$

and bytes:

$$Vmsg^{MP} = 1392n^2(\sqrt{p} - 1)/p\,. \tag{9}$$

(*the latter figure is not yet the absolute minimum; this will be remedied in the final paper*)

4.4 Comparison

Besides programmer proficiency and serendipitous compiler optimizations, there are five major factors determining the relative efficiencies of the algorithms described above: communication volume, number of messages sent, load balance, vectorizability, and cache use.

These factors can be analyzed in isolation, although it is their combination that determines overall efficiency.

If message passing is synchronous, the amount of time t_c spent on transferring k messages with an aggregate size of m bytes on a contention-free network can be expressed as: $t_c = k\alpha + m/\beta$, where α is the latency in seconds, and β is the message bandwidth in bytes/second.

When comparing communication loads we can draw up a division of the (n, p)-plane where either of the different algorithms is most efficient, depending on the value of the parameter $\alpha\beta$ (the relative latency). As it turns out, for no non-trivial grid size is there a point in (n, p)-space where TR has a lower communication load than the other two methods, regardless of the relative latency. Figure 1 shows how the other two methods divide the (n, p)-plane for the relative latencies of the Paragon and SGI/PVM configurations, respectively.

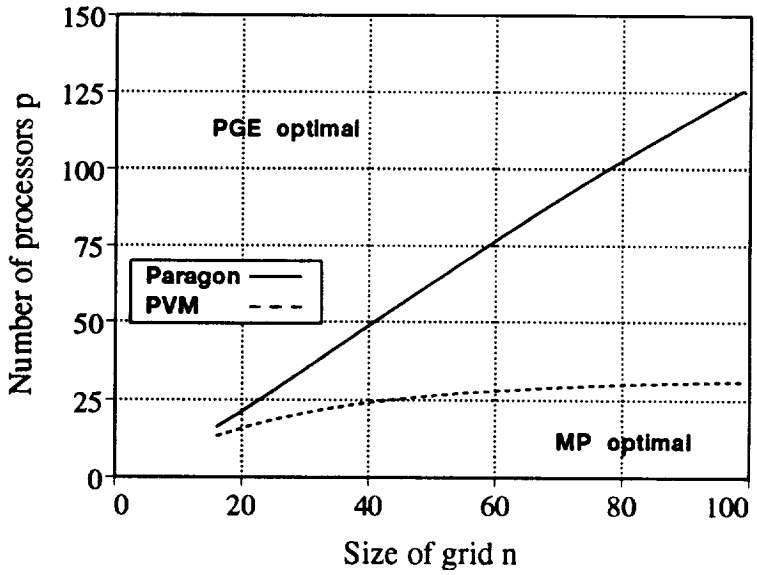


Figure 1: Regions of optimality for PGE and MP

Clearly, as the number of nodes keeps increasing for a given grid size, PGE will become more efficient than MP, whereas the converse is true when the number of processors is fixed and the grid size keeps increasing. This is largely a latency effect; even for infinite bandwidth (i.e. $\alpha\beta \to \infty$) Figure 1 remains qualitatively the same. Since the number of messages k for PGE is determined by the number of pencils passing through a block, k goes down as p increases, simply because the grid block decreases in size. At the same time, k goes up for MP, because the number of messages is proportional to the number of cells in each coordinate direction, which scales as $\sqrt{p}$. In the limit of $p = n^2$, each of the n MP cells per processor contains exactly one grid point, while a single PGE grid block still

contains n points, and has a face area of $\sqrt[3]{n^2}$ points; MP has become overfragmented, and should not be used.

The situation is reversed when the number of processors is kept fixed and the grid size is allowed to grow; k for PGE scales with grid size, so it keeps going up, while k for MP stays the same. It should be noted, however, that latency for PGE has been reduced by pencil grouping at the expense of extra idle time for pipeline fill, which is not accounted for in figure 1.

The other factors influencing efficiency are not so easily quantifiable. First, there is load balance. Whereas MP is perfectly balanced when $n \bmod \sqrt{p} = 0$, PGE and TR never are. The latter two always experience load imbalance from boundary effects ('interior' grid blocks/planes have different computation and communication requirements than 'boundary' blocks/planes).

Second, there is vectorizability. Put simply, the method with the longest vector length (inner loop length) wins. Since vector length is determined by the dimensions of planes, blocks, and cells, TR, PGE, and MP are ranked in this order for vectorizing efficiency.

Finally there is data cache utilization. Here the method that has the highest data locality wins, because more of the data used inside loops will fit in cache. TR and PGE have the same number of points per plane/block, but MP has a smaller number of points per cell, because each processor owns multiple cells. Consequently, for large n MP will always be the first to run loops completely in cache as p increases, provided the same number of data elements per grid point have to be loaded into cache for each loop in the three respective methods.

Somewhat different optimization routes have been taken by the implementors of the three methods, resulting in different numbers of auxiliary arrays that influence the operation count (Table 1), the memory requirements, and the number of data elements per point inside loops. Table 2 shows the total memory requirements for all processors combined for each of the three implementations. The table distinguishes between memory needed for a single processor-case, and the parallel overhead for multi-processor cases. Only terms that scale as n^3 or more in the worst case (smallest possible grain size) are included.

	single processor	parallel overhead	worst case
PGE	$18n^3$	$32n^2\sqrt[3]{p} + 48n\sqrt[3]{p^2} + 10324p$	$p = n^3$
MP	$38n^3$	$228n^2\sqrt{p} + 336np$	$p = n^2$
TR	$24n^3$	$(36a + 3)n^3$	$p = an = n^2$

Table 2: Memory requirements (double precision words)

In order to assess the effects of different non-communication-related optimization strategies on the performance of the parallel algorithms used, we also present results

for single-processor computations on several different machines (Table 3). An intermediate grid size ($n = 32$) was selected in order not to favor any particular algorithm; the maximum vector length is not large enough to disadvantage the cache-based PGE and MP methods, while the whole problem still cannot be run entirely in cache. For reference we also include results for the original serial code distributed by NAS*.

	IBM RS6000-590	Paragon XP/S-15	MIPS R3000
PGE	3.89	30.1	65.8
MP	4.92	38.5	76.7
TR	8.20	44.5	120.
NAS*	21.2	82.7	246.

Table 3: Single-processor cpu times for 10 iterations on 32^3 grid

As always, caution should be used in interpreting the data for purposes of scaling parallel results (see section 5.1). While PGE performs significantly better in the single-processor case than the other two methods, an important part of its advantage is due to a reduction of memory strides by performing data rearrangements (local transposes). But as the number of processors grows, grid blocks for PGE and MP shrink, and the stride problem lessens, leading to a loss of efficiency for PGE due to the copying involved in the local transposes.

5 Paragon computations

5.1 Performance models

In order to predict execution times for the three methods, models are constructed for their parallel performance based on single-processor computations and the communication figures presented in section 3. The general formula applied is: $t_{total} = t_a + t_c = t_a + k\alpha + m/\beta$, where t_a is the pure computational cost, which can often be written as $t_p(n-2)^3/p$, i.e. the computational cost per point, multiplied by the number of interior points in the grid, divided by the number of participating processors. When grids are reasonably large (greater than, say, 16^3) the total number of arithmetic operations is indeed proportional to the number of interior points in the grid, and t_p can often be considered constant. For MP the cell sizes shrink rapidly as p grows, and if small grids are used to predict their computational cost, a more complex performance model must be used that is based on operation count rather than on point count. Even then, it is found that for small grid sizes ($4 < n < 16$) the time per operation is not constant, but depends on n. Measurements of (n,Mflops)-pairs for some small grids are: $(5, 4.00), (6, 5.16), (8, 5.37), (12, 5.64), (16, 5.72)$. A crude fit to these results is: Mflops $= 6 - 4/(n-3)$. This can subsequently be used in

the parallel performance prediction (400 time steps) by replacing the grid size with the cell size, i.e.:

$$t_{total}^{MP} = 0.33\frac{(n-2)^3}{p(6-4/(n\sqrt{p}-3))} + 0.27\sqrt{p} + 0.016n^2(\sqrt{p}-1)/p \qquad (10)$$

The results of this prediction formula are included in Table 7. Performance models for the other two methods are under construction.

5.2 Results

Figure 2 shows the number of (millions of) interior grid points updated per second for the two classes of problems, i.e. $n = 64$ and $n = 102$, for each of the three methods. This figure is derived from the wall clock timings of 400 iterations (see Tables 5, 6, and

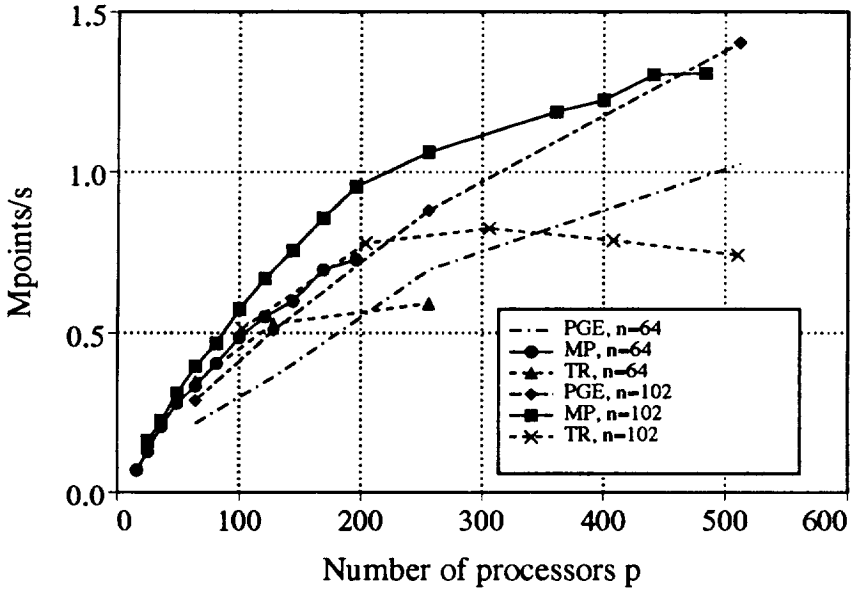


Figure 2: Performance of MP, PGE, and TR on Paragon

7), which are the more common—but less insightful—way of presenting SP performance data. The results are obtained by using the vectorization flag for TR. MP and PGE fared better when vectorization was disabled. It follows that MP is best as long as the problem does not become overfragmented. For the two problem classes tested this means that cells should be of size 5^3 or bigger.

An important reason for the success of MP that is not immediately apparent from its communication load is its ability of latency hiding. Since the granularity of MP is relatively coarse, namely at least the size of one whole cell, it is possible to execute send operations long in advance, do useful work within a cell, and postpone posting receives

for interface data until after that data has arrived; network contention and other latencies are generally greatly reduced or eliminated. And in the case of truly asynchronous communication, such as when a message co-processor (MCP) is used, there is not even a price to be paid for interrupts due to data packets arriving from the network. It has been observed that MP benefits about 5 to 10% from using the MCP, whereas PGE benefits to a lesser extent, and TR barely speeds up at all. The reason for this is as follows. All three methods can do some work computing (parts of) the right hand side vector while waiting for interface information to arrive from neighboring planes/blocks/cells, thus reducing the cost of that communication. But the penta-diagonal inversion process is too fine-grained for PGE to allow any overlap of computation and communication, and the flips in TR cannot be utilized efficiently either, because again the granularity is fine, and the transposed U needs to be fully assembled in order to compute after-flip coefficient matrices $\mathbf{L}_{x_i}$ and $\mathbf{B}_i^{-1}$. In contrast, MP offers opportunities for overlap during all stages of the solution process. Examine, for example, the forward elimination of the penta-diagonal inversion in the x_1-direction. While an interior cell is waiting for information from the processor that owns the adjacent cell on the left, it can compute the new $\mathbf{L}_{x_1}$ and block on receive only after it has formed its new left hand side. During the backsubstitution there are no more $\mathbf{L}_{x_1}$'s to compute, but now a processor can apply the block-diagonal inversion matrix $\mathbf{B}_2^{-1}$ to the latest completed cell while the next cell is waiting for interface data to arrive from the right. Analyses with the AIMS performance monitoring system [9] on the MP code running on the Intel iPSC/860 have shown that there is no latency noticeable for either problem class during any stage of the computation.

Interestingly, our positive experiences with MP run counter to the investigation reported in [4], whose authors obtain the best performance for PGE. Several factors can explain this discrepancy. First, Naik et al. [4] have used MP and PGE to implement the complete flow solver ARC3D instead of the SP benchmark that is based on it. More importantly, they have implemented a two-dimensional version of MP, which has poorer communication properties than the three-dimensional scheme used here. Not only is the number of cells per processor larger (more messages), but also the total communication volume is larger, leading to significantly higher data transmission costs and poorer scalability. Worst of all, during the penta-diagonal inversions messages are not consolidated on a per-cell basis, but are sent at roughly the same frequency as for PGE (see Table 9 in [4]), leading to a greatly inflated latency penalty. Consequently, we believe that a careful implementation of MP for ARC3D can lead to better performance than PGE, provided the number of processors does not get excessively large.

6 PVM results

Based on the Paragon results, MP was selected for a port to a network of workstations using Ethernet and PVM, version 3.2 [2]. The only part of the code that needed to be changed was concerned with the creation of processes and with message passing, which

accounted for less than 1% of the program text. All arithmetic and other logic was copied verbatim.

Although the 8 processors in each of the 4 workstations in the cluster use a single shared memory, PVM 3.2 does not take advantage of this hardware feature and routes all communications through the local PVM daemon anyway. Consequently, for communication purposes the configuration consists of a network of physically distinct processors connected through Ethernet.

The grid size chosen is $42 \times 42 \times 42$ points; this is smaller than the problem classes run on the Paragon. The reason for this is that the number of processors applied to a problem on a network of workstations is usually significantly smaller than that within a tightly-coupled massively parallel machine like the Paragon. In order to keep simulation times reasonably short a smaller grid must then also be selected, especially since many runs need to be done to establish reliable timings. As was argued in [7], timings for PVM jobs on multi-user systems such as the one used in this study are best carried out by taking ensemble *minima* of large sets of short runs (few iterations), as opposed to the common engineering practice of using averages. The latter will give a statistical performance figure on an averagely loaded system, while the former gives a good estimate of performance on a dedicated cluster.

In Table 4 we present results for computations on the cluster.

No. processes	time/step (s)	speed-up	efficiency	Mflops
1	51.0	1	1	1.1
4	13.9	3.7	0.92	3.9
9	9.3	5.5	0.61	5.8
16	8.6	5.9	0.37	6.3

Table 4: Performance of MP on cluster using PVM

It was shown in [7] that these figures deteriorate only slightly when the grid is not cubical, even when the largest aspect ratio is more than 2. Clearly, the best speed-up is obtained by going from 1 to 4 processes. Beyond that the performance deteriorates quite rapidly. The main reason for that would appear not to be a limitation of message bandwidth. After all, even for a 16-process case the number of floating point operations within one iteration for one process is about 3,400,000, while the total amount of data transferred per process is just about 460,000 bytes. So the total computation time should be about 3.2 seconds, while the communication time should not be more than 0.8 seconds, for a total of 4.0 seconds per time step. Net latency is around 0.1 second per time step. The remaining 4.5 seconds are spent on idle time when the scheduler is not allocating time for a process, on load imbalance due to such time-outs, and—most importantly—on message contention. All along it was assumed that the PVM message bandwidth is about

0.58 MB/s, but this number was obtained from a message passing experiment between two machines on a dedicated network [3]. So the *aggregate* bandwidth may indeed be reasonably large, but the *effective* bandwidth, namely the bandwidth share for each process, is only a fraction of the total. Should all nodes be connected directly, then there will still be contention in the present cluster, because there are only four physical machines and four gateways for 32 processors. On a better connected network (or one with a significantly higher aggregate bandwidth, such as FDDI) better scalability will be obtained.

Note that scalability on a network of workstations is generally not as good as on massively parallel machines that feature fast communication networks. So even good, coarse-grain algorithms will typically not scale well to hundreds or thousands of workstations. This need not even be necessary in order to employ an extensive cluster usefully. In certain important applications, such as flow solvers on compound grids, several levels of parallelism exist [7]. At the highest level near-trivial parallelism obtains [6], since iteration on a grid can occur completely independently from, and hence in parallel with, iterations on other grids; boundary values get exchanged only after each whole iteration is complete. If more parallelism is sought, individual grids can be divided among several processors, using any of the domain decomposition methods described here. So the number of concurrent processes is the product of the number of grids and the number of processors per grid. That latter number need not be large in order still to obtain large-scale overall parallelism. This is important, because there exist certain inherently unscalable algorithms for small numbers of processors that are potentially more efficient than multi-partitioning, the most important being two-way Gaussian elimination. As was mentioned in section 4.2, if the number of processors dividing a coordinate direction is two, the pencil segment size equals the grid block size and the processors can be active all the time, except during the swap of their interface data. If all three coordinate directions are divided in two, eight processors can be active continuously, which is the maximum for this method.

7 Conclusions

Three strategies have been presented for implementing the implicit solution method defined by the NAS Scalar Penta-diagonal Parallel Benchmark. Of these the multi-partition method has shown the largest range of optimality on the massively parallel Intel Paragon machine. This good performance was achieved because of the combination of coarse granularity and modest communication volume. The multi-partition method was then ported to a network of workstations using PVM, which was an easy exercise; no code was changed, except for message passing syntax and for creation of PVM processes. Reasonable scalability was observed on a small number of processors (up to 16) connected through Ethernet.

References

[1] D. Bailey, J. Barton, Th. Lasinski, H. Simon, *The NAS Parallel Benchmarks*, Report RNR-91-002 Revision 2, Nasa Ames Research Center, Moffett Field, CA, 1991

[2] A. Geist, A. Beguelin, J. Dongarra, W. Jiang, R. Manchek, V. Sunderam, *PVM3 User's guide and reference manual*, Oak Ridge National Laboratory report ORNL/TM-12187, Oak Ridge, TN, 1993

[3] T.G. Mattson, C.C. Douglas, M.H. Schultz, *A comparison of CPS, LINDA, P4, POSYBL, PVM, and TCGMSG: Two node communication times*, YALEU/DCS/TR-975, Yale University, New Haven CT, 1993

[4] N.H. Naik, V.K. Naik, M. Nicoules, *Parallelization of a class of implicit finite difference schemes in computational fluid dynamics*, International Journal of High Speed Computing, Vol. 5, No. 1, pp. 1-50, 1993

[5] T.H. Pulliam, D.S. Chaussee, *A diagonal form of an implicit approximate factorization algorithm*, Journal of Computational Physics, Vol. 29, p. 1037, 1975

[6] M.H. Smith, J.M. Pallis, *MEDUSA - An overset grid flow solver for network-based parallel computer systems*, AIAA Paper 93-3312CP, 24^{th} Fluid Dynamics Conference, Orlando, FL, July 1993

[7] M.H. Smith, R.F. Van der Wijngaart, *Granularity and the parallel efficiency of flow solution on distributed computer systems*, To appear as AIAA Paper 94-2260, 25^{th} Fluid Dynamics Conference, Colorado Springs, CO, June 1994

[8] R.F. Van der Wijngaart, *Efficient implementation of a 3-dimensional ADI method on the iPSC/860*, Supercomputing '93, Portland, OR, November 1993, IEEE Computer Society Press, Los Alamitos, CA

[9] J.C. Yan, P.J. Hontalas, C.E. Fineman, *Instrumentation, performance visualization and debugging tools for multiprocessors*, Proceedings of Technology 2001, vol. 2., pp. 377-385, San Jose, CA, December 1991

8 Appendix

p	n = 64	n = 102
64	444	1396
128	264	787
256	137	454
512	93	285

Table 5: Paragon execution times for PGE (400 steps)

p	n = 64	n = 102
64	274	
102		778
128	180	
204		514
256	161	
306		485
408		508
510		539

Table 6: Paragon execution times for TR (400 steps)

	measured		predicted	
p	n = 64	n = 102	n = 64	n = 102
16	1370		886	
25	747	2496	580	2339
36	467	1793	414	1645
49	343	1290	313	1225
64	293	1012	248	952
81	246	923	204	764
100	196	696	173	630
121	173	598	152	530
144	159	529	137	455
169	151	467	128	396
196	141	419	125	350
256		377		285
361		337		228
400		327		217
441		307		210
484		306		207

Table 7: Paragon execution times for MP (400 steps)