Parallel computing on Unix workstation arrays

F. Reale, F. Bocchino, S. Sciortino

Istituto e Osservatorio Astronomico, Palazzo dei Normanni, 90134 Palermo, Italy

Abstract

We have tested arrays of general-purpose Unix workstations used as MIMD systems for massive parallel computations. In particular we have solved numerically a demanding test problem with a 2-D hydrodynamic code, generally developed to study astrophysical flows, by executing it on arrays either of DECstations 5000/200 on Ethernet LAN, or of DECstations 3000/400, equipped with powerful Alpha processors, on FDDI LAN. The code is appropriate for data-domain decomposition, and we have used a library for parallelization previously developed in our Institute, and easily extended to work on Unix workstation arrays by using the PVM software toolset. We have compared the parallel efficiencies obtained on arrays of several processors to those obtained on a dedicated MIMD parallel system, namely a Meiko Computing Surface (CS-1), equipped with Intel i860 processors. We discuss the feasibility of using non-dedicated parallel systems and conclude that the convenience depends essentially on the size of the computational domain as compared to the relative processor power and network bandwidth. We point out that for future perspectives a parallel development of processor and network technology is important, and that the software still offers great opportunities of improvement, especially in terms of latency times in the message-passing protocols. In conditions of significant gain in terms of speed up, such workstation arrays represent a cost-effective approach to massive parallel computations.

1 Introduction

Much effort has been devoted in the recent past to achieve high performances of massive computing on dedicated distributed-memory parallel systems, by developing specific parallel solutions for particular problems (e.g. [1], [2]). The hardware has been sometimes tailored on the problem to be solved, but, more usually, the MIMD systems allow to change the processor connectivity to match a variety of problems. Since parallelism offers potentially infinite computing resources, a variety of new computational solutions have been developed both on the algorithms and on the practical implementations in a wide range of applications.

Recently the idea of parallel systems consisting of arrays of general-purpose computers has been proposed and has received a vigorous push by the computing community, for some obvious reasons:

1. low costs of initial settlements; the user can access parallel computing on the workstation on his desk.
2. a network of workstations is inherently easily expandable and highly modular.
3. portability; applications tailored for a workstation array, typically Unix-based, will be widely executable in the scientific environment.
4. optimization of usage of actual computing resources; workstations as components of large computers for massive and demanding numerical applications fully exploit the local computational power, avoiding long idle periods.

The interest for this approach is clear from the recent availability of several software packages, which allow the parallel execution on different architectures, like P4, PVM, Express, Linda.

In this work we test the execution of specific applications on a Local Area Network of Unix workstations used as a proper distributed-memory MIMD system. Here we concentrate on data-domain decomposition problems, representative of a wide class of possible parallel applications and suitable for a straightforward geometric parallelization. In this case the topology of the connections between the processors working in parallel to the solution of the same problem maps accurately the geometry of the data-domain the problem operates on.

We take advantage of the experience previously acquired in using dedicated parallel systems for the execution of multi-dimensional hydrodynamic codes for astrophysical applications. In particular we have used the Parallel Network and Communication Library (PNCL) [3] originally developed for the parallelization of data-domain decomposition codes on a Meiko Computing Surface (CS-1) with T800 transputer boards and Intel 860 boards, which is a dedicated parallel MIMD system integrated in a Unix Local Area Network. For this work we have ported PNCL to work on arrays of workstations and then we have taken as test code the 2-D hydrodynamic code [4] already appropriately parallelized by means of PNCL. The code is the same as that extensively tested on the Meiko CS-1 and therefore, by running *exactly the same application*, we can compare directly the performances obtained on the workstation arrays with those obtained on a dedicated system. Of course the point is not so much to achieve performances better than those obtained on dedicated systems, as to show that arrays of workstations are appropriate for a whole class of problems and that they open interesting perspectives for the future when more efficient and faster networks will be easily available, like the ones based on the fiber-optics technology.

For this purpose, we have used one of the aforementioned public-domain parallel environment, the Parallel Virtual Machine (PVM), which has provided us with the low-level communication and task-management procedures to port the Meiko CSTools version of PNCL.

We will show how, for data-domain decomposition problems like the ones we are interested in, the performances depend critically on the balance of the processor unit power, of the network communication speed and of the size of the problem data-domain.

This paper is organized as follows: in Section 2 we describe the parallel system, the software package, and the applications we have made the tests on; in Section 3 we describe the results achieved, and compare them with results previously obtained on other parallel systems; in Section 4 we discuss our results, make predictions in view of possible future hardware improvements and draw our conclusions.

2 The parallel system and applications

2.1 The Hardware: Unix Local Area Networks

For part of our tests we have used the computer system available at the Osservatorio Astronomico di Palermo, consisting of a Unix Local Area Network (LAN). The computers we have used inside the LAN for our tests are basically Ultrix DECstations 5000 (but also a DECsystem 5500), which run with a MIPS R3000 processor. We have also used a recently acquired DEC3000, equipped with an Alpha processor and OSF/1 operating system, and an older SUN 4/370 workstation, with a RISC Sparc processor and SUN/OS 4.0.3 operating system. The network backbone consists of a TCP/IP Ethernet thin/thick wire connection, with a nominal data throughput of 10 Mbit/s. Our system is schematically shown in Fig. 1.

The rest of our tests were executed on a LAN consisting of several Digital workstations of the Alpha series, located at CASPUR, University "La Sapienza", Rome, with faster FDDI connections (100 Mbit/s).

2.2 The Low-level Software: the PVM

Among the software systems available for the execution of parallel applications on non-dedicated computer arrays, we have selected the PVM [5], which is of public domain[1] and is quite popular among the parallel-computing community, as basic parallel environment.

The PVM software system has been developed at the Oak Ridge National Laboratory, Tennessee, USA, and allows to program parallel applications on arrays consisting of heterogeneous Unix computers, from workstations, like those from Digital, SUN, Silicon Graphics, HP, IBM, to larger systems, also with vector facilities, like Alliant FX/8, Thinking Machine CM series, Convex C-series, Cray's, Intel Paragon.

PVM consists essentially of three components: a daemon process, running permanently on the allocated machines, a library of routines for the development of parallel Fortran and C applications, and an interactive session for the configuration and management of the virtual machine. The package is built so to handle transparently network tasks like the message routing and the data conversion for incompatible architectures. On the other hand, the user is left with wide capabilities to configure his array and to control it.

Message-passing and processor interconnections are set up by means of the library, which is organized as a collection of subroutines and accomplishes a series of tasks: to control processes, to get information on and from PVM, to configure and allocate processes dynamically, to exchange messages and/or signals between processes, to allocate buffers.

In our approach we have made extensive use of the library routines, in particular those for process control and message-exchange, to port the Meiko version of PNCL to the PVM environment. We have paid particular attention for the new version to be entirely compatible with, and to have the same calling sequence as the previous one. Therefore now the parallel Fortran programs developed using our library run equivalently on CStools and PVM based systems. The only very minor, extra-operation required by the version with PVM is to include at the beginning and at the end of the master and the slave main programs two instructions, one for the initialization ("pvmfmytid") and one for the release of the PVM processes ("pvmfexit").

[1]The PVM can be obtained by anyone by sending email to *netlib@ornl.gov* with the message "send index from pvm3" and then following the instructions of the return message.

2.3 The High-level Software: the parallel network and communication library

We now briefly summarize the main features of our PNCL; for more details see Reale et al. [3]. The library is focussed on the parallelization of applications subject to data-domain decomposition and consists of a set of high-level routines which, if included at the appropriate locations, should automatically make the code run parallel on the selected processor array. Our parallel implementation is based on a master-slave paradigm: a single "master" processor executes the input/output operations and synchronizes the contacts between the external world and the "workers" slaves; multiple slave processors execute the bulk of the computations on fractions of the original data-domain.

The library allows for the allocation of different numbers of processors and for an easy change of the processor topology. Subdomains of roughly the same size are distributed to all the processors. This warrants a rough load balancing. The tasks addressed by the library routines are:

- Connect the processors according to the selected topology.
- Split the computational domain into as many subsections as the number of allocated processors with a mapping of the domain structure consistent with the processor topology.
- Assign and transmit scalars, vectors and matrices to the processors.
- Collect and reassemble scalars, vectors and matrices from the slaves for output.
- Exchange vectors of side values between adjacent processors.

In our previous work [3] we had been able to outline a recipe to parallelize sequential Fortran codes, which can be summarized as follows:

- Split the sequential source code into the two separate programs, *master* and *slave*. The same slave program is executed by all worker processors, with slight differences which depend on the position of each processor inside the processor topology.
- Include in *master* single-shot (typically initial setup and configurations) and input/output operations.
- Include in *slave* iterative computer-intensive operations.
- Insert calls to the relevant network and communication routines inside *master* and *slave*.
- Adjust and rearrange output routines (whenever necessary).
- Adjust boundary conditions.

2.4 The test

In order to test the feasibility of an array of workstations as parallel distributed-memory computational system, we have considered a standard application extensively used previously for benchmarking other systems: the description of the double Mach reflection of a strong shock wave (Mach 10) against a solid wedge [6, 4], computed with a two-dimensional hydrodynamic code developed at the Osservatorio Astronomico di Palermo to study a wide variety of astrophysical flows [4]. The code solves the equations of mass, momentum and energy conservation for a compressible fluid and is based on an explicit fully two-dimensional Flux Corrected Transport (FCT) scheme [7, 8, 9, 10, 11, 12], which is particularly well suited to treat shock waves, but has been accurately refined to describe other regimes as well. The code, which has recently been applied for detailed studies of thermal instabilities in stratified gases [13, 14] solves the hydrodynamic equations in the conservative form:

$$\frac{\partial \rho}{\partial t} + \nabla \cdot (\rho \mathbf{v}) = S_m \tag{1}$$

$$\frac{\partial (\rho \mathbf{v})}{\partial t} + \nabla \cdot (\rho \mathbf{v}\mathbf{v}) = -\nabla p + S_q \tag{2}$$

$$\frac{\partial u}{\partial t} + \nabla \cdot [(u+p)\mathbf{v}] = S_u \tag{3}$$

$$u = \frac{1}{2}\rho v^2 + E, \qquad\qquad p = (\gamma - 1)E$$

where ρ is the mass density, $\mathbf{v}$ the fluid bulk velocity, p the pressure, u the total energy density, E the internal energy density, S_m, S_q and S_u possible source terms of mass, momentum and energy, respectively, γ is the ratio of the specific heats (C_p/C_V). As described in Paper 1, the code in this standard test runs with Scheme C of Reale, Peres and Serio [4], in which the limiter is a hybrid between that devised by Zalesak [10] (see also Kössl and Müller [12]) and that originally developed by Boris and Book [7]; the diffusion and antidiffusion coefficients are those suggested by Boris and Book [9]. For a more complete description of the code see Reale, Peres and Serio [4], and Reale, Barbera and Sciortino [3].

The code had been previously parallelized [3] and here we have used the same identical parallel code, with the same identical calls to the high-level library described above, as the one already used on a dedicated parallel system, the Meiko CS-1, based on CSTools package. We remind that we did not have to arrange a specialized version of the library to run on arrays of non-homogeneous processors, since PVM transparently accomplishes this task.

To solve the problem of the wedge interaction, the source terms reported for the general hydrodynamic equations are absent and γ is assumed 7/5 (adequate for air, for instance). The inclination of the wedge wall with respect to the plane shock front is 60°. The problem admits a self-similar solution (see [4] and [3], for further details). This test can be run with various resolutions of the spatial grid. In our case we have considered a rectangular grid with the sides 372×122 points, each side being four times as long as the respective ones used for benchmarking in Reale, Barbera and Sciortino [3], therefore with a total number of cells 16 times as large.

This larger grid implies a larger amount of computations per processor and therefore allows to achieve a significant speedup on the general-purpose system we address here, in which the communications will presumably be less efficient than in a dedicated transputer-based system.

3 Results

We have benchmarked the 2-D hydrodynamic code by running it on the 372x122 computational grid of square cells over 713 integration time steps. We report in Table 1 the execution time of the original sequential code on several computers, spanning from PC's to the last generation Cray supercomputer, and using the same identical input parameters as for the parallel version of the code we will report on below. In Table 1 we report, for each computer, the compiler used, the net CPU time and the speedup taking the Digital DECsystem 5500 as unit. The performances reported, which range between 1.5 days for the PC to less than two minutes with the Cray C-90, can be compared with those of the parallel code on the arrays we have used.

We have executed the parallel code both on a dedicated parallel system – a Meiko CS-1 containing up to four Intel i860 processors – and on arrays consisting of general purpose workstations connected in a LAN. In particular we have considered:

1. Arrays of DECstations of the 5000 series + 1 SUN 4/370 + 1 Digital ALPHA 3000 Model 400, connected by a packet collision ETHERNET LAN.

2. Arrays of DECstations 3000, Model 400, connected by a fiber-optics based FDDI LAN (100 Mbit/s).

Since the number of cells along x axis (372) is about three times as large as the number of cells along the y axis (122), a linear topology of processors disposed along the x axis results to be the most convenient in arrays of few processors, at variance with what occurs for square computational domains [3]. In this linear topology each processor exchanges relatively short messages with two processors at most, the adjacent ones. For proper 2-D topologies (e.g. 2×2, 3×2) we expect slightly degraded performances because every processor has to exchange values with two processors at least, up to four processors, and because of the highly asymmetric domain.

In Table 2 we report the results obtained on the Meiko CS-1, on linear arrays up to four processors. For each array, composed by a number $N_X \times 1$ processors, we report (a) the execution time t_{par} on N_{proc} processors, (b) the speed-up S with respect to the execution time t_{seq} of the sequential code on a single processor, and the fractional parallel efficiency ϵ defined as:

$$\epsilon = t_{seq}/(N_{proc}t_{par}) = S/N_{proc} \tag{4}$$

A maximum efficiency 1 is expected for a perfect parallel execution for data-domain decomposition problems[2]. The larger the weight of the communication overhead on the overall execution time, the lower will be the efficiency obtained.

On the arrays of few processors on the Meiko CS-1 we obtain parallel efficiencies steadily above 85%. Although we might expect poorer results, since the message-passing is performed by transputer links while i860's provide much enhanced computational speed with respect to transputers, the relatively large size of the computational subdomains assigned to each processor makes the communication bandwidth still adequate to reach high gains.

In Table 3 we report the benchmark results of the workstation arrays of configuration 1 on various topologies. At variance with the case of dedicated parallel systems, we have now to pay attention to additional factors in analyzing our results, and, first of all, the share of resources on each of the allocated machines. The tests reported have been executed in minimum load conditions and should therefore be little influenced by multitasking activity. We have to consider also that different processors take different times for the scheduling and execution of the communication processes. Therefore we expect that the communication overhead will depend not only on the bandwidth of the network connections, but also on the processors.

We have considered a linear topology of processors, each assigned with subdomains aligned along the longer coordinate axis (the X axis), except for the case 2×2 processors. Up to 4 processors we have a homogeneous array of DECstations. Arrays of 5 processors include either one DEC ALPHA station or a slower SUN 4/370 workstation. If we look exclusively at the results obtained with the linear topology, we see that up to 4 processors the speedup is always increasing with increasing number of processors, although remaining below 2. It is worth noting that in general for a linear processor topology we expect the time spent in the communications to remain practically constant whatever the number of processors, since this time should depend solely on the size of the boundary vector (always 122 in our case). For decreasing size of the computational subdomains, i.e. increasing number of processors, the time spent by each processors for the computations decreases, and therefore also the efficiency. The observed significant reduction of the parallel efficiency is indicative of a high weight of communications.

The topology 2×2 takes longer than the 4×1 as expected. The highest speedup (2.57) is achieved with the topology 5×1 including the DEC ALPHA. The efficiency is higher than that of the 4×1 topology, likely due to the smaller dead times induced by the more powerful ALPHA processor. The efficiency is instead reduced considerably in the array with the slower SUN workstation. In absolute terms, we note that at most our Ethernet LAN of 5 DECstations 5000[3] performs almost as 1/2 a single DEC3000/400.

[2] Indeed one may obtain even larger efficiencies because processors usually manage better smaller-sized matrices.

[3] Actually the array consists of 4 DECstations 5000 + 1 DEC-Alpha; however the system is indeed as fast as 5 DECstations 5000 since it is synchronized to the speed of the slowest processor in the array.

In table 4 we report the results of the configuration 2, i.e. on arrays with up to 6 workstations with Alpha processors connected by FDDI. The tests have been executed in night jobs, in conditions of minimum machine load. The arrays consist now only of homogeneous processors. In addition to linear topologies, from 2×1 to 6×1 processors, we have considered also to cases with proper 2-D topologies, $2 \times 2 = 4$ and $3 \times 2 = 6$ processors.

In table 4 we observe that the speedup grows steadily with the number of processors of the linear topology up to 3.29 with 6 processors, i.e. a parallel efficiency of 55%. This efficiency is even higher than the one obtained with 5 processors in configuration 1, i.e. in configuration 2 the fractional overhead of communications is smaller. This is reasonable since, even if the computational power has increased by a factor ~ 5 (cf. Table 1), the bandwidth of FDDI is of the order of 10 times higher than Ethernet. Although the efficiency is rapidly decreasing we still expect to obtain a significant gain in terms of speed up for a number of processors up to ~ 10. Note that the topology 2×2 yields a performance close to 4×1, while 3×2 performance is significantly degraded with respect to 6×1.

We can assume that the execution time on the array is a linear combination of the time t_{calc} spent for the algebraic calculations and of the time t_{comm} spent for exchanging messages between processors [3]:

$$t_{par} = t_{calc} + t_{comm} \tag{5}$$

where

$$t_{comm} = (N_{bit} t_{bit} + t_{lat}) N_{comm} \tag{6}$$

t_{lat} being a start-up time taken to initialize any communication between two processors (*latency time*), and t_{bit} the proper transfer time of each of the N_{bit} bits across the link, N_{comm} the total number of times that the procedure for message-exchanges between adjacent processors is called inside the slave program.

We have used expression (6) to estimate accurately the two characteristic times of communication t_{lat} and t_{bit}, by measuring t_{comm} from couples of two otherwise identical simulations but with a significantly different number of bits transferred (N_{bit}). In particular in one of them we have set the transferred boundary vectors to double precision, therefore doubling the size of the messages exchanged. The execution on the workstation arrays is influenced by a number of unpredictable factors due to the presence of a pervasive network management and operating system, and therefore we obtain values affected by relatively large fluctuations. By averaging on the various topologies for each configuration we obtain the following values: $t_{bit} = 2.7 \pm 1.5 \; 10^{-7}$ sec and $t_{lat} = 1.8 \pm 0.3 \; 10^{-2}$ sec for configuration 1, $t_{bit} = 3.4 \pm 1.2 \; 10^{-8}$ sec and $t_{lat} = 1.1 \pm 0.3 \; 10^{-3}$ sec for configuration 2, to be compared to $t_{bit} \approx 1.2 \; 10^{-7}$ sec and $t_{lat} \approx 7 \; 10^{-4}$ sec for the Meiko CS-1. The inverse of the bit transfer rate can be taken as a sustained bandwidth of the network connections; we obtain 3.7 Mbit/s for Ethernet LAN and 26 Mbit/s for FDDI LAN, which scale reasonably with the respective nominal bandwidths. As for the latency times, they are higher on configurations 1 and 2 than on the Meiko CS-1, which is a dedicated system. We notice that the latency time scales inversely with the power of the CPU: there is a ten-fold improvement from DECstations 5000 to DECstations 3000/400. Incidentally we mention that we have obtained approximately the same value of t_{lat} as those reported for configuration 2 with some tests made on a configuration of workstations Alpha linked in a Ethernet LAN. The latency time is a characteristic also of the software implementation, i.e. of PVM in this case. We may expect significant improvements for this in future releases of the software. At the moment an array of 6 DEC 3000's perform as $\sim 1/2$ the Cray YMP and $\sim 1/5$ the Cray C-90.

In Fig. 2 we report the execution times obtained in our various configurations as a function of the number of processors. In the figure we report also the curves of maximum (100%) efficiency ($\epsilon = 1$). As shown, the Meiko CS-1 better approaches the best line, although we present results only for a limited number of processors (up to 4). However, we notice that, in terms of absolute execution times, one needs at least 4 i860 processors to achieve the performance of a single

Alpha processor. Despite the efficiency of configuration 2 degrades more rapidly as the number of processors increases, we expect that few Alpha-stations will perform as tens of i860's and this prospect poses severe questions about the cost-effectiveness of the two architectures, as will be further discussed in the following.

4 Discussion and conclusions

4.1 The present

In the previous section we have reported the results of our benchmarking on various parallel systems. Our goal was to test the feasibility of massive computations on parallel systems based on general-purpose workstations, in particular for applications suitable to data-domain decomposition. In order to evaluate the performances for realistic applications, we have tested the parallel execution of a fully-developed two-dimensional hydrodynamic code for modelling astrophysical plasma flows, which requires periodic message exchanges between processors.

We could take advantage of the direct comparison with the performances obtained on a dedicated parallel system, a Meiko CS-1 including boards with Intel i860's as computing processors and T800 transputers providing the message-passing capabilities and equipped with a tailored software package for parallelization (CSTools). We had the possibility to execute exactly the same parallel code on both architectures, thanks to our parallel network and communication library PNCL, that we have extended to work on both dedicated and general-purpose systems. The efficiency of our application on the Meiko CS-1 is quite high on arrays up to 4 processors and the speedup is close to the number of processors available. This high efficiency is favoured by relatively large data-domains as ours, because the communication times become small relative to the computation times.

The communication overhead is even larger on the workstations arrays, which therefore require a fortiori parallel applications to be decomposed in relatively large subdomains. In other words, for a problem of given domain size the execution on the parallel architecture will be convenient up to a maximum number of processors. For instance, we expect arrays of tens of DECstations (or even Alpha-stations) to work properly on 3-D domains, like $100 \times 100 \times 100 = 10^6$ cells, while 2-D domains could be feasible on arrays of few units.

On the other hand we have experienced that the available software packages for parallel applications on workstation arrays, and PVM in particular, are already as friendly and robust as those provided for dedicated systems, like CSTools. In particular they allow the parallel execution on arrays of non-homogeneous processors, and therefore the allocation of different tasks to different machines. For instance it can be very convenient to run the master program on a relatively slow machine, but with large memory, while the workers, which work on smaller computational domains, can be memory-limited, but faster, machines. On the contrary it is not advisable to use processors with very different performances as workers, since, unless sophisticated and specific systems of load balancing are developed, the execution will be synchronized at the speed of the slowest processor. We notice that there is still a lot of room for improvements of the software tools, especially in terms of reduction of the latency times.

Another factor to be considered to evaluate the feasibility of the usage of LAN's as parallel systems is the level of occupancy of the computational resources. As a general rule, the parallel execution will be worthwhile while the individual workstations are under-used, i.e. they are loaded with no more than one running process at a time. Otherwise, in fact, the system would be already working as a parallel system – although on different and independent tasks – with an efficiency of 100%. The usage of loaded workstations as array processor would be still convenient in this case only for extremely demanding and long tasks, for which we expect some speedup from the parallel execution even with the leak in speed due to the CPU time sharing.

As shown in the previous section, arrays of 6 workstations based on Alpha processors connected in FDDI perform on times not far from those of a supercomputer Cray YMP. This result has been obtained using a general code, with no particular optimization for the specific parallel system, and proves that this approach to massive computations is quite cost-effective, although it requires some intervention on the original sequential code. However, we have shown that, depending on the modularity and on the numerical scheme used, the parallelization may require only few man-hours of programming [3]. On the basis of the above results, the general-purpose system is cost-effective even with respect to the dedicated-system (Meiko CS-1).

4.2 The future

The fundamental parameters which govern the parallel efficiency are the ratio of the computation to the communication time and the size of the computational grid. Based on our tests, we have afforded to extrapolate the performances which can be obtained, expressed in terms of a relation between these two parameters. As shown in the Appendix, in the hypothesis that the size of the messages exchanged between adjacent processors is approximately proportional to the square root of the subdomains (true for approximately square subdomains), with some algebra we have derived from Eq. (4) and (5) an expression which links the size of the subdomains to the system power in terms of the ratio of the time spent for computations to the time spent for message-exchanges:

$$\alpha(X) = \frac{1-\epsilon}{\epsilon}\frac{\sqrt{X}}{k} - \frac{\beta}{\sqrt{X}} \tag{7}$$

where $\alpha = t_{real4}/t_{calc}$, t_{real4} is the time taken to transfer a 32 bits word, $X = N_x N_y / N_{proc}$, the number of cells assigned to each subdomain, k is the number of times that processors exchanges messages within an integration time step, $\beta = t_{lat}/t_{calc}$. The parameters α and β express the message-passing capabilities in terms of network bandwidth and latency times. We can then tune the parameters k and β according to the results of our tests, therefore based on a specific application, a 2-D FCT hydrodynamic code, and use expression (7) to extrapolate our results for future perspectives.

In Fig. 3 we report expression (7) in terms of X as a function of α expressed as the ratio of the processor unit power (in units of DECstation's 5000) to the network bandwidth (in Mbit/s), for two values of the efficiency (80% and 50%) and of β ($\beta = 20$ and $\beta = 2$), and for $k = 128$. The higher β is the one resulting from our tests. The lower value β represents a 10-fold improvement of the latency time, which we may expect for the future. The regions above the threshold lines are regions of higher parallel efficiencies, i.e. performances are better on larger domains. For relatively faster network connections (leftward in the figure), the latency times dominate in the message-passing operations and determine a minimum domain size, independent of the network bandwidth. The higher β determines a larger minimum domain size, because a larger amount of computations is necessary to balance the larger latency time. In the range $1 < \alpha < 10$ for $\beta = 20$ and $0.2 < \alpha < 1$ for $\beta = 2$ the curves begin to increase monotonically: larger subdomains are required, because the fraction of time spent in the data-transfer gradually increases as the network speed decreases with respect to the processor unit speed. In other words the latency times become relatively unimportant, and for lower β this occurs for lower α as expected.

This plot provides a quantitative measure of the requirements for a balanced improvement of the CPU computational power, the network bandwidth and the software quality, which would be appropriate for massive parallel computing on arrays of general-purpose systems. If, for instance, a major effort is devoted to improve the CPU unit power, arrays of such processors will yield significant parallel performance only for very large domains, because of the severe limitations posed by the degraded message-passing capabilities. The lower limit to the domain size will eventually be determined by the latency times, i.e. the software ability to manage the message-passing activity is a crucial factor.

Acknowledgments: We acknowledge partial support from the Italian Ministero dell'Università e della Ricerca Scientifica e Tecnologica, GNA/CNR and Comitato Fisica of CNR. We thank CASPUR, University "La Sapienza", Rome, for the kind availability of the DEC 3000/FDDI LAN.

Appendix: A model of parallel performances

From the definition of speed-up, we have:

$$S = \frac{T_{seq}}{T_{par}} = \frac{T_{seq}}{T_{calc} + T_{comm}} = \frac{N_c t_{calc}}{\frac{N_c t_{calc}}{M} + K t_{comm}} \tag{8}$$

where T's are total times, t's times per unit cell and per unit time step, M the number of processors used, $N_c = N_x \times N_y \times N_{step}$ and $K = 2 \times 32 \times N_{step} \times F_{topol}$ ($F_{topol} = 1$ for 2x1, 2 for 3x1 or above). Moreover, approximately:

$$t_{comm} = \sqrt{\frac{N_x N_y}{M}} t_{real4} + t_{lat} \tag{9}$$

Inserting (9) in (8),

$$S = \frac{N_c t_{calc}}{\frac{N_c}{M} t_{calc} + K(\sqrt{\frac{N_x N_y}{M}} t_{real4} + t_{lat})} \tag{10}$$

Since $\epsilon = S/M$, we can write:

$$\epsilon = \frac{X N_{step} t_{calc}}{X N_{step} t_{calc} + k N_{step}(\sqrt{X} t_{real4} + t_{lat})} = \frac{1}{1 + k(\frac{1}{\sqrt{X}} \frac{t_{real4}}{t_{calc}} + \frac{1}{X} \frac{t_{lat}}{t_{calc}})} \tag{11}$$

where $X = \frac{N_x N_y}{M}$ and $k = K/N_{step} = 2 \times 32 \times F_{topol}$. If we set $\alpha = t_{real4}/t_{calc}$ and $\beta = t_{lat}/t_{calc}$:

$$\alpha(X) = \frac{1-\epsilon}{\epsilon} \frac{\sqrt{X}}{k} - \frac{\beta}{\sqrt{X}} \tag{12}$$

References

[1] G.C. Fox, M.A. Johnson, G.F. Lyzenga, S.W. Otto, J.K. Salmon and D.W. Walker, *Solving Problems On Concurrent Processors* (Prentice Hall, Englewood Cliffs, N. J., 1988).

[2] E.V. Krishnamurthy, *Parallel Processing: Principles and Practice* (Addison-Wesley, Sydney, 1989).

[3] F. Reale, M. Barbera, S. Sciortino, *Comp. Phys. Comm.* 72 (1992) 129.

[4] F. Reale, G. Peres and S. Serio, *Nuovo Cimento B* 105 (1990) 1235.

[5] A. Geist, A. Beguelin, J. Dongarra, W. Jiong, R. Manchek, V. Sunderam: *PVM3 user's guide and reference manual* (Oak Ridge Nat. Lab., Tennessee, 1993).

[6] P. Woodward, and P. Colella, *J. Comp. Phys.* 54 (1984) 115.

[7] J.P. Boris and D.L. Book, *J. Comp. Phys.* 11 (1973) 38.

[8] J.P. Boris, D.L. Book and K. Hain, *J. Comp. Phys.* 18 (1975) 248.

[9] J.P. Boris and D.L. Book, *J. Comp. Phys.* 20 (1976) 397.

[10] S.T. Zalesak, *J. Comp. Phys.* 31 (1979) 335.

[11] J.P. Boris, D.L. Book and S. Zalesak, in: *Finite-Difference Techniques for Vectorized Fluid Dynamics Calculations* (ed. D. Book, Springer-Verlag, New York, 1981) p.29.

[12] D. Kössl and E. Müller, *Astron. Ap.* 206 (1988) 204.

[13] F. Reale, R. Rosner, A. Malagoli, G. Peres and S. Serio, *Mon. Not. R. Astr. Soc.* 251 (1991) 379.

[14] F. Reale, S. Serio and G. Peres, *Ap. J.* (1994) submitted.

Figure captions

Fig. 1 : The Local Area Network of the Osservatorio Astronomico di Palermo. Links on Internet and DECnet/SPAN are also indicated.

Fig. 2 : Execution time of the wedge test on a 372×122 cells grid with the parallel Palermo 2-D FCT hydrodynamic code on various hardware configurations (labelled), as a function of the number of processors in the selected array. Execution times with 100% parallel efficiency should lie on the dashed lines, for the respective configuration.

Fig. 3 : Minimum number of cells per processors to obtain the labelled efficiency as a function of the ratio of the speed processor S_{proc} (in units of the DECstation 5000 performances) to the network bandwidth S_{net} (in Mbit/s). The value of $\beta = t_{lat}/t_{calc}$ labels each curve (dashed curves: $\beta = 2$, solid curves: $\beta = 20$).

Table 1: Execution times of the sequential test code

Computer	Compiler	CPU Time (s)	Speedup (DEC-f77)
DECsystem 5500	f77 -O	10976	1
Intel 386+387 based PC	f2c & gcc -O2	128300	$\sim \frac{1}{12}$
SUN4/370	f77	28997	$\sim \frac{1}{3}$
Meiko/MK096 - 1 i860	pgf77 -O4	6868	1.6
Meiko/MK096 - 1 i860	Greenhills/f77860 -OMAL	7416	1.5
DEC Alpha 3000/400	f77 -s	1941	5.7
Cray YMP8/432 (1 CPU)	cft77	290.8	38
Cray C-90	cf77	108.3	100

Table 1:

Table 2: Parallel performances on Meiko CS (MK096, Greenhills compiler)

N_{proc}	N^x_{proc}	N^y_{proc}	t (sec)	S	ϵ
1	1	1	7416	1	1
2	2	1	3926	1.89	0.94
3	3	1	2697	2.75	0.92
4	4	1	2079	3.57	0.89

Table 2:

Table 3: Parallel performances on workstation arrays on Ethernet LAN

N_{proc}	N^x_{proc}	N^y_{proc}	t (sec)	S	ϵ
1	1	1	10976.8	1.00	1.00
2	2	1	7836.6	1.40	0.70
3	3	1	6456.9	1.70	0.56
4	4	1	5989.6	1.83	0.46
4	2	2	7014.4	1.56	0.39
5 (DEC+ALPHA)	5	1	4265.8	2.57	0.52
5 (DEC+SUN)	5	1	15589.6	1.86(*)	0.37(*)

(*) Speed-up and efficiency has been computed with respect to the Sun 4/370 sequential execution time.

Table 3:

Table 4: Parallel performances on Alpha workstation arrays on FDDI LAN

N_{proc}	N^x_{proc}	N^y_{proc}	t (sec)	S	ϵ
1	1	1	1941	1.00	1.00
2	2	1	1043	1.86	0.93
3	3	1	857	2.26	0.75
4	4	1	755	2.57	0.64
4	2	2	749	2.59	0.65
5	5	1	666	2.91	0.58
6	6	1	589	3.29	0.55
6	3	2	677	2.86	0.48

Table 4:

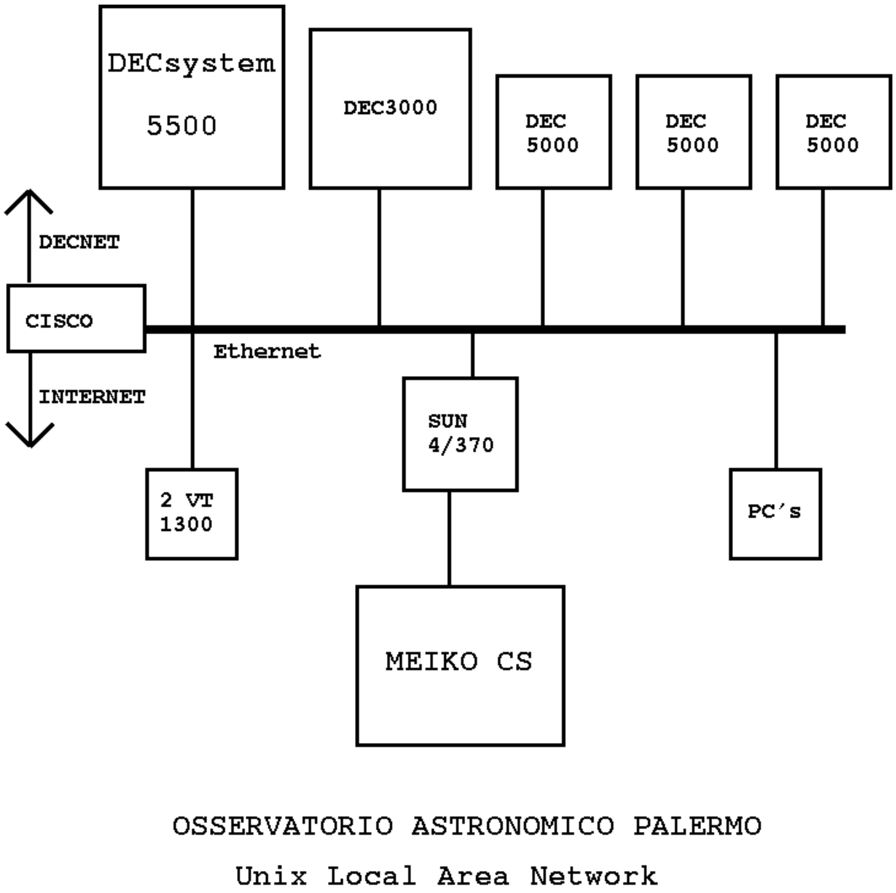
DECsystem
5500
DEC3000
DEC
5000
DEC
5000
DEC
5000
DECNET
CISCO
Ethernet
INTERNET
SUN
4/370
2 VT
1300
PC's
MEIKO CS
OSSERVATORIO ASTRONOMICO PALERMO
Unix Local Area Network

Fig. 1

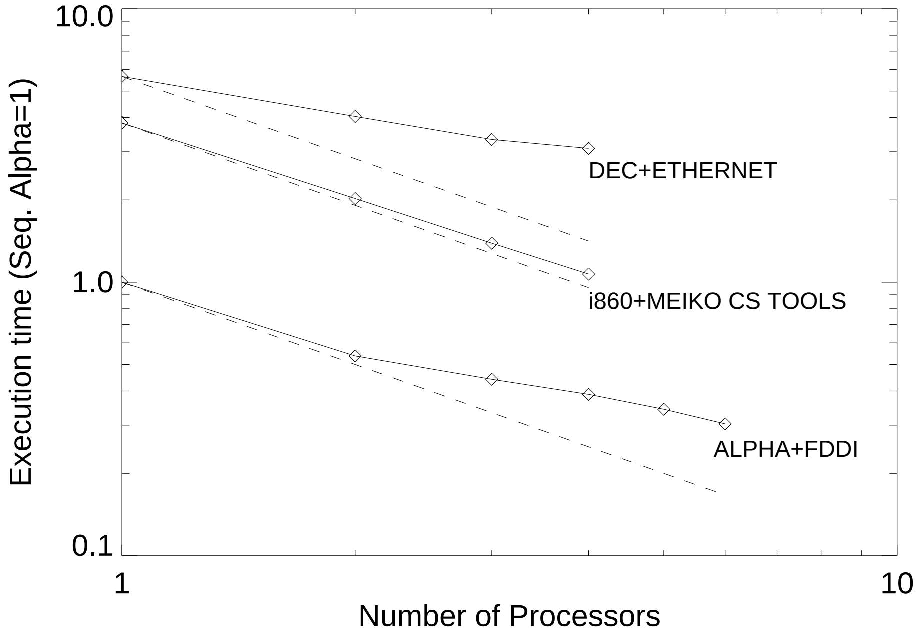
10.0
1.0
0.1
Execution time (Seq. Alpha=1)
DEC+ETHERNET
i860+MEIKO CS TOOLS
ALPHA+FDDI
1
10
Number of Processors

Fig. 2

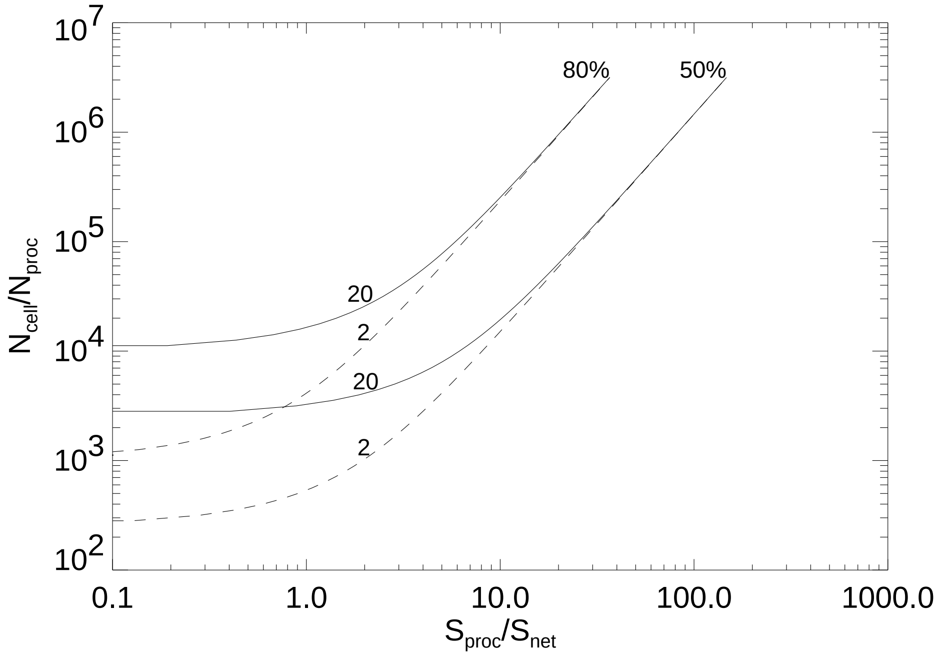
10^7
10^6
10^5
10^4
10^3
10^2
N_{cell}/N_{proc}
80%
50%
20
2
20
2
0.1
1.0
10.0
100.0
1000.0
S_{proc}/S_{net}

Fig. 3