JOURNAL OF COMPUTATIONAL PHYSICS **123,** 111–118 (1996)
ARTICLE NO. 0009

Parallel Implementations of 2D Explicit Euler Solvers*

L. GIRAUD†
CERFACS, 42 Av. Coriolis, 31057 Toulouse, France

AND

G. MANZINI‡
CRS4, via N. Sauro 10, 09123 Cagliari, Italy

Received July 14, 1994; revised August 1995

In this work we present a subdomain partitioning strategy applied to an explicit high-resolution Euler solver. We describe the design of a portable parallel multi-domain code suitable for parallel environments. We present several implementations on a representative range of MIMD computers that include shared memory multiprocessors, distributed virtual shared memory computers, as well as networks of workstations. Computational results are given to illustrate the efficiency, the scalability, and the limitations of the different approaches. We discuss also the effect of the communication protocol on the optimal domain partitioning strategy for the distributed memory computers. © 1996 Academic Press, Inc.

1. INTRODUCTION

The domain decomposition method is now a fairly well-established technique for the parallel solution of PDEs problems. In this work we consider an explicit solver for the bidimensional Euler equations and its parallelization on different multiprocessor architectures by a subdomain-partitioning strategy. Our main goal is to study the impact of this strategy for an efficient implementation of the same code on several different architectures. All our parallel implementations only differ within the low level routines, that manage the multidomain environment on the different machines and depend on the programming tools available. In order to develop an efficient parallel version, several different approaches both for the implementations and for the subdomain partitioning have been investigated. We present some implementations on a BBN TC2000, a distributed virtual shared memory computer, on an Alliant FX/80, a shared memory multiprocessor, as well as on a network of workstations using the packages PVM [2] and P4 [3].

The subdomain partitioning strategy adopted in this work is a special case of the more general multiblock technique where the grid is divided into several blocks linked by appropriate internal boundary conditions. In the field of computational fluid dynamics (CFD) applications the advantages of such an approach are multiple: it provides a way to adapt a structured grid to complicated geometries yielding better results in terms of convergence rates and numerical accuracy, and, finally, it helps to reduce the amount of data needed to be "in-core" for the processor, allowing a better fitting of the code on the memory of the machine.

Moreover, the possibility of performing calculations inside any block in a nearly independent way introduce a natural parallelism that can be directly exploited on distributed memory machines [4, 18].

The set of compressible Euler equations in the absence of diffusive phenomena and thermal exchanges can be written in the integral formulation as

$$\int_{\Omega} \frac{\partial U}{\partial t}\, d\Omega + \oint_{\partial\Omega} (F, G) \cdot \mathbf{n}\, ds = 0, \qquad (1)$$

where Ω is an arbitrary domain of integration defined by a closed curve $\partial\Omega$, and $\mathbf{n}$ is the outward normal vector to this curve. The conservative variables and the fluxes are given by:

$$U = \begin{pmatrix} \rho \\ \rho u \\ \rho v \\ \rho E \end{pmatrix}, \quad F(U) = \begin{pmatrix} \rho u \\ \rho u^2 + p \\ \rho u v \\ \rho u H \end{pmatrix},$$

* This work has been carried out with the financial support of CERFACS and partially with the financial contribution of the Sardinia Regional Authorities.
† email : giraud@cerfacs.fr, mail.
‡ email : manzini@crs4.it, mail.

$$G(U = \begin{pmatrix} \rho v \\ \rho uv \\ \rho v^2 + p \\ \rho vH \end{pmatrix}.$$

In the above formulae, ρ is the density, ρu and ρv are the two components of the momentum, ρE is the energy, p is the pressure, and H is the dynamic enthalpy. This last variable is related to the other quantities by $H = E + p/\rho$.

The system of Eqs. (1) is discretized using a structured cell-centered finite volume approach, where the cell-averaged approximation $\overline{U}$ of the conserved variables U is logically associated to each cell and advanced in time via a flux balance estimation,

$$\frac{d\overline{U}_{ij}}{dt} + \frac{1}{|\Omega_{ij}|} \sum_{k=1}^{4} (F_k, G_k) \cdot \mathbf{n}_k \, \Delta s_k = 0,$$

where $|\Omega_{ij}|$ is the measure of the area of the generic quadrilateral cell Ω_{ij} and the index $k = 1, 4$ refers to the four edges defining it.

A great number of numerical schemes have been developed in this framework in recent years for the simulation of compressible gas dynamics and are available in the literature. They differ essentially in the way they address the very difficult problem concerning the formation of the flow discontinuities like shocks and contact discontinuities. This difficulty is of primary importance because the overall accuracy of these calculations is very closely related to the accuracy with which flow discontinuities are represented [11].

In this work, the Eqs. (1) are discretized using a conservative shock-capturing high-order accurate Godunov-type scheme [8]. High-order accuracy is achieved by using a TVD-MUSCL or an ENO polynomial reconstruction both on conservative and characteristic variables. These polynomials are built by a special interpolation of the discrete set of cell-averaged data satisfying an a priori condition to ensure nonlinear stability, such as a total variation diminishing (TVD) or an essentially nonoscillatory (ENO) constraint [14]. The interpolation technique provides high-order accurate results in the regions of smoothness of the solution and avoids unwanted growth of spurious numerical oscillations near flow discontinuities (Gibbs phenomena) by imposing monotonicity or by limiting them up to the truncation error level [12].

Finally, fluxes estimation can be computed by three different Riemann solvers: the approximate one by Roe [16], the iterative one by Gottlieb and Groth [9], and the HLLE approximate one developed by Einfeldt [5]. The time-stepping is given by an explicit second- or third-order TVD Runge–Kutta scheme.

2. AN UNSTEADY COMPRESSIBLE TEST CASE: THE DOUBLE MACH REFLECTION

To show the ability of the code to capture shocks and contact discontinuities in two-dimensional compressible flows we consider the double Mach reflection with Mach number 3.72 of a shock wave on a 40° ramp. A detailed description of the parameters of the calculation and of the physical phenomena can be found in [13, 7]. In Fig. 1 we report the results of a numerical simulation on a 200×100 grid using the second-order ENO scheme on the conservative variables in space, the exact iterative Riemann solver, and the second-order TVD Runge–Kutta scheme by Shu [17] in time. The first 10 time steps of this calculation using 64-bit double precision have been taken to evaluate the performance of the different parallel implementations reported in Section 4.

The sequential performance of the implemented schemes are displayed in Table I. The times displayed in this table show that the second-order ENO scheme used in conjunction with the exact Riemann solver is poorly vectorizable. On the Convex, which is a vector computer, the speedup produced by the vectorization is 1.94, due to the implemented reconstruction technique and the exact Riemann solver which are essentially scalar (first row in Table I). However, this technique is highly accurate for CFD computations and efficient parallel implementations can be considered.

3. PARALLEL IMPLEMENTATIONS BASED ON DOMAIN DECOMPOSITION

The explicit numerical schemes considered in Section 1 have a natural parallelism. From the point of view of a parallel implementation, we consider an implementation more suitable for distributed memory computers based on a decomposition of the physical domain into subdomains assigned to different processors. That is an alternative to a more standard but also more strongly machine dependent loop-level tuning of the code convenient for shared memory computers.

Euler equations have a hyperbolic nature, which implies a finite velocity of propagation for all the linear and nonlinear waves forming the solution. That is, the value at a time at each point depends on the values at the previous time at some points lying in their neighbourhood, which is defined from a mathematical point of view by the domain of dependence [14]. It follows that the update of the flow variables of a cell in the mesh requires the knowledge of the flow variables in a local region around the cell in consideration. This is immediately clear when the MUSCL or the ENO reconstruction/interpolation methods are implemented, because they require a wide stencil of neighbouring cells. If a partitioning into subdomains is introduced, the update

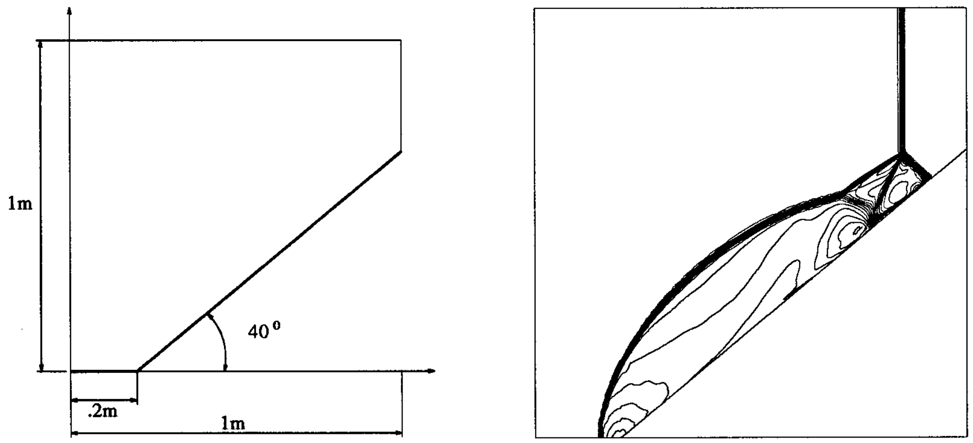

FIG. 1. Double Mach reflection with Mach number 3.72: domain of integration and computed solution.

of each subdomain requires the knowledge of the values of the flow variables inside a larger region, which is in fact the union of the domain of dependence of any cell inside that subdomain. For any cell close to an internal boundary, shared by two adjacent subdomains, a portion of this stencil falls outside the subdomain in a neighbouring one and requires interprocessor communication. The size of the regions of any subdomain needed by the neighbouring processors is a function of the order of the reconstruction and are referred to as the overlapping data areas.

The multidomain environment developed for any of our parallel implementations works in the following way:

• each processor takes the values on the overlapping data areas in the neighbouring domains which it needs for its update;

• each processor performs a complete update of its domain, which means it computes the reconstruction, it solves all the Riemann problems, and it does one time step for each cell;

• each processor makes available to its neighbours the updated values in the overlapping areas.

The first and the last steps are implemented in a very different way depending on the parallel programming paradigm available on the different target multiprocessors: a simple copy from local to global data structures on shared and distributed virtual shared memory systems; some send/receive calls from some message passing routines for distributed memory implementations. Furthermore, in order to have a complete set of Riemann problems for each cell close to the interfaces of the subdomains, each processor needs the outer interface state computed in the reconstruction phase. To minimize the communication among the processors and optimize the ratio of local to remote data accesses, we choose to compute redundantly on each subdomain one reconstruction for each cell adjacent to the subdomain boundary. This choice was essentially motivated by the fact that the cost of this redundant computation, in terms of number of operations, is not excessive and it avoids one more synchronization or message exchange during the updating step.

In the progress of developing a distributed version based on the domain decomposition technique introduced in Section 3 a parallel loop-level version for the shared memory

TABLE I

CPU Time (μs) of the Different Numerical Schemes per Cell per Timestep on a 200 × 100 Grid

Numerical scheme	IBM RS6000 model 550	Convex C220 without vectorization	Convex C220 with vectorization
Second-order ENO (Cons) + exact Riemann solver	534.85	1315.90	677.55
Second-order ENO (Char) + exact Riemann solver	713.45	1890.70	945.55
Third-order MUSCL (Cons) + exact Riemann solver	527.45	1075.20	476.70
Second-order ENO (Cons) + Roe's approximate Riemann solver	432.00	1456.60	501.45
Second-order ENO (Char) + Roe's approximate Riemann solver	609.25	2028.10	761.35
Third-order MUSCL (Cons) + Roe's approximate Riemann solver	417.09	1226.10	302.20

TABLE II

Speedup of the Shared Memory Implementation on a 200×100 Grid

	No. of domains					
Computer	1	2	4	8	16	20
Alliant FX/80	348.07	1.79	3.31	5.61	—	—
BBN TC2000	429.00	1.71	3.27	6.21	10.38	11.36

computers has been written on the BBN TC2000. This multiprocessor is a MIMD distributed virtual shared memory computer that exhibits features of both shared and distributed memory architectures. The nodes communicate through a high performance switch (the butterfly switch). It provides a transparent access by each processor to all locations in memory, whether local to a processor or remote on another processor. Within the parallel loop, the update on a subdomain is handled by the first free processor, which reads from a shared data structure A into local array data structure the values of the unknowns at the time step n for any subdomain and its overlapping area, computes locally the updated values, and stores back the partial results at time step $(n + 1)$ in a shared data structure B. When all the domains have been updated, B is copied into A. The implementation of the data movements between local and shared arrays was motivated by the works described in [1, 6], where this approach was the most efficient on this computer according to its memory hierarchy.

Furthermore, the portability of this shared memory version is achieved by the use of a loop-level parallelism available on a wide range of MIMD shared memory multiprocessors. Thus, experiments have been performed on a 26-node BBN TC2000 and also on an 8-processor Alliant FX/80 that is a vector multiprocessor machine with a shared memory accessed through a shared cache memory. The performance shown in Table II corresponds to the first 10 time steps of the simulation described in Fig. 1. The number in the second column represents the elapsed time T_1 for a stand alone execution of the single domain code, i.e., the most efficient sequential version of the program. In this version, the computations are done only on local data, without any copy. In the remaining columns, the number of domains (equal to the number of processors involved in the computation) is varied. For a fixed number of p processors only the performance for the best decomposition into p subdomains is given. For the parallel experiments we display the speedup SU_p that is defined by

$$SU_p = T_1/T_p,$$

where T_p is the elapsed time for a stand alone execution on p processors.

In Table II, it can be seen that the speedup on the Alliant becomes worse than on the BBN when eight processors are used. This behaviour can be explained by a higher cache memory contention that can occur during all the computation when the eight processors of the Alliant are involved. On the BBN, the memory contention can only occur during the copies at the beginning and at the end of each time step on a subdomain, since all the computations are performed on local data. However, when the number of processors increases, the memory contention effect also appears (combined with a decrease of the granularity of the parallel tasks), although the shared arrays are declared *interleaved*; in this case, the arrays are distributed over all the nodes of the machine by pieces whose size is the length of the cache lines (16 bytes, or 4 single or 2 double precision words). Nevertheless, on the BBN the asymptotic speedup of this implementation seems to be reached on 20 nodes for a 200×100 grid.

The limitation of the shared memory implementation on the BBN is clearly due to the amount of data movement. One way to reduce this data movement is to attach one subdomain per processor that keeps the data in the local memory of the node. This approach corresponds to a usual domain decomposition implementation on a distributed computer, where each processor performs the update on one subdomain. The data locality combined with a coarse grain parallelism gives rise to natural and efficient implementations on this class of architecture. Different implementations have been considered depending on the tools available on the target computer: the BBN TC2000 and a network of IBM RS/6000s connected by Ethernet. On the BBN, in order to implement the communication using the virtual shared memory, each processor only writes in some shared arrays the updated values it computed on the overlapping data areas, synchronizes, and reads from the shared arrays the values computed by its neighbours on the overlap subregions. For this implementation the communication is done through the virtual shared memory and the synchronizations are implemented using locks. Furthermore, in order to limit the shared memory contention, the shared data structures are declared *interleaved*. In the rest of this paper, this version is referred to as "BBN Fortran," because it has been implemented using only the BBN Fortran extensions.

The implementations using P4 and PVM, both on the BBN and on the network of workstations, are based on a master-slave scheme. The master is in charge of all the initializations; it broadcasts the overall information and receives at the end of the simulation the results of the computation. This process does not perform any other computation, it is only an I/O interface between the slaves processes and the user. The slave processes receive all the characteristics of the simulations from the master and then start to iterate. At the end of each time step, they exchange

TABLE III

Speedup of the Distributed Implementations on a 200 × 100 Grid

Computer	No. of domains 1	2	4	8	16	20	80
BBN Fortran	429.00	1.98	3.91	7.61	13.70	16.14	54.86
P4 BBN	429.00	1.99	3.83	7.67	13.98	16.76	—
P4 RS/6000	140.54	1.92	3.73	7.11	—	—	—
PVM RS/6000	140.54	1.83	3.39	5.54	—	—	—

the updated values on the overlap subregions. For this implementation, the send/receive primitives express both the communication and the synchronization.

4. EXPERIMENTAL RESULTS

The performance shown in Table III corresponds to the first 10 time steps of the simulation described in Fig. 1. We can verify that the distributed approach on the BBN is much more efficient than the shared one. On the 200 × 100 grid the speedup observed on the BBN using the distributed implementation is 16.14, while it is only 11.36 for the shared approach on 20 nodes. It can also be observed, that on the network of workstations the performance of the PVM implementation is always worse than the P4 one. The inefficiencies were found to be mainly caused by both the PVM architecture and the Aix operating system, which is not very efficient performing context switches and semaphores handling, as mentioned in [15]. With P4 the messages are directly sent from the sender to the recipient, while in PVM they are sent from the sender to a local daemon, from the local daemon to the remote one, and finally are received by the recipient. This more complicated path is the cause of both context switches (from daemon to user's process) and semaphore handling for the communication between these processes. This overhead is particularly penalyzing for small meshes; on a 100 × 50 grid the speedup is only 2.44 with PVM and 4.82 with P4 on eight workstations. These results also illustrate the coarse-grain parallelism required on a network of workstations. On the BBN, since the CPU is slower and the communication network faster, the performance in terms of speedup is better. On the previously mentioned mesh size on eight BBN nodes the speedup is 7.05. Further, these experiments illustrate the portability of the codes developed using such packages, as the results displayed in this table correspond to the same code performed on the different platforms. The curves depicted in Fig. 2 show the effect of the granularity of the parallel tasks on the performance. In particular, it can be seen that for the smallest test case, a 200 × 100 mesh, the parallel code is slower on 100 nodes than on 80 nodes; this observation is no longer true on a 400 × 200 grid.

We conclude our evaluation of this code on the BBN by considering the performance using larger problems. In particular, we wish to determine the *scaled* speedup that one can expect using this highly parallel solver on parallel machines. The *scaled* speedup is defined as

$$S_p^* = pT_1^*/T_p^*,$$

where T_p^* is the time required to complete 10 time steps on a problem with $p \times n$ discretization points using p processors; see [10]. The scaled speedups observed on the Lawrence Livermore 128-node BBN TC2000 are displayed in Fig. 2 for $n = 64 \times 64$. The best observed speedup is close to 80 when 100 processors are in use. This behaviour illustrates the scalability of this parallel code. That is, if the number of grid points is increased linearly with the number of processors, the computational time is still almost constant (the curve of the *scaled speedup* is almost linear). This feature also illustrates the interest of such parallel codes that allow addressing bigger problems on a bigger configuration of a computer, without increasing significantly the global elapsed time required to perform one timestep. However the stability constraint that link both space and time discretization steps for the explicit schemes has some consequences on the performance of the parallel scaled code for a complete simulation:

1. If the mesh space is kept unchanged, the time step that ensures the stability of the scheme does not change. The scalability allows us to perform the simulation on a bigger physical domain without increasing significantly the

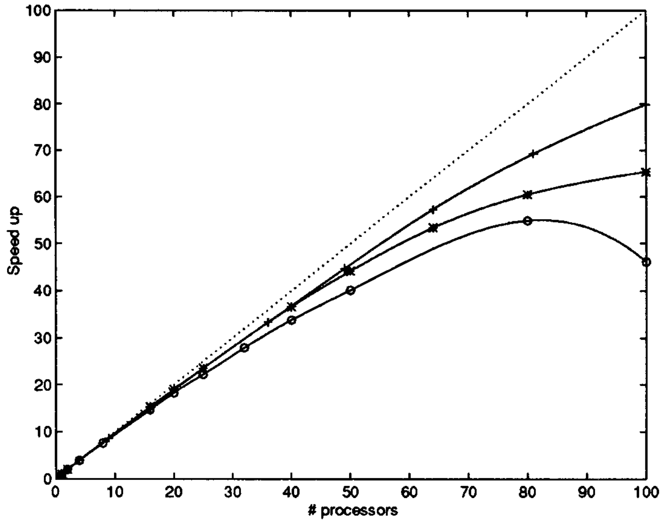

FIG. 2. Speedups on a 128-node BBN TC2000: ○, 200 × 100 mesh; *, 400 × 200 mesh; +, scaled speedup (64 × 64).

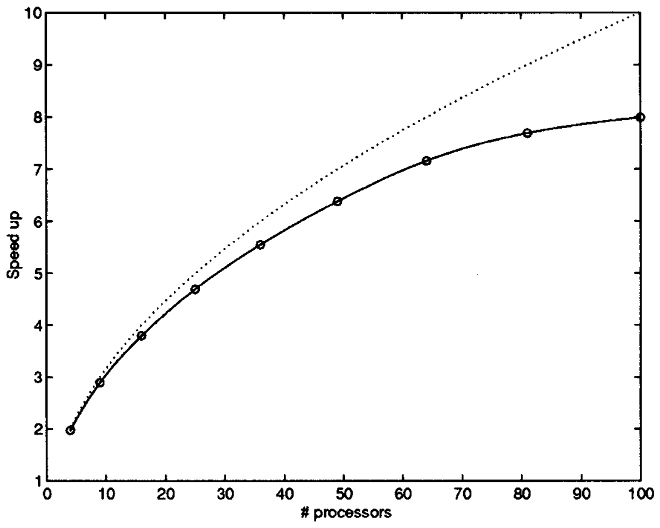

FIG. 3. Scaled speedups for the complete simulation.

computational time for a complete experimentation. In that case the speedups of the scaled code is the one displayed in Fig. 2.

2. If the mesh is refined, according to the number of processors on the same physical domain, the timestep has also to be reduced in accord with a CFL constraint, in order to ensure the stability of the schemes,

$$\Delta t \leq \Delta_{\text{allowed}} = \left(\frac{1}{\Delta t_x} + \frac{1}{\Delta t_y}\right)^{-1},$$

where the two inviscid signal "frequencies" $1/\Delta t_x$ and $1/\Delta t_y$ can be roughly estimated as

$$\frac{1}{\Delta t_x} \sim \frac{|u| + c}{\Delta t_x}, \quad \frac{1}{\Delta t_y} \sim \frac{|v| + c}{\Delta t_y}.$$

If the mesh is refined linearly in one direction, the timestep has also to be linearly refined. Then the number of timesteps to perform a complete experimentation increases linearly with this respect (i.e., as the squared root of the number of processors for uniform box decompositions). In this latter case, the theoretical speedup to perform a complete simulation on n^2 processors is bounded by n (and would be bounded by n^2 on n^3 processors for 3D calculations with the same assumptions on the relation between time and space steps). From our experiments the speedups for a complete simulation are displayed in Fig. 3.

4.1. *Partitioning Strategies*

For all our experiments, two different strategies have been explored for decomposing in subdomains the global

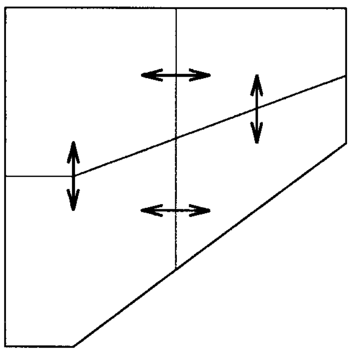

FIG. 4. Example of four box-decomposition.

mesh: box-partitioning (e.g., see Fig. 4), and slice-partitioning (e.g., see Fig. 5). It can be observed that for a fixed number of subdomains, box-partitioning minimizes the amount of communicated data while slice-partitioning minimizes the number of messages exchanged between the processors. These features are illustrated in Figs. 4 and 5 with an example of decomposition into four subdomains. In this example an $n \times n$ mesh computation requires 8 messages of length $n/2$ per time step if a box-decomposition is applied and 6 messages of length n if a slice-decomposition is considered.

On the BBN, where data are exchanged through interleaved double precision arrays, the communication through the virtual shared memory is similar to packets switching communication, where packet size is 16 bytes (a cache line length). The communication time can be modeled by

$$T_{\text{Packet}} = \frac{N_{\text{Bytes}}}{16} \times (16 \times \text{Bandwidth}^{-1} + \text{Latency}).$$

For such an interconnection network, the communication time will be minimum when the amount of the exchanged

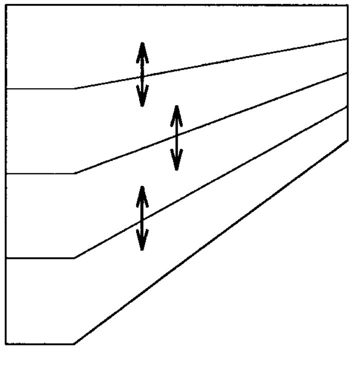

FIG. 5. Example of four slice-decomposition.

TABLE IV

Averaged Communication Time (ms) per Time Step for the Distributed Implementation on a 200×100 Grid

	Partition					
Computer	4×1	2×2	1×4	20×1	5×4	1×20
BBN Fortran	137	96	188	810	416	1108

data will be minimum. In Table IV we display the time spent in communication on the BBN, i.e., the time corresponding to writing and reading the shared arrays. It can be seen that for a fixed number of domains the box-decomposition effectively provides us with the less expensive communication.

Instead, on a network of workstations using modest length messages, the communication through Ethernet is similar to a circuit-switching model and the communication time can be modeled by

$$T_{\text{Circuit}} = \text{Latency} + N_{\text{Bytes}} \times \text{Bandwidth}^{-1}.$$

For modest length messages when the latency time is high, compared to the bandwidth, the minimum will be achieved by minimizing the number of messages, i.e., the number of accesses to the network, which also reduces the probability of collision on Ethernet.

These simple models explain the different behaviour of the code when the two different partitioning strategies are adopted. Especially on the BBN, box-partitioning yields better performance by minimizing the amount of exchanged data, while slice-partitioning yields better performance on a network of workstations connected via Ethernet, by minimizing the number of exchanged messages, due to the high latency time to access Ethernet compared to its bandwidth and to the decrease of the probability of collision.

5. CONCLUDING REMARKS

This experience shows that for some applications of CFD based on explicit time-advancing schemes, a domain decomposition approach can result in a very good strategy to efficiently parallelize a code.

On the BBN, the parallelism provided by the explicit scheme is efficiently exploited since the observed speedups are around 65 on 100 nodes for a medium size (400×200) mesh, since in this case each processor only works on a 40×20 subgrid. The speedup is close to 80 for the 640×640 mesh used in the *scaled* speedup experiments. This performance illustrates the scalability of the resulting implementation. TVD and ENO high-order shock-capturing methods generally require enough computation to provide us with a good ratio between communication and computation. This feature can be directly exploited for efficient implementations on networks of workstations. In this case, we observed a speedup greater than 7 on 8 workstations with very low traffic on the local area network. Of course, an increase in traffic would have a negative impact on the performance due to a bad balance between communication and computation time because of Ethernet contention.

Furthermore, our experiments show that even if the decomposition has no impact on the convergence of the explicit scheme, it has an impact on the elapsed computational time. According to simple models of communication, the observed results show that on distributed multiprocessors with packet switching communication, the box-decompositions give the best performance, while on systems with circuit switching communication and high latency the slice-decompositions are the most efficient. Thus, on the BBN the box-decompositions provide us with the best performance of the parallel code, and slice-decompositions are the most efficient on the network of workstations.

Last, the parallelizations of the Euler solver, first on shared memory multiprocessors, then on a virtual shared memory computer, and finally on distributed memory multiprocessors using message-passing, shows that there is a possibility of progressively moving a code from shared memory towards distributed memory programming paradigms using an intermediate step that is the virtual shared memory programming paradigm. The parallel versions of our code is easily portable on most of the MIMD multiprocessors and heterogeneous networks of computers currently available, as well as on the MPP, as the Cray T3D, due to application of standard public domain message-passing packages like P4 and PVM.

ACKNOWLEDGMENTS

We thank the Lawrence Livermore National Lab for having provided us with an access to the 128-node BBN TC2000. It allowed us to complete our experiments and evaluate the scalability of the developed code.

REFERENCES

1. P. R. Amestoy, M. J. Daydé, I. S. Duff, and P. Morère, *Int. J. High Speed Comput.,* **7,** 21–44 (1995).
2. A. Beguelin, J. Dongarra, A. Geist, R. Manchek, and V. Sunderam, Tech. Rep. ORNL/TM-11826, Oak Ridge National Laboratory, Tennessee 37831, 1992 (unpublished).
3. R. Butler and E. Lusk, *User's Guide to the P4 Parallel Programming System* (Mathematics and Computer Science Division, Argonne National Laboratory, 1992).
4. F. Dellagiacoma, S. Paoletti, F. Poggi, and M. Vitaletti, "Multidomain Computations of Compressible Flows in a Parallel Scheduling Environment, 1992," in *Parallel CFD*'92 *Conference* (unpublished).
5. B. Einfeldt, C. D. Muntz, P. L. Roe, and B. Sjogreen, *J. Comput. Phys.* **92,** 273 (1991).

6. L. Giraud, *Int. J. High Speed Comput.,* **7,** 161–190 (1995).
7. L. Giraud and G. Manzini, Tech. Rep. TR/CFD-PA/93/49, CERFACS, Toulouse, France, 1993 (unpublished).
8. S. K. Godunov, *Mat. Sb.* **47,** 271 (1959).
9. J. J. Gottlieb and C. P. T. Groth, *J. Comput. Phys.* **78,** 437 (1988).
10. J. Gustafson, G. Montry, and R. Benner, *SIAM J. Sci. Stat. Comput.* **9,** 609 (1988).
11. A. Harten, ICASE Report 91-8 (unpublished).
12. A. Harten, S. Osher, B. Engquist, and S. R. Chakravarthy, *Appl. Numer. Math.* **2,** 347 (1986).
13. P. A. Jacobs, ICASE Interim Report 18 (unpublished).
14. R. J. LeVeque, *Numerical Methods for Conservation Laws,* Birkhauser, Basel, 1990.
15. G. Richelli, Tech. Rep., ECSEC, Italy, 1992 (unpublished).
16. P. L. Roe, *J. Comput. Phys.* **43,** 357 (1981).
17. C. W. Shu, ICASE Report 90-55 (unpublished).
18. Y. Yadlin and D. A. Caughey, ‘‘Block Implicit Multigrid Solution of the Euler on a Parallel Computer,’’ in *Parallel Computational Fluid Dynamics: Implementation and Results,* edited by H. Simon (MIT Press, Cambridge, MA, 1992), 127.