NASA/CR-2001-211238
ICASE Report No. 2001-34

A Parallel Multigrid Solver for Viscous Flows on Anisotropic Structured Grids

Manuel Prieto, Ruben S. Montero, and Ignacio M. Llorente
Universidad Complutense, Madrid, Spain

ICASE
NASA Langley Research Center
Hampton, Virginia

Operated by Universities Space Research Association

National Aeronautics and
Space Administration

Langley Research Center
Hampton, Virginia 23681-2199

Prepared for Langley Research Center
under Contract NAS1-97046

October 2001

A PARALLEL MULTIGRID SOLVER FOR VISCOUS FLOWS ON ANISOTROPIC STRUCTURED GRIDS[*]

MANUEL PRIETO[†], RUBEN S. MONTERO[‡], AND IGNACIO M. LLORENTE[§]

Abstract. This paper presents an efficient parallel multigrid solver for speeding up the computation of a 3-D model that treats the flow of a viscous fluid over a flat plate. The main interest of this simulation lies in exhibiting some basic difficulties that prevent optimal multigrid efficiencies from being achieved. As the computing platform, we have used Coral, a Beowulf-class system based on Intel Pentium processors and equipped with GigaNet cLAN and switched Fast Ethernet networks. Our study not only examines the scalability of the solver but also includes a performance evaluation of Coral where the investigated solver has been used to compare several of its design choices, namely, the interconnection network (GigaNet versus switched Fast-Ethernet) and the node configuration (dual nodes versus single nodes). As a reference, the performance results have been compared with those obtained with the NAS-MG benchmark.

Key words. parallel multigrid, Beowulf clusters, block implicit smoothers, semicoarsening

Subject classification. Computer Science

1. Introduction. Multilevel techniques are generally accepted as fast and efficient methods for solving a wide range of partial differential equations, especially elliptic operators. For these kinds of problems, standard multigrid algorithms based on classical iterative methods, such as Gauss-Seidel or damped Jacobi, exhibit an optimal complexity (the computational work is linearly proportional to the number of unknowns), optimal memory requirements and good parallel efficiencies [12, 2]. These characteristics have made multigrid a common solution method in many application areas, particularly computational fluid dynamics (CFD). In fact, as a result of its popularity, some multigrid solvers such as the NAS-MG (one of the five kernels included in the well-known NAS parallel benchmarks [1]) have also gained widespread acceptance among both the scientific and the computer architecture communities as standard performance indicators.

However, standard multigrid algorithms suffer from a slow-down in convergence in practical CFD applications and the use of more advanced robust techniques is required [2, 13]. One of the most common difficulties that prevent optimal convergence rates from being achieved is the presence of anisotropies. These anisotropies occur naturally in the field of CFD since grid nodes are usually concentrated in certain regions of the computational domain for accuracy reasons or to capture small-scale physical phenomena such as boundary layers. There are two main approaches to dealing with these anisotropic operators. The first approach consists in improving the smoothing process by using an alternating-direction block-implicit smoother [10]. This algorithm explores all the possible directions of coupling of the variables. The second approach

[*]This research was supported by the National Aeronautics and Space Administration under NASA Contract No. NAS1-97046 while the authors were in residence at ICASE, NASA Langley Research Center, Hampton, VA 23681-2199. This research was also supported in part by the Spanish research grant TIC 99-0474 and by the US-Spain Joint Commission for Scientific and Technological Cooperation.

[†]Departamento Arquitectura de Computadores y Automática, Universidad Complutense, 28040 Madrid, Spain. E-mail: mpmatias@dacya.ucm.es

[‡]Departamento Arquitectura de Computadores y Automática, Universidad Complutense, 28040 Madrid, Spain. E-mail: rubensm@dacya.ucm.es

[§]Departamento Arquitectura de Computadores y Automática, Universidad Complutense, 28040 Madrid, Spain. E-mail: llorente@dacya.ucm.es relies on improving the coarse-grid operator. Algorithms like selective coarsening [7], flexible multiple semi-coarsening [30] or block implicit relaxation combined with semicoarsening [6], among others, fall into this category. Although these methods have been successfully applied to fully elliptic equations [24] and the 2-D Navier-Stokes equations [20, 28] their application to the Navier-Stokes equations in 3-D has been limited [21].

The multigrid solver proposed in this research combines a semicoarsening procedure with a plane implicit smoother [14]. To test its robustness we have chosen the simulation of a viscous flow over a yawed flat plate at high Reynolds numbers. Although the flow structure of this problem is relatively simple, it requires a high density of nodes concentrated near the plate surface in order to capture the viscous effects. The numerical properties of this solver have been presented in [15], the parallel implementation of which is the main subject of this paper. To the best of the authors' knowledge, this is the first study of a parallel-plane implicit smoother combined with semicoarsening applied to the Navier-Stokes equations.

As a parallel computing platform we have employed Coral [8], an heterogeneous PC-cluster installed at ICASE, based on Intel Pentium processors and equipped with GigaNet and switched Fast-Ethernet networks. Coral is an ongoing project whose main goal is to evaluate the efficiency of cost-effective Beowulf systems for applications of interest to this center. Among them, we can mention parallel multigrid methods, which have been one of its most important research activities for the last two decades. Given that the computing characteristics of robust algorithms substantially differ from the standard multigrid algorithms included in most benchmark suites (such as the NAS-MG), we think that the proposed solver is also a good application for evaluating Coral's performance.

The rest of this paper is organized as follows. In the second section we describe briefly the characteristics of Coral. The robust multigrid algorithm investigated and the test problem employed are presented in Section 3 and 4 respectively. Section 5 gives some remarks about the most important approaches, in our view, to devising a parallel multigrid solver, focusing our attention on the main complications that arise when block smoothers are applied. Section 6 studies the performance achieved by the investigated solver on Coral and makes a comparison with the NAS-MG benchmark. Finally, the scalability of the proposed algorithm is analyzed in Section 7. The paper ends with some conclusions.

2. Experimental Environment: Coral. The computing platform evaluated in this study, known as Coral [8], is a 96-CPU heterogeneous cluster installed at ICASE, a research institute operated at the NASA Langley Research Center. The original cluster (see Figure 2.1) consisted of a dual CPU front-end server and 32 single compute nodes with 400 MHz Pentium II processors connected via a Fast Ethernet Switch. In a second phase, Coral was upgraded with two file servers and 16 dual nodes, which are equipped with two 500 MHz Pentium III (PIII-500) and linked by another Fast Ethernet Switch. A root Gigabit Ethernet Switch connects the servers and the Fast Ethernet Switches via Gigabit Ethernet uplinks. Currently (Phase 3), Coral has 16 additional compute nodes with two 800 MHz Pentium III processors (PIII-800) per node and a GigaNet cluster area network (cLAN), which connects the 32 dual CPU nodes [4]. GigaNet is a connection-oriented interconnect based on GigaNet's proprietary implementation for ATM switching. Its host interfaces consist in a hardware implementation of the standard Virtual Interface Architecture (VIA), giving user processes direct access to the network interface. In this research we have concentrated on the PIII-800 (Phase 3) *subcluster* in order to assess the impact of using dual CPU nodes and the improvements achieved via GigaNet.

3. Robust Multigrid. The *full multigrid* (FMG) [2] algorithm employed by the robust solver investigated is characterized by a sequence of grids $G = \{\Omega_k : k = 0, 1, 2, ..., N\}$, where Ω_0 is the finest target grid

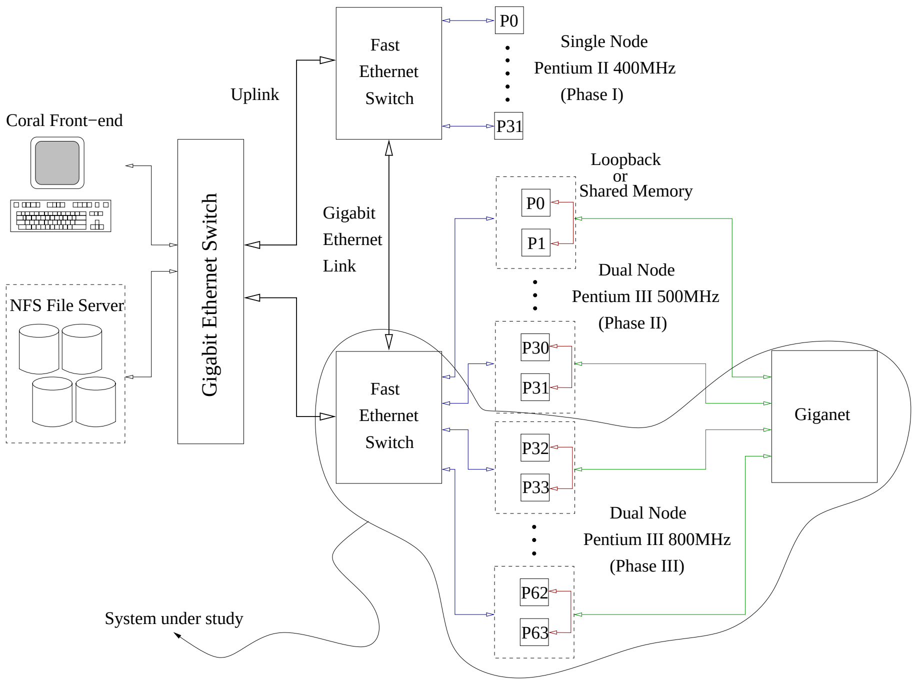


FIG. 2.1. *Coral PC-Cluster at ICASE (Nasa Langley Research Center)*

and the rest of the grids are obtained by applying a semicoarsening procedure, which basically consists in doubling the mesh size in just one direction. The computations are initiated on the coarsest grid and once the discrete system is solved on that level, the solution is transferred to the next finest grid, where it is used as an initial guess. This procedure is repeated until the finest grid is reached.

The algorithm employed a Full Approximation Scheme (FAS) [2] to solve each level in the FMG cycle, which can be recursively defined as in algorithm 1:

Algorithm 1 FAS(ν_1,ν_2,γ,n) multigrid cycle applied to the system $L_n u^n = f^n$ defined on a grid Ω_n. The variables ν_1 and ν_2 denote the number of pre and post-smoothing iterations. The cycle type is fixed with γ.

```
if n=N then
  Apply smoother: $\hat{u}^N = \text{Smooth}(L_N, u^N, f^N, \nu_0)$
else
  Apply smoother: $\hat{u}^n = \text{Smooth}(L_n, u^n, f^n, \nu_1)$
  Evaluation of the residual: $r^n \leftarrow f^n - L_n \hat{u}^n$
  Restriction of the residual: $r^{n+1} \leftarrow I_n^{n+1} r^n$
  Restriction of the solution: $u^{n+1} \leftarrow I_n^{n+1} \hat{u}^n$
  Computation of the right hand-side: $f^{n+1} \leftarrow r^{n+1} + L_{n+1} u^{n+1}$
  for $i = 0$ to $\gamma$ do
    FAS($\nu_1, \nu_2, \gamma, n+1$)
  end for
  Update Solution: $\hat{u}^n \leftarrow \hat{u}^n + I_{n+1}^n (u^{n+1} - \hat{u}^{n+1})$
  Apply smoother: $\hat{u}^n = \text{Smooth}(L_n, \hat{u}^n, f^n, \nu_2)$
end if
```

This multigrid cycle is characterized by the number of pre- and post-smoothing iterations (ν_1, ν_2) and γ, which sets the order in which the grids are visited. Depending on γ the cycle is denoted by $\mathrm{V}(\nu_1, \nu_2)$ if $\gamma = 1$ and by $\mathrm{W}(\nu_1, \nu_2)$ if $\gamma = 2$. In general, a growing γ implies an increasing complexity and more smoothing sweeps in coarser levels with the consequent deterioration of the parallel efficiency [29](see section 5). However low γ cycles (i.e V-cycles) are known to be less robust than W-cycles, especially in convection-dominated problems [22]. Due to this trade-off, the investigated algorithm employed F-cycles, which correspond to a γ between the V and W-cycles, i.e. $1 < \gamma < 2$. Figure 3.1 shows the flowcharts for the V and F-cycles.

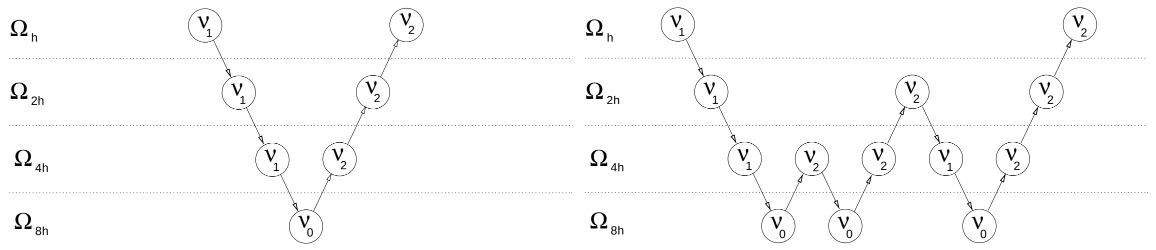


FIG. 3.1. *Scheme of a V-cycle V(ν_1, ν_2) (left-hand chart) and an F-cycle F(ν_1, ν_2) where ν_0 represents the number of iterations of the smoother performed to solve the coarsest level*

The algorithm proposed in this work deals with the anisotropy problem by combining x-semicoarsening (i.e., doubling the mesh space only in the x direction) with a yz-plane implicit solver. We will refer to this method as SCPI (semicoarsening combined with a Symmetric-Coupled Plane-Implicit smother). The planes will be approximately solved by a 2-D multigrid algorithm consisting of one 2-D FAS V(1,1) cycle. Since the same kind of anisotropies found in 3-D problems may appear in the 2-D system a similar 2-D robust multigrid algorithm has been employed based on a line-implicit smoother combined with semi-coarsening. To solve the lines, one 1-D FAS V(1,1) cycle is also applied.

From a computational point of view, block smoothing is obviously more expensive that standard point-wise smoothing. However, we should note that a block smoother can exploit the memory hierarchy more efficiently. The employment of point-wise smoothers, which have to perform global sweeps through data sets that are too large to fit in the cache, often means that multigrid methods only reach a disappointingly small percentage of the theoretically available CPU performance. Some authors have successfully improved cache reuse (locality) using well-known data access and data layout transformations [25, 26, 31]. However, the improvements that can be achieved using these techniques in our algorithm are less relevant since plane smoothers exploit blocking in an implicit way.

4. Flat Plate Boundary Layer Simulation.

As a test problem we have considered the steady-state viscous flow over a cascade of square plates of side L (as depicted in Figure 4.2) with a Reynolds number 10000. In order to obtain the discrete expressions of the equations that govern this problem (the incompressible Navier-Stokes equations), the solution domain is divided into a finite set of hexahedra (control-volumes), where the variables are stored in a staggered way, i.e., the velocities are evaluated on their faces and the pressure field at their centers. Note that in this problem, a plane is understood as a slab of cells as shown in Figure 4.1 (left-hand chart). Hence, the plane smoother will update all velocity components and pressures contained within a slab at the same time (a more detailed description of the plane solver can be found in [15]). Due to the particular dependencies of this problem, the parallel implementation of the smoother has been constructed based on a four-color ordering of planes (right-hand chart in Figure 4.1).

In order to capture the viscous effects, the grids employed in this test are highly stretched near the plate

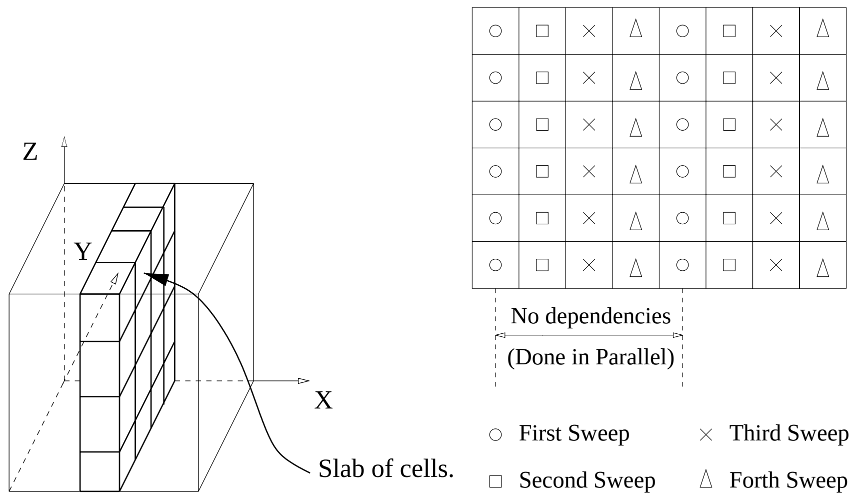


FIG. 4.1. *Slab of cells updated simultaneously when using the yz-plane implicit smoother (left-hand chart). Four-color ordering of planes and data dependencies (right-hand chart).*

(see Figure 4.2). Moreover, the grids are refined near the plate edges to reduce the large discretization errors in these zones [27]. In this work, the experiments have been performed over different geometrically stretched grids of the form $h_{k+1} = \beta h_k$, where h_k is the mesh space of the k^{th} control-volume and the stretching factor β ranges from 1.2 to 1.05 (depending on the grid size).

The number of multigrid levels has been fixed so that the coarsest level has four yz-planes (the coarsest level where it makes sense to apply a four-color plane smoother). Using more levels does not result in any significantly faster convergence and, moreover, it increases the execution time of the parallel version (see Section 5). For the 2-D plane solver, the choice of the number of grid levels is also a trade-off. The optimum has been found empirically in all the experiments reported.

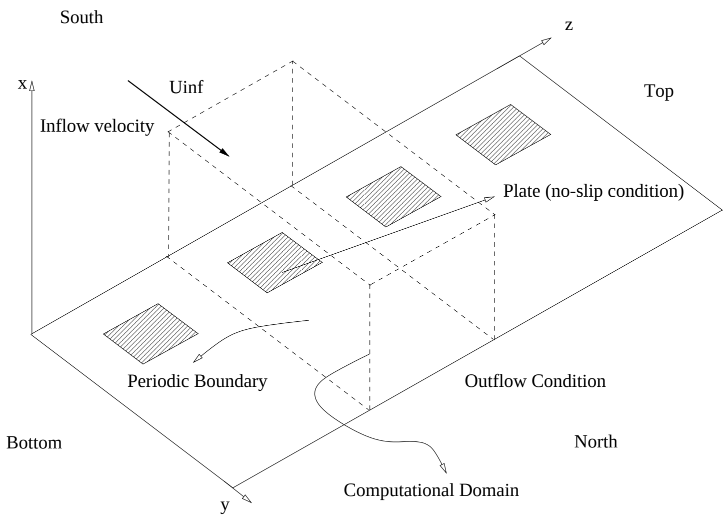
5. Parallel Multigrid.

Generally speaking, there are two different strategies to devise a parallel implementation of a multigrid solver [29]: domain decomposition combined with multigrid (DD-MG) and global multigrid partitioning (GMP or MG-DD). The first approach is based on the general principles of domain decomposition methods. The finest grid is decomposed into a number of blocks, which are then treated with a multigrid method as independently as possible. The main advantage of this scheme lies in its straightforward application to general multi-block and irregular grids. However, it requires a careful treatment of the connections between the different blocks in order to achieve satisfactory convergence rates, which often involves domain overlapping.

The second technique consists in applying domain decomposition on every grid level, not only on the finest grid. In this way, for many classical multigrid algorithms, all parallel approaches based on GMP are algorithmically equivalent to their non-partitioned versions. Nevertheless, the algorithmical equivalence may not be easily achieved for more complicated applications where block-implicit smoothers are required. In addition, unlike DD-MG approaches, the degree of parallelism changes from one multigrid level to the next and the communication-to-computation ratio may become unsatisfactory on coarse grids. Indeed, on very coarse levels, some (or many) of the processors may be idle.

5.1. Parallel Block Smoothers.

The parallel efficiency that can be achieved by means of the GMP approach depends on the characteristics of the different multigrid components. Common grid transfer operators (I_n^{n+1}, I_{n+1}^n) or residual evaluations (r^n) do not need any further discussion since these components are



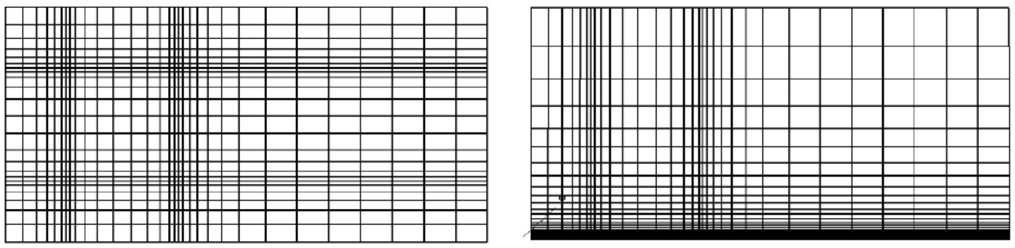


FIG. 4.2. *Schematic configuration of the numerical simulation (top chart). Computational grid used, yz-plane (Bottom-left chart) and xy-plane (Bottom-right chart).*

embarrassingly parallel by nature and consequently, their parallel counterparts do not impose any significant overheads on the execution time (as shown in algorithm 2, these operators only require the usual exchange of halos). Unfortunately, this is not the case for block smoothers. Indeed, this component can be difficult or even impossible to parallelize.

Focusing our attention on the plane solver employed by the SCPI algorithm, it is always possible to avoid the need for a parallel version by using a 1-D data decomposition in the semicoarsening direction, as Figure 5.1 shows. From an implementational point of view, this is by far the best scheme that can be considered, since it avoids the programming effort and the overheads that a parallel plane solver introduces into the code. These considerations have been employed for example in [24] and [11] to parallelize a robust multigrid algorithm for the anisotropic diffusion and advection equations respectively. Nevertheless, although 1-D decompositions have no need for a parallel plane smoother, they also have some drawbacks. The most important one, which can be denoted as the critical level problem [11], is discussed below.

5.2. Critical Level Problem. The need to solve *exactly* the system of equations on the coarsest grid [2] usually leads to choosing the coarsest multigrid level as coarse as possible to reduce the computational cost. However, in the parallel implementation, this decision may cause some processors to be idle on the coarsest grids. To clear up this problem it is convenient to define the multigrid critical level as the level L where the following condition is satisfied:

$$\left(\frac{N_x(L)}{S\cdot P_x} \vee \frac{N_y(L)}{S\cdot P_y} \vee \frac{N_z(L)}{S\cdot P_z}\right) = 1, \text{ with } S = \begin{cases} 1, & \text{Damped Jacobi} \\ 2, & \text{Zebra Gauss–Seidel} \\ 4, & \text{Four–Color Gauss–Seidel} \end{cases} \tag{5.1}$$

Algorithm 2 Parallel FAS(ν_1,ν_2,γ,n) multigrid cycle applied to the system $L_n u^n = f^n$ defined on a grid Ω_n. The variables ν_1 and ν_2 denote the number of pre and post-smoothing iterations. The cycle type is fixed with γ.

```
if n=N then
  Apply parallel smoother: $\hat{u}^N$ = Parallel_Smooth($L_N, u^N, f^N, \nu_0$)
else
  Apply parallel smoother: $\hat{u}^n$ = Parallel_Smooth($L_n, u^n, f^n, \nu_1$)
  Exchange solution halos $u^n$ in grid $\Omega_n$
  Evaluation of the residual: $r^n \leftarrow f^n - L_n \hat{u}^n$
  Exchange residual halos $r^n$ in grid $\Omega_n$
  Restriction of the residual: $r^{n+1} \leftarrow I_n^{n+1} r^n$
  Restriction of the solution: $u^{n+1} \leftarrow I_n^{n+1} \hat{u}^n$
  Exchange solution and residual halos in grid $\Omega_{n+1}$
  Computation of the right hand-side: $f^{n+1} \leftarrow r^{n+1} + L_{n+1} u^{n+1}$
  for $i = 0$ to $\gamma$ do
    FAS($\nu_1, \nu_2, \gamma, n+1$)
  end for
  Exchange solution halos $u^{n+1}$ in grid $\Omega_{n+1}$
  Update Solution: $\hat{u}^n \leftarrow \hat{u}^n + I_{n+1}^n (u^{n+1} - \hat{u}^{n+1})$
  Apply smoother: $\hat{u}^n$ = Parallel_Smooth($L_n, \hat{u}^n, f^n, \nu_2$)
end if
```

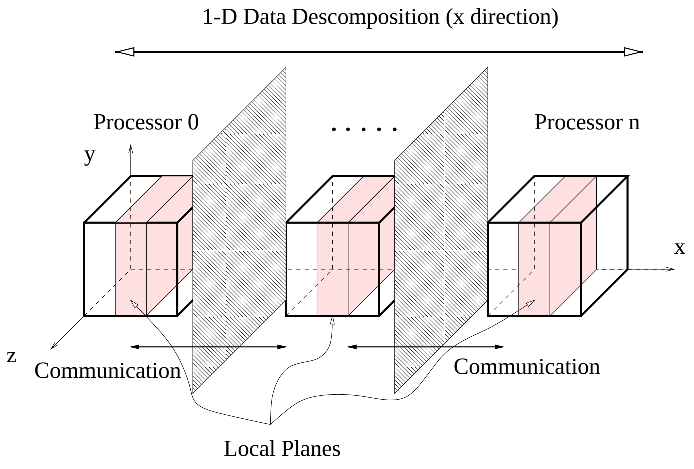


FIG. 5.1. *Schematic data distribution of a 1-D data decomposition*

where $N_x(L)$, $N_y(L)$, $N_z(L)$ are the local number of cells per side on level L in direction x, y and z respectively, and P_x, P_y and P_z are the number of processors in direction x, y and z. That is, the critical level is the coarsest level at which all processors can perform the smoothing operation concurrently or, or in other words, the multigrid level where each processor has one local plane in the case of a damped Jacobi smoother, two planes for a zebra update and four planes in the case of a four-color ordering.

Below the critical level, the parallel algorithm has serious load-balance problems that reduce its efficiency, since the number of idle processors is doubled on every level below the critical one. It also complicates its implementation because, as we pass below the critical level, it may be necessary to dynamically rearrange the communication patterns and grid distributions. Among the most popular alternatives that could alleviate this problem, we should mention:

- *Agglomeration on coarsest grids.* In some cases, the idleness of processors is not the main overhead source and multigrid may be faster just using one processor on the very coarse grids (below the critical level). It makes sense to apply this approach, which for example has been successfully employed in [16], when the communication overhead on coarse levels is more problematic than the load-imbalance, i.e. when the communications are more expensive than the computation. However, when plane-wise smoothers are considered, the communication-computation ratio is still low even on the coarsest grids and this approach fails to achieve satisfactory efficiencies. In other words, this strategy is more suitable in the context of point-wise relaxation because plane-wise smoothers already have an implicit degree of agglomeration.
- *Multiple Coarse Grids.* This approach keeps the processors busy below the critical level using multiple coarse grids. Although it slightly increases the execution time, since extra work is necessary to merge the solutions of the different grids, it may improve the convergence properties of the method. However, it is quite difficult to find satisfactory merging operators for sophisticated problems.
- *U-Cycle method.* In some cases, it is advisable to avoid idle processors by fixing the number of grid levels so that the coarsest grid employed is the critical level. This strategy makes the implementation easier and keeps all the processors busy. However, not going down to the coarsest possible grids changes the algorithm, since in the sequential counterpart the coarsest level is usually chosen to be as coarse as possible. For many applications where a large number of grid levels is still processed, this strategy has achieved very satisfactory results. If an iterative method is employed to solve the system of equations on the coarsest grid, the efficiency of the U-Cycle depends on the required number of iterations, which in turn depends on the number of processors since it grows with the system size [24].

5.3. U-cycle. In this research we have tried to steer clear of load-imbalance problems by using the U-Cycle approach. However, if this strategy were combined with a 1-D decomposition to avoid a parallel plane smoother, the scalability of the corresponding solver would be very limited due to a high critical level. This fact is of a great relevance in the SCPI algorithm, since it employs a four-color ordering for updating the planes.

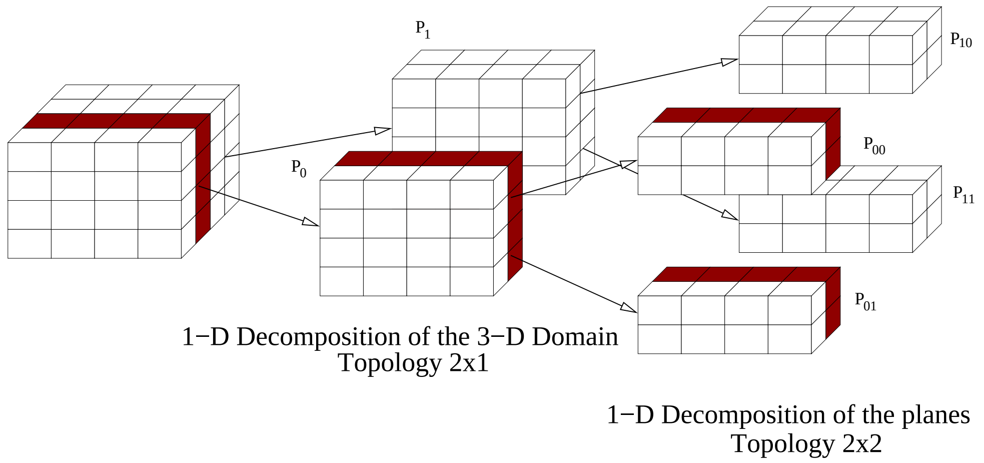
Let us assume, without losing generality, that the avaliable processors (P) are equally distributed among all the partitioned directions. If we define d as the number of the dimensions in which the data is partitioned, the critical level L satisfies the following condition:

$$\frac{\min(N_i(L))}{S \cdot P^{1/d}} = 1 \quad i \in (x, y, z) \;\; / \;\; i = \text{partitioned direction}. \tag{5.2}$$

As shown in equation 5.2 the critical level can be lowered by either reducing the coloring of the smoother or using a higher order decomposition, both alternatives have been combined in the parallel implementation of the solver.

We have opted to employ a 2-D decomposition since 3-D data decompositions require a parallel tridiagonal solver. Although these kinds of solvers have been widely studied (see for example [9]) and it is possible to obtain quite satisfactory efficiencies for large and even moderate problem sizes, current memory limitations make it impossible to solve 3-D problems whose corresponding lines are big enough to obtain reasonable efficiencies [24]. Obviously, a 2-D decomposition introduces some overhead to the plane solver. Indeed, since it consists in a 2-D version of the SCPI algorithm, it presents the same complications as those discussed above for the 3-D problem. However, the critical level problem is less troublesome in the 2-D counterpart because the computational cost required to solve the coarsest levels is much lower than in 3-D.

As Figure 5.2 shows, such a decomposition can also be seen as a 1-D partitioning of the plane solver integrated into the 1-D decomposition of a 3-D domain. Although it is beyond the scope of this research, this view suggests that an hybrid message-passing (a 1-D distribution of the 3-D domain using MPI) and shared-memory (the plane solver is parallelized using OpenMP) parallelism could take advantage of modern HPC machines based on clusters of shared-memory compute nodes.



P_1
P_{10}
P_0
P_{00}
P_{11}
P_{01}
1−D Decomposition of the 3−D Domain
Topology 2x1
1−D Decomposition of the planes
Topology 2x2

FIG. 5.2. *Schematic data distribution of a 2-D data decomposition*

In addition, the plane smoother employed in our parallel SCPI algorithm reduces its coloring S dynamically, using a zebra and damped Jacobi updates on grids below the critical level. Obviously, this new smoother causes the numerical properties of the algorithm to deteriorate. Nevertheless, increasing the number of iterations when zebra or Jacobi updates are used compensates for their impact on the convergence rate. In this way, the parallel SCPI algorithm achieves the same convergence rate as that of the sequential counterpart.

Finally, we should remark that given a certain problem size and a certain number of processors P, the choice of the optimum 2-D process topology is a tradeoff between non-parallelization of the plane solver (following the notation introduced in Figure 5.2, a topology $Px1$), and a topology $1xP$, where all the processors cooperate in solving each plane. The former is at the expense of a change in the algorithm (not going down to the 3-D coarsest possible grid) while the latter is at the expense of some communication overhead in the plane solver. The experimental results presented in the next section have been obtained using the optimal topology.

6. Performance of the SCPI solver on Coral. In this section we have studied in more detail the performance of the SCPI solver on Coral. The results have been compared with the well-known NAS-MG benchmark (class B)[1], a standard multigrid V-cycle (see algorithm 3) based on global multigrid partitioning that solves the Poisson equation on a 3-D uniform grid (class B uses a 256^3 grid). This comparison can only be seen as a reference, whose main goal is to highlight how an optimal cluster design strongly depends on the target applications, even for such a specific area as that of parallel multigrid methods based on global multigrid partitioning.

Algorithm 3 NAS(n) V-cycle multigrid cycle applied to the system $L_n u^n = f^n$ defined on a grid Ω_n.

```
if n=N then
    Apply Smoother: $u^N$ = Smooth($r_N$, 0)
else
    Restriction of the residual: $r^{n+1} \leftarrow I_n^{n+1} r^n$
    Recursively solve the system on $\Omega_{n+1}$: NAS($n+1$)
    Prolongate solution: $u^n \leftarrow I_{h+1}^h u^{n+1}$
    Evaluate residual: $r^n \leftarrow f^n - L_n u^n$
    Apply Smoother: $u^n$ = Smooth($r_n$, $u^n$)
end if
```

6.1. Analysis of the Interconnection Network. The interconnection network is probably the key factor in the design of a Beowulf-class system. Its overall cost, as well as the potential efficiency that can be achieved, strongly depends on its choice. Possibilities range from a low cost Fast-Ethernet switch to a state-of-the-art cluster area network interconnect, such as Myrinet [19] or GigaNet [4]. In this section, we will evaluate the effect of the two different interconnection networks available in Coral on the performance of the two multigrid solvers considered. To take advantage of the GigaNet network, the codes have been compiled against the MPI/Pro library [18], a commercial implementation of the MPI standard which offers access to the VIA interface in an interrupt-driven receive mode.

6.1.1. Raw Performance. Before discussing the network impact on the multigrid solvers investigated, it is worthwhile comparing the raw performance achieved by MPI/Pro in both networks, particularly the point-to-point communication performance, since this operation accounts for the greater part of the communication cost in both codes (collective communications are also required to compute vector norms but their overheads are insignificant in both cases). As a point-to-point benchmark we have employed the classical ping-pong test between two processes running on different nodes [23]. In this basic test GigaNet clearly outperforms Fast-Ethernet. MPI/Pro over GigaNet achieves an asymptotic bandwidth of about 102 Mbytes/s, which is about nine times better than the Fast Ethernet bandwidth (around 11.1 Mbytes/s) [8].

Nevertheless, this basic test ignores the effect of message memory layout on message-passing performance, since it assumes that the data to be communicated are contiguously stored in memory. However, this is not always the case in practical applications since boundary data are not, in general, contiguous in memory.

A quantitative measurement of the effect of this characteristic can be obtained using a modified ping-pong test, where message spatial locality is modified by means of different strides between successive elements of the message (see [23] for a detailed discussion). As Figure 6.1 shows, non-unit-stride memory accesses have a severe impact on performance (following the notation of the MPI_Type_vector data type [17], stride-one represents contiguous data). This fact is especially relevant for GigaNet, where the effective bandwidth is reduced from a peak of about 102 Mbytes/s to about 12 Mbytes/s for stride-four messages, a performance drop of around 88%. The equivalent drop over Fast-Ethernet is also very significant but only about 42% (from about 11.1 Mbytes/s to about 6.5 Mbytes/s).

6.1.2. SCPI Performance. Figure 6.2 shows the efficiency obtained by the SCPI solver for a fixed 32x128x128 problem size. As usual, the efficiency has been defined as:

$$E(N,P) = \frac{T(N,1)}{P \times T(N,P)}, \tag{6.1}$$

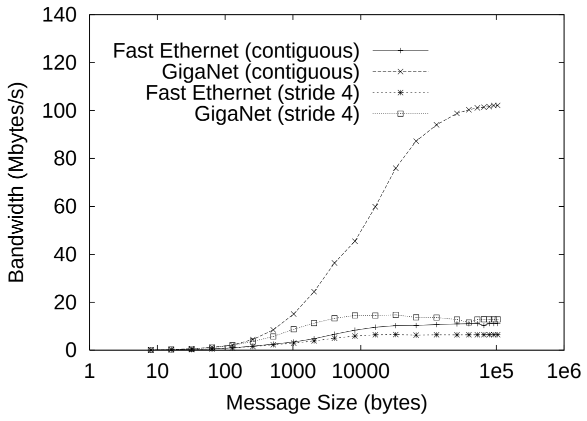


FIG. 6.1. *Point-to-point communication bandwidth obtained by MPI/Pro over GigaNet and over Fast-Ethernet in Coral. The measurements have been obtained using the classical ping-pong test with two different message memory layouts: contiguous and stride-four vectors.*

where the execution time chosen is the time needed to perform one cycle of the SCPI solver on every grid level. These measurements have been performed over GigaNet and over Fast Ethernet under an unloaded network using two MPI processes per node.

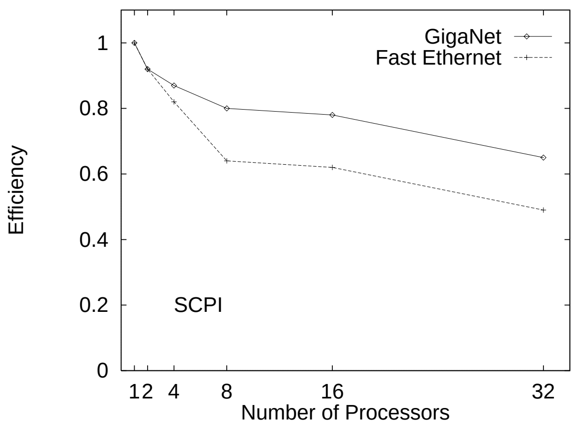


FIG. 6.2. *Parallel efficiency obtained by the SCPI solver for a fixed problem size using a dual node configuration in combination with the GigaNet or the Fast-Ethernet networks.*

As could be expected, GigaNet outperforms Fast Ethernet, especially as the number of processors grows. The improvement achieved by GigaNet varies from a small margin of 7%, using 4 processors, to 30% for 32 processors. The drop in efficiency experienced by the code when moving from the GigaNet to Fast-Ethernet is due to the worsening communication-to-computation ratio. Given that the problem size is fixed, the difference in this ratio for the two interconnects grows linearly with the number of processors (see right-hand chart in Figure 6.3).

The great differences in performance experienced with both networks are due to the efficient exploitation of the interconnection hardware made by the SCPI algorithm. In order to analyze this difference it is useful to consider the GigaNet communication and computation gains. The communication gain has been defined as the ratio of the communication cost over Fast-Ethernet to the GigaNet counterpart (and similarly for the computation gain):

$$G = \frac{T_{eth}}{T_{gnet}}. \tag{6.2}$$

The left-hand chart in Figure 6.3 shows the gains in communication and computation achieved by GigaNet. As expected the computation time is the same for both networks (i.e. $G_{computation} \approx 1$), whereas the communication gain is around 2, which is lower than the gains obtained in the ping-pong test for the problem sizes involved in the simulation (note that these measurements also involve intra-node communications).

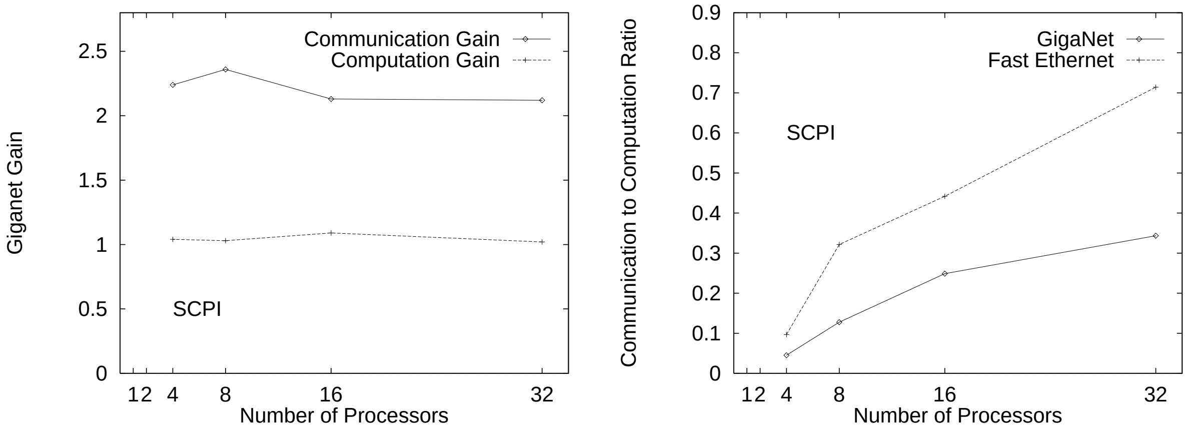


FIG. 6.3. *Computation and communications gains achieved by GigaNet (left-hand chart) and communication to computation ratio for the Fast Ethernet and the GigaNet networks (right-hand chart). These experimental results have been obtained for the SCPI solver (using a fixed 32x128x128 problem size) and a dual node configuration.*

6.1.3. NAS-MG Performance. Using the NAS-MG benchmark, the efficiency obtained is again better with GigaNet (Figure 6.4). However, in this case the difference between both networks is not as remarkable as for the SCPI solver. This is due to the lower communication gain obtained by GigaNet in this case, which is a consequence of the poor spatial locality in some of the boundaries employed by the NAS-MG.

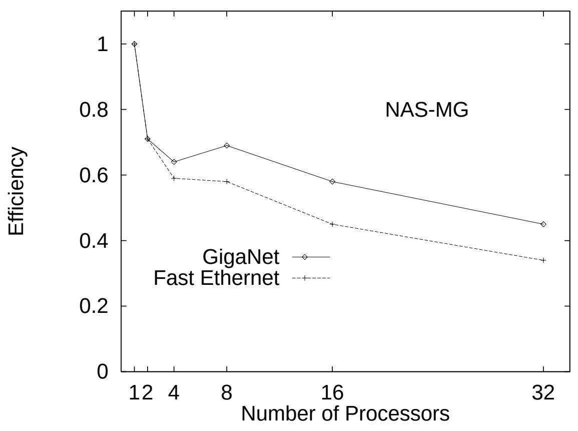


FIG. 6.4. *Parallel efficiency obtained by the NAS-MG (class B) benchmarks using a dual node configuration in combination with the GigaNet or the Fast-Ethernet networks.*

The 2-D data decomposition employed in the SCPI algorithm has been deliberately chosen so that boundaries are stored almost contiguously in memory. However, the NAS-MG benchmark uses a 3-D data decomposition, forcing the usage of non-contiguous boundaries. Indeed, point-to point communications are done in the NAS-MG benchmark via an explicit packing of data, i.e. messages are first built by transferring data from the original boundaries into a message buffer explicitly managed by the program. In this case, we can assume that the communication cost can be split into:

$$T = t_{net} + \tau, \tag{6.3}$$

where only the term t_{net} depends on the interconnection network. The τ parameter, which accounts for the explicit message packing, is network independent and, consequently, it limits the potential communication improvement that can be achieved by the interconnection network. Applying Amdahl's law (i.e. assuming an ideal network where t_{net} is insignificant), the maximum communication gain that can be achieved is:

$$G_{max} = \frac{T_{target\_net}}{T_{ideal\_net}} = 1 + \frac{t_{net}}{\tau}. \tag{6.4}$$

Combining equation (6.2) and (6.3), the communication gain achieved by GigaNet can be predicted by:

$$G_{pre} = \frac{1 + \frac{\tau}{t_{eth}}}{\frac{t_{gnet}}{t_{eth}} + \frac{\tau}{t_{eth}}}. \tag{6.5}$$

In the left-hand chart in Figure 6.5 the experimental communication gain is plotted against the maximum gain (G_{max}) and the predicted gain (G_{pre}). The quotient t_{gnet}/t_{eth} in equation (6.5) has been obtained using data from the ping-pong test and τ has been experimentally measured. The experimental gain, which matches the predicted one, is about 1.6 for more than 8 processors, which is only 46% lower than the maximum gain. Compared to the SCPI algorithm, this gain is around 20% lower.

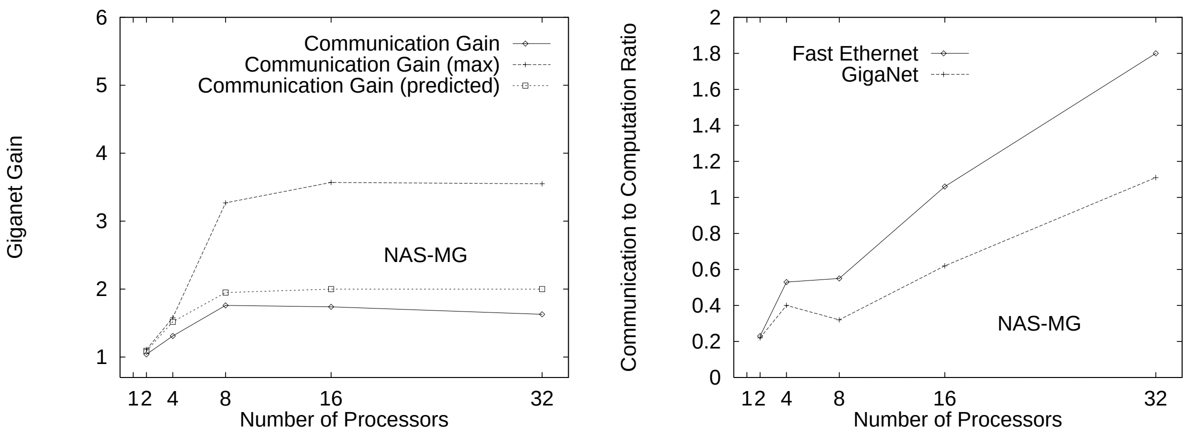


FIG. 6.5. *Computation and communications gains achieved by GigaNet (left-hand chart) and communication to computation ratio for the Fast Ethernet and the GigaNet networks (right-hand chart). These experimental results have been obtained for the NAS-MG (Class B) benchmark using a dual node configuration.*

In the right-hand chart in Figure 6.5 the communication to computation ratio of the NAS-MG benchmark is shown for the two networks. As for the SCPI, the difference between the two ratios grows linearly with the number of processors, thus increasing the efficiency of GigaNet over Fast-Ethernet as the number of processors is increased. However, when compared with the SCPI, the communication-to-computation ratio of the NAS-MG is substantially higher. For example, for the 32 processors simulation the ratio is around 0.3 for the SCPI, while for the NAS-MG it is around 0.9. This result is due to both the higher computation count of the implicit plane solver and the locality of the messages exhibited by the SCPI. This fact is clearly reflected in the efficiency results of Figure 6.2, note that using 32 processors the efficiency of the SCPI is 0.7, which is 40% better than that obtained by the NAS-MG.

6.2. Analysis of the Dual Node configuration. In recent years, dual node configurations have become a standard in cluster computing. The drop in system cost and power consumption, the reduction in space and wiring-complexity and the attraction of the possible use of shared memory paradigm have been, among others, the main reasons leading to this fact. Focusing on system cost and quoting July 2001 prices obtained from Compaq [5] and Myricom [19] sites, a 16 node cluster equipped with Compaq ProLiant DL320 single nodes (with Intel Pentium III processors running at 1GHz) and a 16 serial-port Myrinet switch (with the corresponding host interface cards) is about 27% more expensive than a similar 8 node cluster equipped with Compaq ProLiant DL360 dual nodes (with two Intel Pentium III processors running at 1GHz and twice the amount of memory and disk space than the DL320 nodes) and a 8 serial-port Myrinet switch. In addition, this difference grows with the number of nodes, since network cost does not scale linearly with system size (making the comparison between a 64 dual-node cluster and its 128 single-node counterpart, the difference grows to 35%).

However, single node configurations can obtain a better performance compared to their dual counterparts, and consequently the right choice (dual versus single node configuration) strongly depends on the cluster target application. In this section, we have assessed dual and single node configurations taking the NAS-MG and the SCPI solver as targets.

Before studying both solvers, we should remark that although one of the advantages of dual computing is the potential reduction in the intra-node communications cost, the current version of MPI/Pro installed on Coral does not seem to take advantage of the shared memory. Indeed, the asymptotic intra-node bandwidth using MPI/Pro is only 83 Mbytes/s. Better performance is obtained in this case using MPI Lam (using the correct Lam driver [8]). The peak bandwidth is about 270 Mbytes/s for message sizes lower than 256 Kbytes (for longer messages it drops to 127 Mbytes/s since the internal message buffers do not fit into the L2 cache)[8].

6.2.1. SCPI Performance. As shown in the left-hand chart in Figure 6.6 , the efficiency obtained by the SCPI solver (for a fixed 32x128x128 problem size) using single nodes combined over GigaNet is almost optimal up to four processors and remains satisfactory for eight or more processors.

The right-hand chart in Figure 6.6 shows the communication and computation overheads introduced by the dual configuration due to the competition for shared resources (local memory and network card), where the overhead has been defined as:

$$O = \frac{t_{dual} - t_{single}}{t_{dual}}. \tag{6.6}$$

We should point out that when the single nodes are replaced by dual nodes the computing time is increased by only 15%. This low increase is due to the locality exhibited by the implicit plane solver, which reduces memory traffic, and hence relieves the memory contention. Thus, the SCPI algorithm does not present a significant reduction (15% to 20%) in efficiency when dual nodes are used.

6.2.2. NAS-MG Performance. As the left-hand chart in Figure 6.7 shows, dual configuration causes an important efficiency reduction on the NAS-MG. As shown in Figure 6.7 (right-hand chart), the computation overhead grows to about 35% (twice than in the SCPI solver). The communication overhead does not grow with the number of processors, although 70% is the overhead achieved on the SCPI for the 16-processor

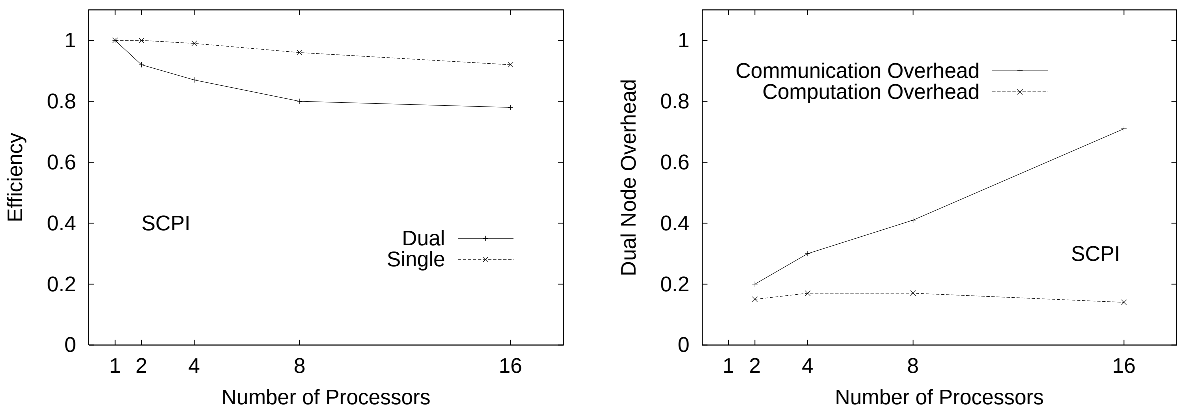


Fig. 6.6. *Efficiency for a fixed 32x128x128 problem size (left-hand chart) and dual node communication and computation overheads achieved for the SCPI solver (right-hand chart). These measurements have been obtained over GigaNet.*

case. The computation overhead increase is due to the poor locality exhibited by the NAS-MG benchmark, which magnifies the competition for the memory system.

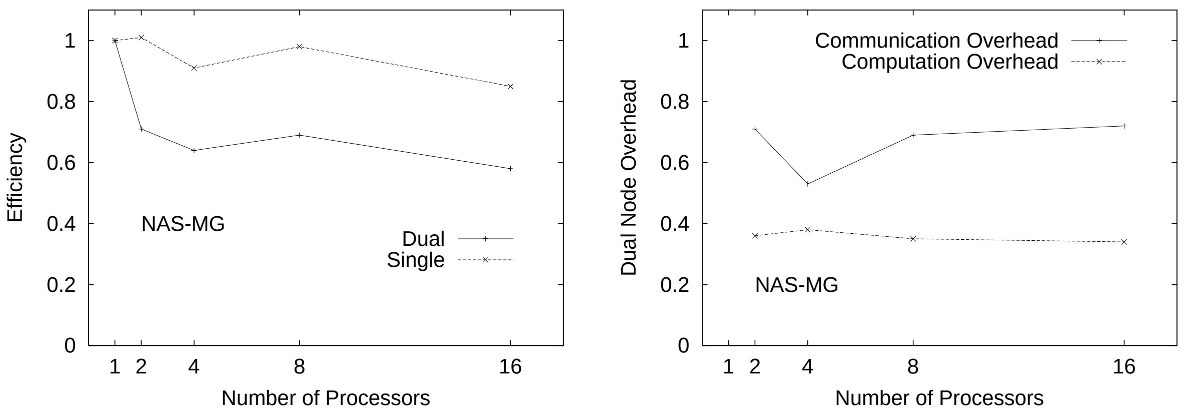


Fig. 6.7. *Dual node communication and computation overheads achieved for the NAS-MG benchmark (left-hand chart) and communication-to-computation ratio for the dual and single node configurations (right-hand chart). These measurements have been obtained over GigaNet.*

7. SCPI Scalability. Before discussing the SCPI scalability, we should remark that for this kind of solvers scalability involves two different aspects, which can be denoted as *algorithmic scalability* and *implementation scalability* [3]. From a numerical point of view, scalability (*algorithmic scalability*) requires that the computational work per iteration only grows linearly with the problem size and that the convergence factor per iteration remains bounded below 1, the bound being independent of problem size. The second aspect (*implementation scalability*) only requires that a single solver iteration is scalable on the target computing platform.

Although the aim of this paper is to study the implementation scalability, Figure (7.1) shows the convergence histories achieved by the SCPI algorithm for the target flat-plate simulation. The residual norm is reduced by nearly five orders of magnitude in the first five cycles on the finer grid (corresponding to a convergence rate of roughly 0.1 per fine grid iteration), which is close to that obtained for the Poisson equation with a semi-coarsened smoother [24]. In addition, the convergence rate is independent of the grid size and the grid-stretching factor. More numerical results can be found in [15].

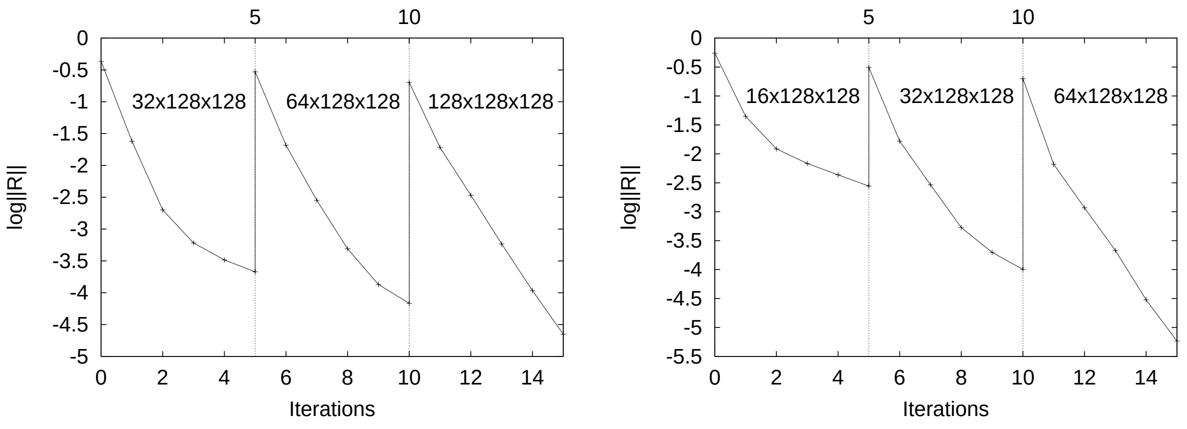


FIG. 7.1. $\mathbf{L_2}$*-norm of the residual versus F(2,1)-cycles of the investigated SCPI algorithm for the flat plate simulation with* $\mathbf{Re} = \mathbf{10^4}$ *on a 128x128x128 grid (left-hand chart), and a 64x128x128 (right-hand chart).*

Focusing on the *implementation scalability*, as mention above, if a fixed problem size (time-critical scaling model) is considered the performance of the SCPI algorithm is almost optimal up to four processors (for the single processor configuration), since in these simulations it is possible to apply a 1-D decomposition and consequently the plane solver, which is by far the most time consuming component of the algorithm, does not suffer any communication overhead. Due to the critical level problem, experiments using eight or more processors require a 2-D decomposition and the efficiency decreases, although it remains satisfactory up to 32-processor simulations.

In practice, the usage of a large number of processors only makes sense for large problem sizes. Hence, although the efficiency data discussed above provide useful information about the *implementation scalability* of the SCPI solver, it is more relevant to study how the algorithm scales when both the size of the problem and the number of processors are increased (accuracy-critical scaling model) [12]. As is well known, in this case it is not possible to study the scalability taking the efficiency as a reference, since it is not possible to obtain the sequential solution of larger problems due to memory constraints.

In this research we have opted to use a *scaled* efficiency:

$$E(N,P) = \frac{T(N,1)}{T(PN,P)}. \tag{7.1}$$

One would like a highly scalable algorithm where $E(N,P) = 1$, i.e. one would like that if the problem size is doubled, doubling the number of processors would keep the solution time constant. Nevertheless, as other authors have pointed out, a solver can be considered nearly scalable if its *scaled* efficiency remains bounded away from zero, i.e. $E(N,P) > 0$.

The *scaled* efficiency data shown in Figure (7.2) has been taken over GigaNet using two MPI processes per node. As expected, the efficiency becomes worse for increasing P, but we can say that our algorithm nearly scales since its *scaled* efficiency only decreases logarithmically and remains bounded away from zero. Indeed, this is the most reasonable scalability that multigrid algorithms can achieve since as the number of processors and the problem size get larger (N/P = const.), the number of levels also increases [29].

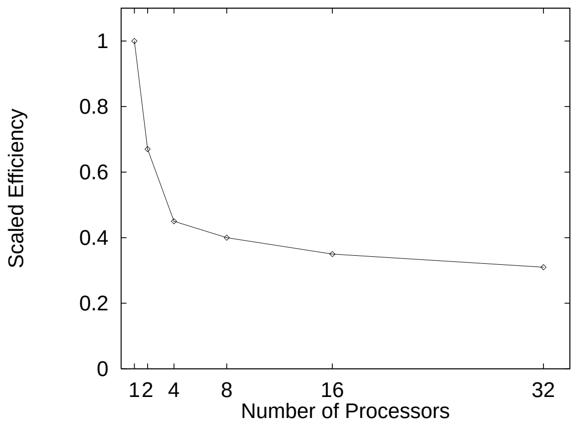


FIG. 7.2. *Scalability of the SCPI algorithm if both the number of processors and the problem size grow. The measurements have been taken using MPI/Pro over GigaNet using two MPI process per node.*

8. Conclusions.

The combination of semicoarsening and a plane-implicit smoother has been studied in the simulation of a flat plate boundary layer, taking into account both numerical and architectural properties. The main conclusions can be summarized as follows:

- The solver reduces the residual norm by nearly five orders of magnitude in the first five cycles on the finest grid in all cases. In addition, the algorithm is fully robust: the convergence rate is independent of the grid size and the grid-stretching factor [15].
- The strategy considered for parallelizing the SCPI solver consists in applying a multigrid U-cycle with a 2-D grid partitioning. Unlike 3-D decompositions, this strategy avoids the need for a parallel block tridiagonal solver that has been previously reported to have low efficiency for small problems. A 1-D grid decomposition was also found to be non-scalable due to the critical level problem.
- Satisfactory efficiencies have been obtained for up to 32 processors and the *scaled* efficiency remains bounded away from zero.

In addition, we have analyzed the different Coral configurations using both the solver investigated and the NAS-MG benchmark. The results highlight the strong dependence of the optimal configuration choice on the target applications, even for such a specific area as that of parallel multigrid methods. For the NAS-MG kernel, the most convenient configuration (taking performance and cost factors into account) seems to be the combination of a switched Fast-Ethernet network with single nodes. GigaNet only achieves a gain of about 1.6 over Fast-Ethernet since the message-packing cost, which is network independent, accounts for an important percentage of the communication cost in this application. On the other hand, a dual configuration imposes a high overhead of about 30% due to poor data locality exploitation, which increases memory traffic. However for the SCPI code, which represents a better characterization of a practical multigrid workload, GigaNet achieves a significant improvement of about 2.15 in communication time and the dual node overhead is only about 15% due to a better exploitation of data locality.

9. Acknowledgment. The authors would like to thank ICASE for providing access to the parallel computer that has been used in this research. We would also like to particularly acknowledge Manuel D. Salas for his constructive comments on the CFD discipline and the simulation of boundary layers, and Josip Loncaric for his valuable assistance with Coral.

REFERENCES

[1] D. H. Bailey, T. Harris, R. V. der Wigngaart, W. Saphir, A. Woo, and M. Yarrow, *The NAS parallel benchmarks 2.0*, Tech. Rep. NAS-95-010, NASA Ames Research Center, 1995.

[2] A. Brandt, *Multigrid techniques: 1984 guide with applications to fluid dynamics*, Tech. Rep. GMD-Studien 85, May 1984.

[3] P. N. Brown, R. D. Falgout, and J. E. Jones, *Semicoarsening multigrid on distributed memory machines*, SIAM J. Sci. Comput,, 21(5) (2000), pp. 1823–1834.

[4] cLAM, *Emulex Inc.* http://wwwip.emulex.com/ip/index.html.

[5] Compaq, *Proliant family.* http://www.compaq.com/products/servers/.

[6] J. E. Dendy, S. F. McCormick, J. Ruge, T. Russell, and S. Schaffer, *Multigrid methods for three-dimensional petroleum reservoir simulation*, in Tenth SPE Symposium on Reservoir Simulation, February 1989.

[7] B. Diskin, *Multigrid algorithm with conditional coarsening for the nonaligned sonic flow*, Electronic Trans. Num. An., 6, pp. 106–119. Proceedings of *the Eighth Copper Mountain Conference on Multigrid Methods*, 1997.

[8] ICASE, *The Coral Project.* http://www.icase.edu/CoralProject.html.

[9] J. López, O. Platas, F. Argüello, and E. L. Zapata, *Unified framework for the parallelization of divide and conquer based tridiagonal systems*, Parallel Computing, 23 (1997), pp. 667–686.

[10] I. M. Llorente and N. D. Melson, *Behavior of plane relaxation methods as multigrid smoothers*, Electronic Transactions on Numerical Analysis, 10 (2000), pp. 92–114.

[11] I. M. Llorente, M. Prieto, and B. Diskin, *An efficient parallel mutigrid solver for 3-d convection-dominated problems*, Tech. Rep. 00-29, ICASE, 2000 (Submitted to Parallel Computing).

[12] I. M. Llorente and F. Tirado, *Relationships between efficiency and execution time of full multigrid methods on parallel computers*, IEEE Trans. on Parallel and Distributed Systems, 8 (1997), pp. 562–573.

[13] O. A. McBryan, P. O. Frederickson, J. Linden, A. Schuller, K. Solchenbach, K. Stuben, C. Thole, and U.Trottenberg, *Multigrid methods on parallel computers-a survey of recent developments*, Impact of Computing in Science and Engineering, 3 (1991), pp. 1–75.

[14] R. S. Montero and I. M. Llorente, *Robust multigrid algorithms for the incompresible Navier-Stokes equations*, Tech. Rep. 00-27, ICASE, 2000.

[15] R. S. Montero, I. M. Llorente, and M. D. Salas, *Semicoarsening and Implicit Smoothers for the Simulation of a Flat Plate at Yaw*, Tech. Rep. 2001-13, ICASE, 2000.

[16] D. Moulton and J. Dendy, *MPI-based black box multigrid for workstation clusters*, in Proceedings of the 9th Copper Mountain Conference on Multigrid Methods, 1999.

[17] MPI Forum, *MPI: A messages-passing interface standard*, Int. J. Supercomput. Appl. 8, (1994).

[18] MPI Software Technology, *MPI/Pro.* http://www.mpi-softtech.com.

[19] Myricom, *Myrinet products.* http://www.myricom.com.

[20] C. Oosterlee, F. Gaspar, T. Washio, and R. Wienands, *Multigrid line smoothers for higher order upwind discretizations of convection-dominated problems*, J. Comput. Phys., 1 (1998), pp. 274–307.

[21] C. W. Oosterlee, *A GMRES-based plane smoother in multigrid to solve 3-D anisotropic fluid flow problems*, J. Comput. Phys., 130 (1997), pp. 41–53.

[22] C. W. Oosterlee, F. J. Gaspar, T. Washio, and R. Wienands, *Multigrid line smoothers for*

higher order upwind discretizations of convection-dominated problems, J. Comput. Phys., 139 (1998), pp. 274–307.

[23] M. Prieto, I. M. Llorente, and F. Tirado, *Data locality exploitation in the decomposition of regular domain problems*, IEEE Transactions on Parallel and Distributed Systems, 11 (2000), pp. 1141–1150.

[24] M. Prieto, R. Montero, D. Espadas, I. Llorente, and F. Tirado, *Parallel multigrid for anisotropic elliptic equations*, Journal of Parallel and Distributed Computing, Academic Press, 61 (2001), pp. 96–114.

[25] D. Quinlan, F. Bassetti, and D. Keyes, *Temporal locality optimizations for stencil operations within parallel object-oriented scientific frameworks on cache-based architectures*, in Proceedings of the PDCS'98 Conference, July 1998.

[26] U. Rüde, *Iterative algorithms on high performance architectures*, in Proceedings of the Europar'97 Conference, 1997, pp. 57–71.

[27] J. Thomas, B. Diskin, and A. Brandt, *Textbook multigrid efficiency for the incompressible Navier-Stokes equations: High reynolds number wakes and boundary layers*, Tech. Rep. 99-51, ICASE, 1999.

[28] M. Thompson and J. Ferziger, *An adaptative multigrid technique for the incompressible Navier-Stokes equations*, J. Comput. Phys., 82 (1989), pp. 94–121.

[29] U. Trottenbrg and K. Oosterlee, *Parallel adaptative multigrid*, Tech. Rep. 1026, GMD, 1996.

[30] T. Washio and K. Oosterlee, *Flexible multiple semicoarsening for three-dimensional singularly perturbed problems*, SIAM J. Sci. Comput., 19 (1998), pp. 1646–1666.

[31] C. Weis, W. Karl, M. Kowarschik, and U. Rüde, *Memory characteristics of iterative methods*, in Proceeding of The International Conference for High Performance Computing and Communications (SC99), IEEE Computer Society, November 1999.