MAX-PLANCK-INSTITUT FÜR PLASMAPHYSIK
GARCHING BEI MÜNCHEN

PVM on Alpha AXP Workstations in a Customer Applications Environment

Mark Ballico and Hermann Lederer

IPP R/45 September 1993

Die nachstehende Arbeit wurde im Rahmen des Vertrages zwischen dem Max-Planck-Institut für Plasmaphysik und der Digital Equipment Corporation über den Einsatz von Alpha AXP Workstations durchgeführt.

Abstract

This paper describes the implementaion of the message passing system PVM ("Parallel Virtual MPachine") from Oak Ridge National Laboratory on a set of three Alpha AXP workstations and an extended heterogeneous workstation cluster, including SUN and IBM RS6000 machines. One Cray YMP processor was also included.

These PVM configurations were tested for a selected computational problem: simulation calculations in plasma fusion research.

Existing Fortran program code was parallelized for the PVM message passing system and prepared for easy automatic execution on the "Parallel Virtual Machine" configured at run time.

The effective computational power obtained from a computer cluster of four machines was found to be almost the sum of the effective CPU powers.

Contents

1. Introduction

1.1 General Aspects

As the present generation of workstations is about to reach supercomputer power, these cheap computing resources appear favourable for migration of badly vectorizable programs from expensive vector machines like Cray and as substitutes of mainframes. DEC Alpha AXP workstations with a 64 bit FPU are promising candidates to keep up with both developments. Similar arguments for better cost effectiveness are, however, used by manufacturers of massively parallel computers.

If we look at problems with inherent parallelism, one class of programs to solve them may be regarded as "loosely coupled" with infrequent communication. This class of programs may be effectively computed in parallel on several workstations using a message passing system. From the machine-independent message passing systems available on the market today, like EXPRESS, PARMACS and PVM, we tested PVM for its usefulness to get effective performance on a Alpha AXP farm as well as from an extended heterogeneous cluster.

1.2 Computing Facilities at Max-Planck-Institute of Plasmaphysics

Most of the intensive computing is done on a four processor Cray YMP vector machine. Development of parallel programs is done on an nCUBE 2 parallel computer with 64 nodes. General batch computing is done on a double processor AMDAHL 5890 and an IBM 3090. Data evaluation of the plasma fusion experiments is done on the mainframes and workstations of various types; several hundred exist on the campus.

2. PVM/HeNCE

PVM is a message passing system that allows to combine al large variety of computer platforms into one "parallel virtual machine" to run related processes in parallel in a distributed computing environment [1]. The project described here was started using version 2.4.2 of PVM. In Feb 1993 a re-designed PVM 3.0 with major changes was introduced [2]. The results presented here are based on PVM 3.1, where a great deal of PVM 3.0 bugs were eliminated. A list of numerous bugs which we have detected in version 3.0 was reported to the authors of PVM.

2.1 Installation of PVM

PVM was installed as suggested by the authors [2]. The PVM libraries libpvm3.a and libfpvm3.a were put in /usr/local/lib, the PVM binaries or shellscripts *pvm, pvmd, pvmd3* and *pvmgetarch* were placed in /usr/local/bin. The individual PVM user created links to these central places from $HOME/pvm3/lib and $HOME/pvm3/lib/ARCH (*pvmd3* only). Executables were placed in $HOME/pvm/bin/ARCH (ARCH stands for architecture and is to be replaced by the concrete architecture used, e.g. ALPHA or RS6K).

2.2 Architectures used

PVM was installed on three ALPHA AXP 3400 systems, one IBM RS6000/580, one IBM RS6000/520, one SUN 4, and a Cray YMP system.

2.3 Problems/Bugs of PVM 3

We installed PVM 3.0 shortly after its availability in Feb 1993. This version contained a lot of bugs which we reported to the authors. For most bugs a workaround could be found. In PVM 3.1 most bugs from the 3.0 version were corrected. The following bug that we reported, however persisted: If a program is to run on three machines, where and only where three PVM daemons are running, the *pvmfspawn* command starts two processes on the same machine, leaving one machine idle, in two out of three trials. Every third trial one process is started correctly per machine, easily to be identified via the process number.

2.4 HeNCE

HeNCE is an X based user interface to PVM [3]. The latest release of HeNCE available at the time of writing this paper was release 1.4. This version already works ("barely ") with PVM 3.x, though, but supports only features of PVM 2.4.2. It does not support any new features of PVM 3.x. HeNCE 1.4 does not use the fault-tolerance facilities of PVM3. Likewise it will fire up only one program at a time. The authors state: "Both PVM 3.x and HeNCE 1.4 code built on PVM 3.x are still somewhat fragile". As these restrictions and handicaps interfere with the aimed tests, we could not use HeNCE as user interface, but had to develop our own strategy to effectively manage PVM and the daemon administration. Our approach will be described in chapter 3.5 (PVM daemon administration)

3. Fusion Research computation

3.1 Background

The tokamak is one promising approach to the development of controlled nuclear fusion which is the energy source powering the sun. In order to attain sufficiently high temperatures in tokamak fusion experiments, it is found necessary to supplement the ohmic heating due to the plasma current. One successful method, used on nearly all large fusion experiments, is the use of radio frequency waves (10-100MHz), which propagate from an antenna mounted on the wall of the machine, to be absorbed in the plasma core. RF heating is used on many installations such as the Joint European Torus, ASDEX, ASDEX-Upgrade [4], Wendelstein 7AS [5] and TEXTOR. The construction of such antennas is expensive. For effective design a proper modelling of the behaviour of these antennas is required. Therefore the current distributions on an assortment of conductors in the presence of a plasma and the resultant power deposition profile must be calculated [6-9].

3.2 Algorithm description

The plasma surface impedance matrix for each toroidal (n) and poloidal (m) mode is calculated by solving the variational form of the plasma wave equations using a finite element method in the radial direction. Each computer in the system computes the surface impedance matrix and partial power deposition profiles for a range of n allocated to it on the basis of effective cpu power as seen by the user. This data is then locally stored on disk. By far the most cpu time is used in this phase of the calculation.

The self-consistent currents are determined by first calcuating the partial impedance matrix of the conductors in the antenna array. This involves solving Maxwells equations for each toroidal and poloidal mode subject to the appropriate boundary conditions. Since the modes are all independent, each cpu can compute for a range of modes, and the partial impedance matrix is then simply globally summed over all cpus. This global sum is the largest data transfer between machines that occurs during the program. The "host" cpu then uses this matrix, together with the source connection matrix, to calculate the currents, this is a very small serial part of the calculation.

The power distribution profile is computed by distributing the now known antenna current distribution to all cpus, which then compute the power distribution for their particular range of toroidal modes, using the locally stored partial power distributions. The total power disribution is then globally summed and stored as an output file.

3.3 Program features and requirements

The Fortran package ported to DEC Alpha AXP and parallelized for PVM consists of 26 subroutines and about 3000 lines of code. Previous porting of this program to nCUBE2 will allow comparisons to performance/scalability on a massively parallel architecture. Therefore a data set was chosen that did not exceed 4 MBytes of memory that could be identically run on the nCUBE 2 computer (with 4 MB local memory per node). NAG library routines were used for generating the required Bessel functions and to solve the band matrix resulting from the finite element representation of the plasma wave equations.

3.4 Program design

For parallel programming the SPMD (*single program multiple data*) model was used. In PVM 3.x this programming model is supported via the *pvm(f)spawn* call. The master copy of the program is started on the PVM master machine. This program then starts remote copies of itself on target machines in the PVM cluster. Figure 1 shows the design of the program in a flow chart for parallel execution.

3.5 PVM daemon administration

The program works on data provided in an input file, as shown in figure 1. This datafile has been extented via a NAMELIST statement at its beginning to declare the hosts the program shall run on. Procedure: First of all, the PVM daemon *pvmd* is started only locally on the master host via the *pvm* console command. Then the PVM program is started ("enrolled") on the master host. This program first reads the PVM hosts to be added to the PVM configuration (*pvm(f)addhost* [2]) and spawns copies of itself on these hosts (*pvm(f)spawn* [2]) (see fig. 1)

3.6 Single source in distributed environment

Source code development was done on one dedicated machine. A shell script was written to distribute this source file on all target machines via the remote copy command (rcp), compile/link this source file remotely (via rsh) and place the executables to the correct place for PVM usage ($HOME/pvm3/bin/ARCH).

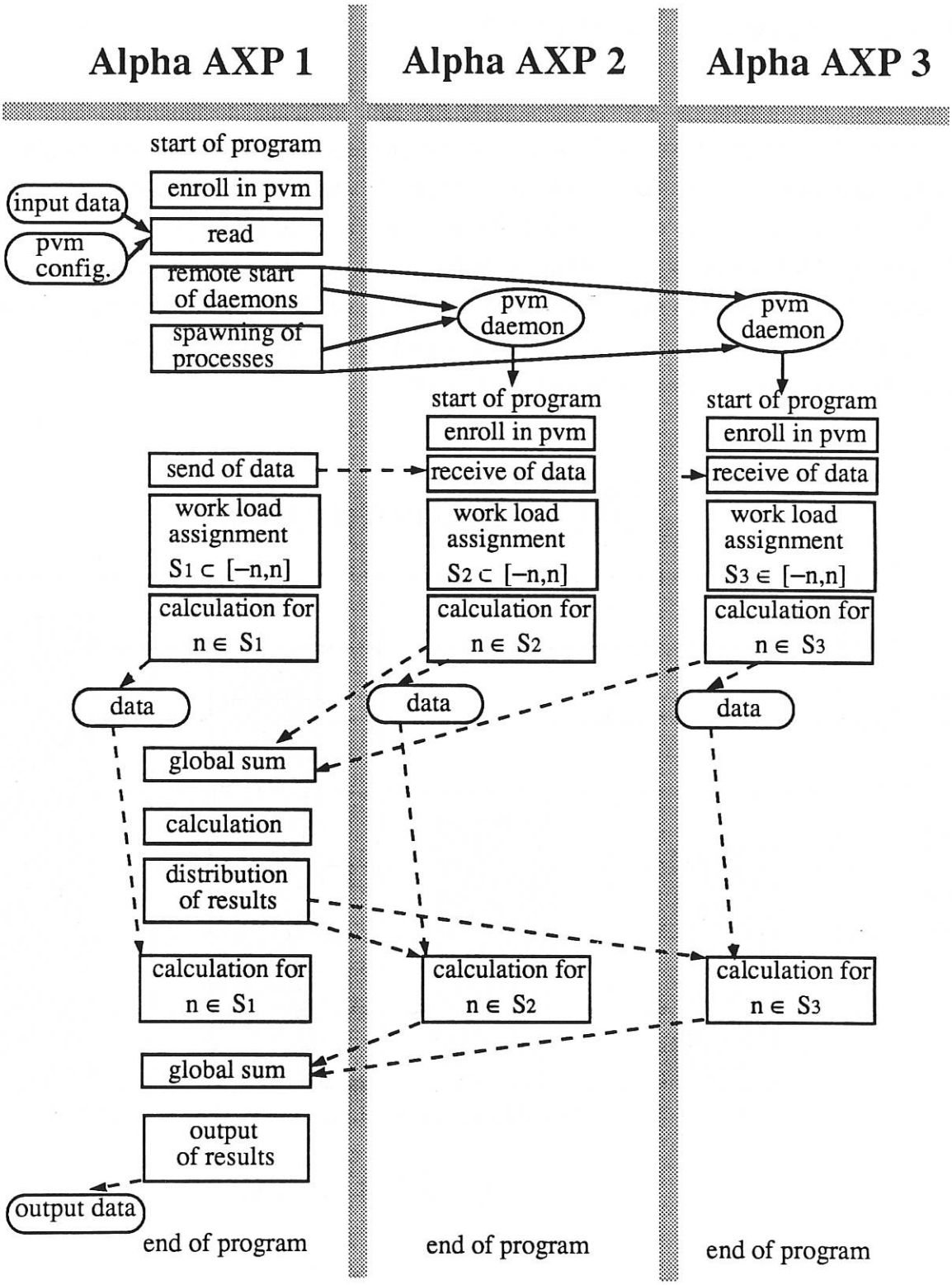

Fig. 1: *Flow diagram of parallel program execution on three Alpha AXP machines*

4. Applications Performance

Application performance was determined using the "time program" command on the PVM master machine and the unix system accounting (accton, acctcom) on the remote machines. System and user time provided by the *time* command were added to form the CPU time which was identical to the value provided by *acctcom* from the local system accounting file (pacct). Measurements were done, however, contrary to the previous description in chapter 3.5, not by starting the remote PVM daemons from the Fortran program as it was designed for comfortable usage, but from within the *pvm* console command in order not to add unnecessary overhead to distributed calculating.

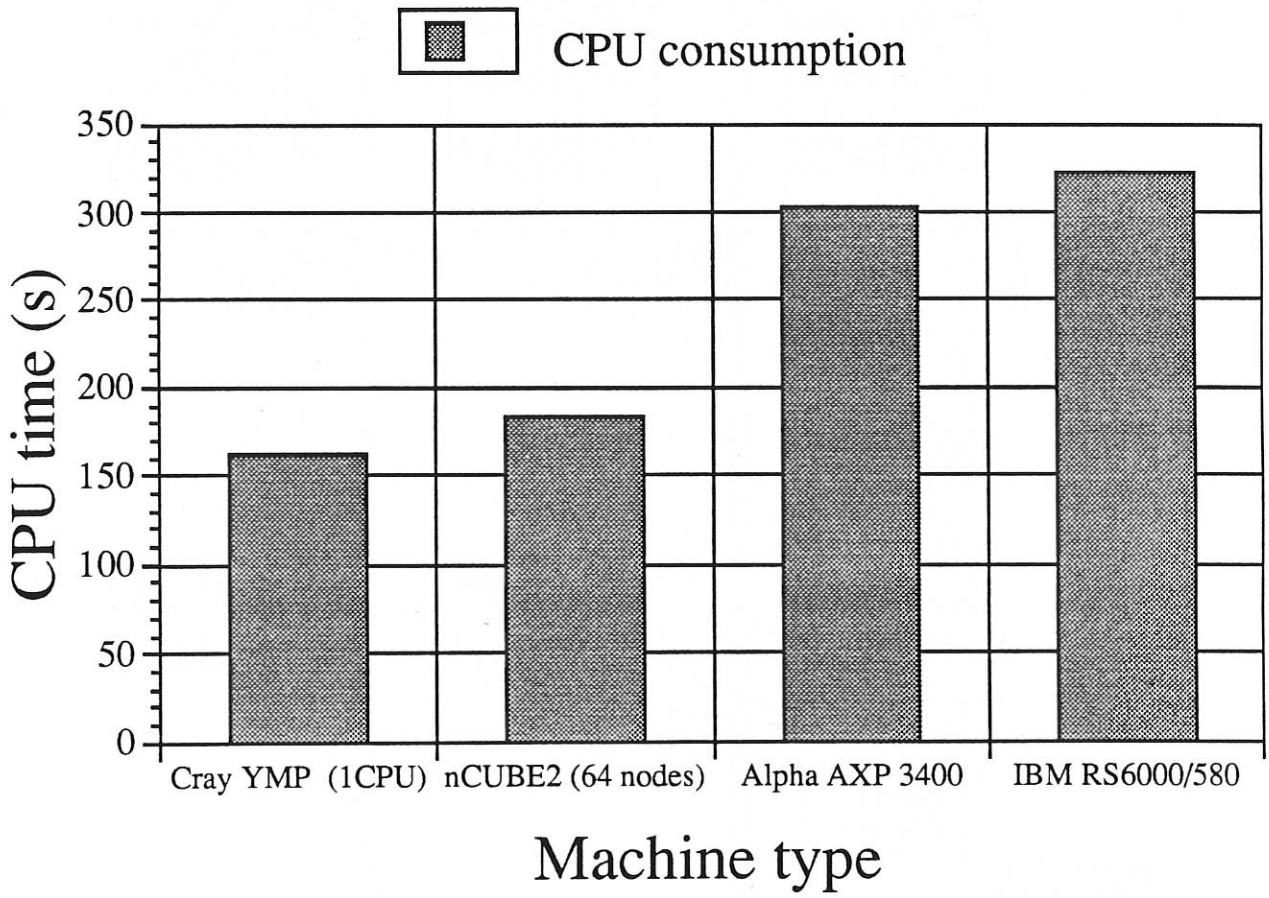

Fig. 2
CPU consumptions of the program PLASMA *on the architectures nCUBE2, Cray YMP (1 CPU) , DEC ALPHA AXP 3400 and IBM RS6000/580*

4.1 Single Machine comparison

The program was executed separately on the following platforms:

- •DEC Alpha AXP 3400
- •IBM RS6000/580
- •Cray YMP
- •IBM RS6000/520
- •nCUBE2
- •SUN 4/330

Results are shown in table 1 and figure 2 (for the four best performances only).

Machine type	Cray YMP (1 CPU)	nCUBE2 (64 nodes)	AXP 3400	RS6000 /580	RS6000 /520	SUN 4/330
CPU (s)	172	183	303	322	3847	6000

Table 1: *CPU consumption for single program execution on different platforms*

For our case the relative performance of 1 AXP 3400 processor is 53% of a Cray YMP processor, 60% of a 64 node nCUBE2 computer and 106% of a RS6000/580 processor.

4.2 Homogeneous cluster (AXP systems only)

Two types of network connections were used for these measurements: Ethernet and FDDI.

4.2.1 Setup for Ethernet and FDDI

Each AXP machine is configured with both an ethernet and an FDDI interface and connected to the campus ethernet and FDDI ring. For performance measurements we had to make sure that only either network type was used exclusively. The following setup was used: Each AXP machine is member of two different internet subnets: 130.183.1.x for ethernet and 130.183.9.x for FDDI. The campus nameserver was reconfigured to keep two different hostnames for each machine: e.g. alpha1 for ethernet access and alpha1f for FDDI access. The local hostnames were then set either to alpha1 or alpha1f (with the *hostname* command) in consistency with the interface to be used.

For program execution we had to make sure that on one machine PVM daemons were not started simultaneously via both network types. Network traffic between the ethernet interfaces was watched from a SUN workstation (using the *etherfind* command). In this way we could exclude mixed network usage.

4.2.2 Performance comparison

The program was executed on the homogeneous AXP cluster with both ethernet and FDDI network connections. Measurements were done with a one-, two- and a three-machine configuration. Results are shown in figure 3.

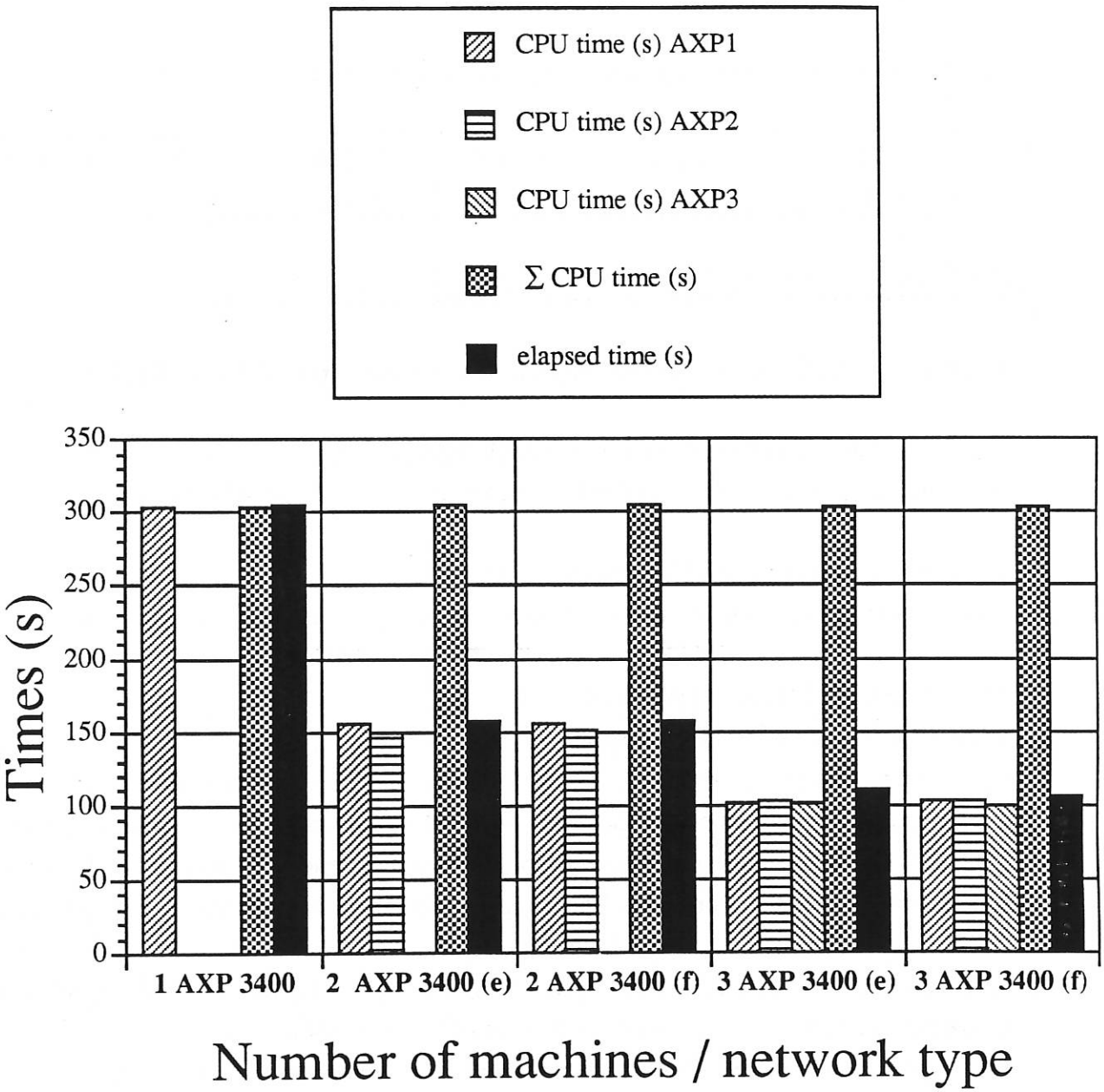

Fig. 3

CPU and elapsed times (s) on one, two and three ALPHA AXP 3400 machines both for ethernet (e) and FDDI (f) connections for program execution in parallel with PVM.

The data from fig. 3 show that the sum of CPU consumptions on several machines exceed the CPU consumption for a single machine run by only less than 1% in all cases. No additional CPU overhead is caused by distributed computing in this case.

Program execution on two machines in parallel leads to a speedup of 1.94 both for ethernet and FDDI. Program execution on three machines in parallel leads to a speedup of 2.78 for ethernet and 2.89 for FDDI, as shown in figure 4.

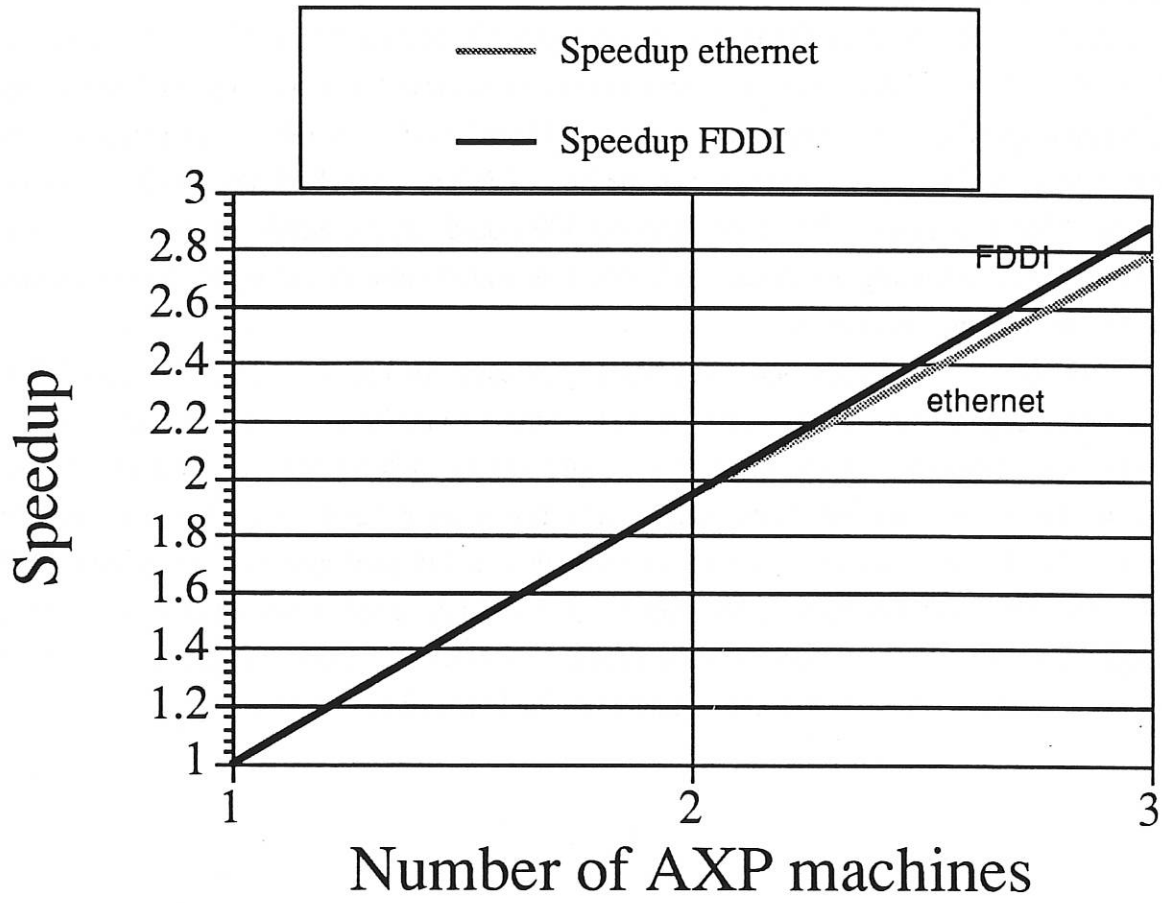

Fig. 4

Speedup of two and three ALPHA AXP 3400 machines compared for ethernet and FDDI connections when executed in parallel with PVM.

Only the use of three (or more) machines allows discrimination between ethernet and FDDI configurations. The increase in communication due to more than two machines benefits from the faster FDDI connection. The speedup gained here is still close to linear.

4.3 Heterogeneous cluster

The exclusively used homogeneous PVM cluster of three Alpha AXP machines was extended to a heterogeneous PVM cluster. By adding machines of different power, architecture or varying load share, one however runs immediately into the problem of load balancing. Appropriate load balancing can be reached by assigning a speed factor to each machine in the PVM pool. This speed factor has to be determined by a single run on each machine type. Varying load of a certain target machine may, however, increase the idle time of other machines.

In a first step the PVM configuration was extended by including the SUN 4/330 and the IBM RS6000/520. Distributed program execution occurred without any problems. The additional speedup compared to that already achieved by three AXP machines was below 10% because of the low performance of the SUN 4/330 and the IBM RS6000/520 relative to the AXP processors. The IBM RS6000/580 could not be added to the three AXP machines due to routing problems to this machine which does not belong to the computing centre and its internet subnet.

As next step we included one Cray YMP processor in the set of three Alpha AXP machines. Significant improvement of performance could be achieved. The Cray speed factor was based on elapsed times as we could not use this machine exclusively. Speed factors based on elapsed times reduce idle times on neighboring hosts, as already mentioned. Four measurements were done in this PVM configuration with three AXP 3400 machines and one Cray YMP processor. The Cray speed factor determined from a single run was varied in steps of 10% relative to AXP speed and the resulting elapsed times measured to demonstrate the influence of load variation, shown in fig. 5.

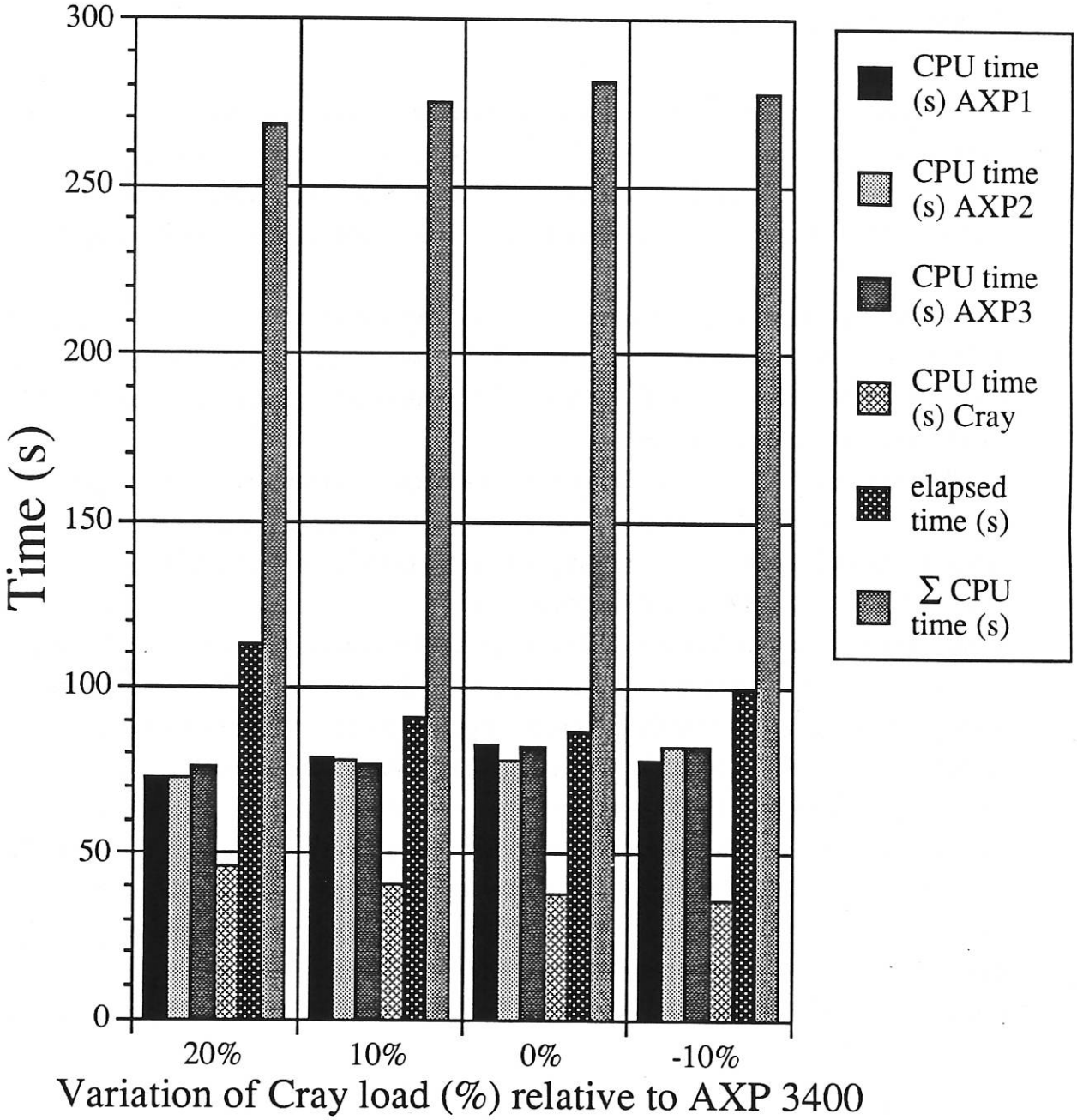

Fig. 5

A "parallel virtual machine" set up of three ALPHA AXP 3400 machines and one Cray YMP processor was timed for parallel program execution with variation of the Cray load. Best performanece (minimal elapsed time) was obtained for the 0% configuration..

5. Discussion

For the type of problem chosen we obtained excellent performance on a DEC Alpha AXP farm of three machines using PVM. The code used is however not well vectorizable and only runs with 60 MFLOPS on a Cray YMP (single processor). Therefore already two Alpha AXP 3400 processors used in parallel could outperform a single Cray YMP processor.

This result may not be generalized, however. For a program with excellent vector performance (e.g. 240 MFLOPS on a Cray YMP), one might expect about eight Alpha AXP 3400 workstations to equal one Cray YMP processor in performance, provided no communication bottleneck occurs.

But this comparison shows a trend: Poorly vectorizable programs should be migrated to fast, but cheap scalar Risc machines. Connecting several fast Risc machines by standard network soft and hardware and a message passing system like PVM transforms the single machines into a parallel usable workstation cluster.

In our case, the communication required for parallel execution was fairly low. It is clear that the amount of communication increases, when the same task (work load) is shared among more processors. Parallel program execution on two AXP machines yielded an identical speedup of 1.94 for both ethernet and FDDI network connections.

Increasing the number of AXP machines from two to three already revealed a slightly reduced scalabilty on ethernet connection contrary to the FDDI connection where the performance still scaled linearly. These differences due to the slower ethernet network and the faster FDDI network presumably increase by using a larger number of hosts. Such a likely bottleneck to maximum performance due to increased communication may be shifted to a larger number of machines by fast interconnect technologies like the GIGAswitch.

As a comparison, on an nCUBE massively parallel computer our program execution performance scaled close to linearly up to a node number of 32. Due to the low performance of a single nCUBE2 processor, however, two Alpha AXP 3400 machines used in parallel could outperform a 64 node nCUBE2.

There is however, one inherent obstacle to parallel computing on a workstation farm: the requirement for exclusive usage. Maximum speedup by parallel execution is not compatible with simultaneous serial batch operation of the same type of machines. And exclusive usage for parallel applications requires a sufficient number of applications that may occupy the disposed CPUs close to 100% in a batch-like mode.

6. References

[1] V.S. Sunderam (1992):*PVM: A Framework for Parallel Distributed Computing*, Dep. of Math and Computer Science, Emory University, Atlanta, GA 30322,

[2] A. Geist, A. Beguelin, J. Dongarra, W. Jiang, R. Manchek, V. Sunderam (1993): *PVM3.0 user's guide and reference manual*, ORNL report ORNL/TM-12187, Oak Ridge Natl. Laboratory, Oak Ridge, Tenessee 37831

[3] A. Beguelin, J. Dongarra, G.A. Geist, R. Manchek, K. Moore, R. Wade, J. Planck, V. Sunderam (1993): *HeNCE: A User's Guide*, Oak Ridge Natl. Laboratory, Oak Ridge, Tenessee 37831

[4] J.-M. Noterdaeme et al.(1993): *Combination of Fundamental and Second harmonic Minority Ion Cyclotron resonance Heating on ASDEX Upgrade*, in: 1993 International Conference on Plasma Physics (in press)

[5] M. Ballico et al. (1991): *Minority Heating Experiments on the W7-AS Stellarator*, AIP Conference Proceedings 244, 150-154

[6] S. Puri (1987): *Antenna Optimization for ALFVÉN Wave Heating,* Nuc. Fus. Vol. 27, No 2., 229-239

[7] G. Cattanei & A.B. Murphy (1991): *Dependence of Heating Efficiency and Impurity Production on the Toroidal Wavelength in Ion Cyclotron Resonance Heating,* Nuc. Fus. Vol. 31, No. 2, 219-232

[8] M. Brambilla & T. Krücken (1988): *Numerical Simulations of Ion Cyclotron Heating of Hot Tokamak Plamas*, Nuc. Fus. Vol. 28, No. 10, 1813-1833

[9] M. Ballico (1991): *ALFVÉN Wave Heating of a Tokamak Plasma,* dissertation, University of Sydney