Performance of the NAS Parallel Benchmarks on PVM Based Networks *

S. White, A. Ålund[†] and V. S. Sunderam
Department of Mathematics and Computer Science
Emory University, Atlanta, GA 30322, USA

Abstract

The NAS parallel benchmarks are a set of applications that embody the key characteristics of typical processing in computational aerodynamics. Five of these, the kernel benchmarks, have been implemented on the PVM system, a software system for network-based concurrent computing, with a view to determining the efficacy of networked environments for high-performance computational aerodynamics applications. We present results of porting and executing the NPB kernels in three different cluster environments using low- to medium-powered workstations on Ethernet and two types of FDDI networks. Our results indicate that mediocre to good performance could be obtained despite the communications intensive nature of the applications. In most cases, we were able to achieve performance levels within an order of magnitude of a Cray Y/MP-1 on 8-workstation clusters via optimizations to the PVM infrastructure alone, i.e. with little or no algorithmic modifications. However, our results also indicate that further improvements are possible, and that network based computing has the potential to be a viable technology for high-performance scientific computing.

1 Introduction

Network-based concurrent computing is receiving increased attention as an effective methodology for high-performance scientific applications in a variety of disciplines. One

*Research supported by the National Aeronautics and Space Administration under grant NAG 2-828, and the National Science Foundation, under Award No. CCR-9118787.

[†]Swedish Institute of Applied Mathematics, Göteborg, Sweden; work done with financial support from NUTEK, Swedish National Board for Industrial and Technical Development, under grant 5321-93-04384, project no 9304384, while visiting Emory University, Fall 1993.

such application area is computational fluid dynamics(CFD). While the mainstay hardware platform for CFD has traditionally been vector supercomputers (e.g. Cray Y/MP), there has recently been an interest in both MPP's (massively parallel processors and hardware multiprocessors) and networked concurrent computing environments. While the communication capabilities of the latter are only a fraction of what MPP's provide, they possess several other significant advantages, including high-availability, excellent price-performance characteristics, heterogeneity, and stable support software and development tools. This mode of parallel computing is currently at a stage where its effectiveness as a suitable technology for high-performance computing is being evaluated.

In this paper, we report on such an exercise – the implementation and execution of a set of CFD benchmark applications on virtual parallel machines using the PVM software system. These applications are the five "kernel" benchmarks from the NAS Parallel Benchmark (NPB) suite [2], an algorithmically specified collection of programs that are representative of real codes (or portions thereof) that are in production use in the aerospace community. The work presented in this paper was undertaken for several reasons. First, the kernels taken collectively, exercise all aspects of the underlying software system (in this case, PVM), especially robustness, portability, message handling limitations, process management, and synchronization. Second, the results of such an exercise would be valuable to the computational aeroscience researchers in evaluating network environments and software systems for their computing needs. Finally, from the point of view of distributed computing research, the kernels highlight critical issues and limitations in the techniques used, and motivate the evolution of new strategies to address them.

2 Background

2.1 The NAS Parallel Benchmarks

The NAS Parallel Benchmarks were devised by the Numerical Aerodynamic Simulation (NAS) Program of the National Air and Space Administration (NASA) for the performance analysis of highly parallel computers. While the NPB suite is rooted in the problems of computational fluid dynamics and computational aerosciences, they are valuable in the evolution of parallel computing, since they are rigorous and as close to "real" applications as may be reasonably expected from a benchmarking suite. The NPB consist of five "kernels" and three simulated applications which "mimic the computation and data movement characteristics of large scale computational fluid dynamics (CFD) applications" [1]. These benchmarks are specified only algorithmically, thereby encouraging optimization and refinement, at the cost of necessitating substantial expertise and effort in porting to a new platform. Complete details of the NPB suite may be found in [1, 2]; for the sake of completeness, we outline the five kernels that were ported to PVM and used in the experiments reported in this paper. In later sections, we provide additional algorithmic details to the extent necessary to understand and interpret our work.

- Kernel EP is to execute 2^{28} iterations of a loop in which a pair of random numbers are generated and tested for whether Gaussian random deviates can be made from them according to a specific scheme. This kernel falls into the category of applications termed "embarrassingly parallel", based on trivial partitionability of the problem, while incurring no data or functional dependencies, and requiring little or no communication between processors. It is included in the NPB suite to establish the reference point for peak performance on a given platform.

- Kernel MG is to execute four iterations of the V-cycle multigrid algorithm to obtain an approximate solution to the discrete Poisson problem $\nabla^2 u = v$ on a $256 \times 256 \times 256$ grid with periodic boundary conditions. This application rigorously exercises both short- and long-distance communication, although the communication patterns are highly structured.

- Kernel CG is to use the power and conjugate gradient methods to approximate the smallest eigenvalue of a symmetric, positive definite, sparse matrix of order 14000 with a random pattern of nonzeros. The communication patterns in this kernel are long-distance and unstructured.

- Kernel FT uses FFT's on a $256 \times 256 \times 128$ complex array to solve a 3-dimensional partial differential equation. Communication patterns in this kernel are structured and long distance in nature, and this benchmark represents the essence of many "spectral" codes or eddy turbulence simulations.

- Kernel IS is to perform 10 rankings of 2^{23} (8388608) integer keys in the range [0, 2^{19} (524288)). The keys are uniformly distributed in the local memories of the parallel machine. Communication in this benchmark is frequent and relatively low-volume, and the pattern of communication is a fully connected graph. This kernel implements a sorting technique that is important in "particle method" codes.

2.2 The PVM System

The typical methodology for parallel or concurrent processing in networked environments is based on a software framework that executes on participating hosts on a network, and emulates a "virtual parallel machine", essentially by implementing various distributed algorithms over standard network protocols. Examples of such software systems are PVM

[4], Express [6], P4 and Parmacs [5], and numerous other systems [7]. PVM *(Parallel Virtual Machine)* is a software system that enables explicit message-passing concurrent computing on networks of heterogeneous machines. Details of the PVM system, the programming model, and experiences with its use have been reported in the literature, e.g. [4, 9, 8, 10]. The basic computing model in PVM views applications as consisting of `components`, each representing a sub-algorithm; each component is an SPMD program, potentially manifested as multiple `instances`, cooperating internally as well as with other component instances via the supported communication and synchronization mechanisms. PVM supports a process-based paradigm and is programmed in a manner akin to other message passing systems. The implementation model in PVM is based on a user-configured "host pool" that comprises the virtual machine, on which *daemon* processes execute and cooperate to emulate a concurrent computing system.

In order to use the PVM system, applications are written as cooperating sequential programs, with embedded library calls to obtain services from the underlying distributed computing infrastructure. Typically, processes are organized in a master-slave or SPMD configuration. This model is very similar to that found in traditional message-passing parallel systems such as the Intel MPP family – except that a more general process structure is provided, and execution platforms may be heterogeneous. In the (common) case of standalone SPMD programs therefore, control and communication structures in a PVM version are usually identical to its message-passing MPP counterpart, and as a result, applications tend to be ported therefrom. However, this influences performance, owing to the general purpose nature of network platforms, inherent differences in communication capacity, and external influences, as will be discussed later.

3 PVM Implementation of the NPB Kernels

In this section we briefly describe the five NPB kernels and their parallelization for PVM. Several iterations were necessary because the release version (including the interface syntax and semantics) of PVM underwent substantial changes during the course of our exercises. The porting process is described in the context of version 3.2 of PVM, whose programming interface is likely to remain stable. We defer discussions of performance to the next section.

3.1 Kernel EP

There are two main portions to this benchmark – the random number generator, and the main processing loop which tests successive pairs of random numbers for the property that Gaussian deviates can be built from them. Parallelization of the problem is straightforward: each process works independently on a subset of the random numbers and the annuli counts are communicated at the end of processing. A front-end PVM process was designed to start the "worker" processes, communicate to them their starting and ending indexes in the loop, and receive their results at the end of computation. A one-time-dynamic scheduling scheme was used, based upon an initial "pilot computation" lasting 10 seconds, to determine the relative CPU capacities of the machines involved.

3.2 Kernel MG

NAS specifies a discrete Poisson problem $\nabla^2 u = v$ with periodic boundary conditions on a $256 \times 256 \times 256$ grid. v is 0 at all coordinates except for 10 specific points which are +1.0, and 10 specific points which are -1.0. The V-cycle multigrid algorithm involves arithmetic on vectors u, v, and a work vector r. A C version, using multigrid recursion, was implemented for PVM. The arithmetic performed are 27-point operations, in which a point, for example, in u is updated by arithmetic on the point in r with the same coordinates *plus* on the 26 points in r which differ by one unit in all combinations of one, two, and three indexes. Therefore any scheme to data partition this problem involves operations at border regions where a processor must "touch" the points which have been assigned to a neighboring processor. Our implementation resolves this by keeping a neighboring processor's border points "in shadow" along with a scheme for updating them before they go out of date. Also, we chose to data partition this problem in only one dimension, such that processor communication fits the ring topology. Figure 1 illustrates the arrangement of processors and the communication pattern for the four processor case.

There is substantial communication in this kernel between neighboring processors, because they must repeatedly communicate the values of vectors at border regions as they change. Message sizes are larger toward the top of the recursion. For example in the first communication, processor p sends the 258×258 (there is periodicity in all dimensions) double precision reals which are the "top" of its slice of the problem to processor $p + 1$, and vice-versa. A similar exchange occurs between the other processors (including 0 and 3 due to periodicity). The total communication, i.e. one of several border communications at the top of the recursion in each of the four multigrid iterations is over 4MB for a four processor set, over 8MB for an eight processor set, etc.

3.3 Kernel CG

Kernel CG is to approximate the smallest eigenvalue of a symmetric positive definite sparse matrix A of order 14000; for accuracy, the `makea` subroutine to generate A was

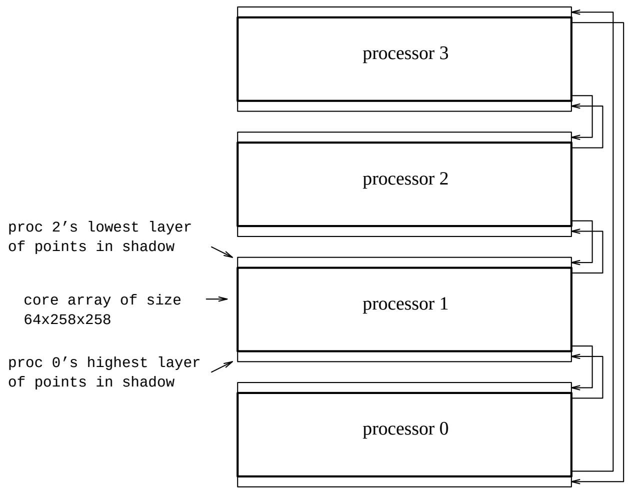

Figure 1: Kernel MG four processor topology and communication pattern

inherited with few modifications from the Intel version — resulting in restricting experiments to n^2 processes. Ten iterations of the power method are employed to approximate the smallest eigenvalue of A; in each iteration of the power method, 25 iterations of the conjugate gradient method are used to solve the linear system $Az = x$. The PVM implementation consisted of rewriting several sections of the code, especially with regard to global communication primitives that are required to be emulated in PVM.

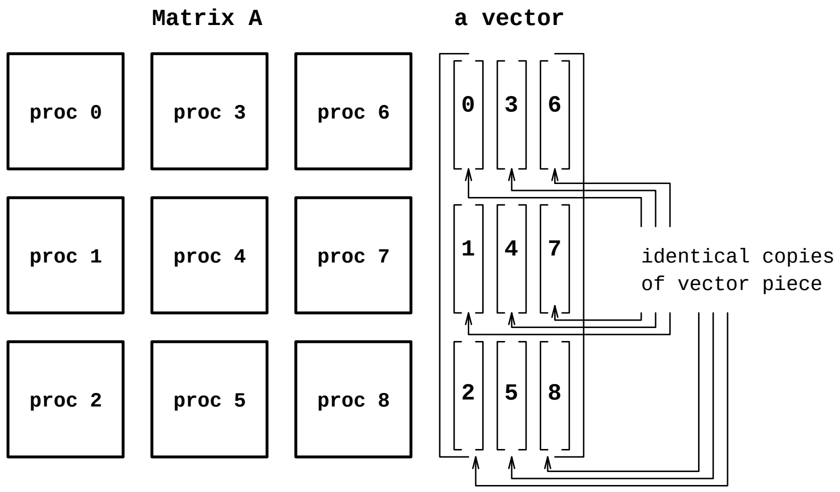

Figure 2: Matrix and vector organization in Kernel CG, nine processor case

The primary design issue in Kernel CG is how to store the various work vectors used in the power and conjugate gradient methods. The operations on these vectors are two dot products, three vector-vector additions, and one matrix-vector multiply for each of the 25 conjugate gradient iterations in each of the 10 power method iterations. The NAS choice is to divide a vector into as many pieces as there are rows of processors, with each processor on a row holding an identical copy of that piece of the vector, as shown in figure 2. Dot products are done in parallel: each processor on a row is assigned computation on a portion $1/P_{row}$ of the vectors in the dot product, with summation done by processor 0. Vector-vector involve a high communicationss-to-computation ratio and are therefore not done in parallel. The scheme for performing the matrix-vector multiplication $q = Ap$ is where efficiency is most crucial. However, as can be seen from figure 2, columnwise multicast of vector pieces is required before the matrix-multiplication can take place. After the matrix-vector multiply by subblocks, processors on a column must sum their resultant vectors. However, if this were done, processors on a row would no longer have identical copies of that row's assigned portion of q. The NAS algorithm takes advantage of the symmetry of A and sums the resultant vector along the row instead. The distribution of the vector p and the summing of the resultant vector q add up (in the 10×25 matrix-vector multiplications) to a total 130MB of communication in the four processor case, 250MB in the nine processor case, and 370MB in the 16 processor case.

3.4 Kernel IS

This benchmark performs 10 distributed rankings of 2^{23} (8388608) integer keys in the range [0, 2^{19} (524288)) that are uniformly distributed among the processing elements. The Intel version of this benchmark was used to derive a PVM implementation, even though the work allocation assumes identically powered processors – incorrect in heterogeneous networks. The strategy for performing the distributed ranking is illustrated using a four processor example, with data structure names indicated in Figure 3. Each processor begins with 2M ($M = 2^{20}$) keys in the range [0, 1/2M) in the `keys` array. The goal is to fill in the `ranks` array with the global rank of each key. First, each processor calculates its distribution of key values by incrementing `distribution[i]` for each occurrence of key value `i` in the `keys` array. Each processor sends its `distribution` array to the root processor, which sums these local distributions to calculate a global distribution of key values, and assigns a range of key values—as balanced as possible by number of keys having values in that range—to each processor. Each processor then enters a loop where for every other processor, it packs a buffer of its keys that are assigned to that processor (`packed_keys`), packs each key's index in its sequence of keys (`packed_indexes`), and sends those arrays to the processor assigned to rank on that range.

Each processor receives incoming keys and indexes into the arrays `keys_to_rank` and `local_indexes`. The number of keys per processor is then broadcast, so that each processor knows the number of keys in each range of key values, and therefore where to begin

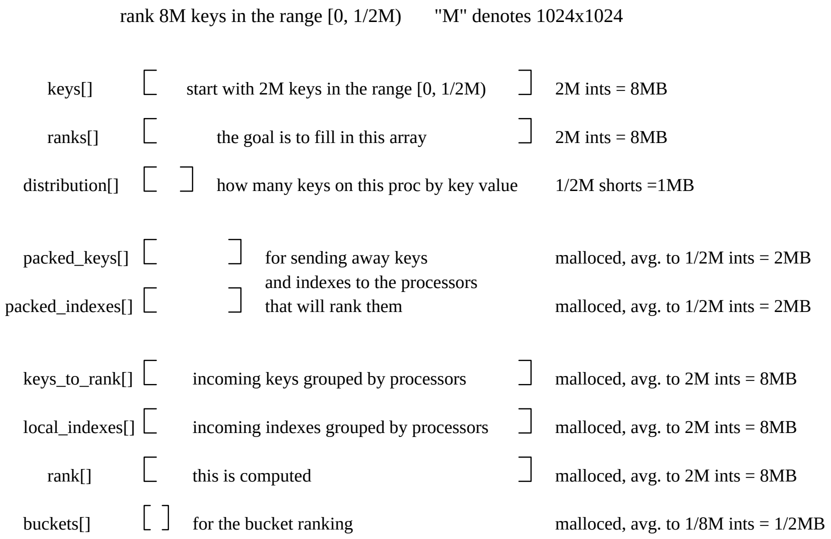

Figure 3: Kernel IS data structures, four processor example

assigning rank numbers. Each processor ranks approximately the same number of keys, give or take a small number due to a non-perfect breaking points in the distribution. A simple bucket rank is used to rank the keys, using the `bucket` array. On conclusion of the bucket ranking, each processor communicates the appropriate portions of the `rank` and `local_indexes` array to the native processors. The communication required in this distributed ranking algorithm is extreme. Each processor sends key values and indexes for approximately $(N_{procs} - 1)/N_{procs}$ of its keys, and receives that many ranks and indexes back after the distributed ranking. It is expected that the computation-intensive code executes in very much less time than the communications code, to the extent that on a distributed memory machine this kernel essentially reduces to a communications-system benchmark. In an attempt to reduce the communications volume in this application, we also implemented a version of Kernel IS that avoids sending indexes as the expense of additional memory and processing.

3.5 Kernel FT

The Kernel FT solves the discrete form of the PDE

$$\frac{\partial u(x,t)}{\partial t} = \alpha \nabla^2 u(x,t), x \in R^3, u \in C \text{ with initial values } u(x,0) = u_0(x)$$

using the Discrete Fourier Transform. After 3-D Fourier transformation of each side, this equation becomes

$$\frac{\partial v(z,t)}{\partial t} = -4\alpha\pi^2|z|^2 v(z,t) \text{ with the initial solution } v(z,0) = v_0(z)$$

where $v_0(z)$ is the 3-D Fourier transform of $u_0(x)$. The solution to this equation is

$$v(z,t) = e^{-4\alpha\pi^2|z|^2 t} v(z,0) = e^{-4\alpha\pi^2|z|^2 t} v_0(z)$$

where $v(x,t)$ is obtained as the inverse 3-D Fourier transform of v. The above is also true for the discrete form of the original PDE when the 3-D Discrete Fourier Transform (DFT) is applied. The serial version of Kernel FT first generates an initial $n_1 \times n_2 \times n_3$ array of complex numbers, $U(j,k,l), 0 \leq j < n_1, 0 \leq k < n_2, 0 \leq l < n_3$ initialized with values from a pseudorandom number generator. Then 3-D DFT of U, called V, is computed using a 3-D FFT. Finally, for each $t \in [1,6]$ $W_{j,k,l}(t) = e^{-4\pi^2\alpha|\overline{j}^2+\overline{k}^2+\overline{l}^2|t} V_{j,k,l}$ is computed, where $\overline{j}$ is defined as j for $0 \leq j < n_1/2$ and $j - n_1$ for $n_1/2 \leq j < n_1$ and $\overline{k}$ and $\overline{l}$ similarly defined with n_2 and n_3. The inverse DFT is then applied to obtain the solution array $X_{j,k,l}$.

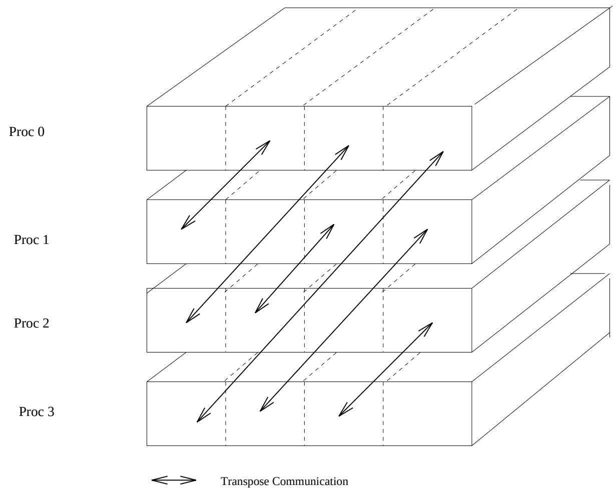

Figure 4: Kernel FT partitioning/communication, four processor example

The PVM implementation on n_p processors organizes the $n_1 \times n_2 \times n_3$ array as n_3 $n_1 \times n_2$ dimensional arrays, with each processor responsible for n_3/n_p of these arrays. The 3-D DFT is implemented as 3 sweeps with a 1-D FFT routine, one in each of the directions j, k, and l and the results are multiplied to form the 3-D DFT. The sweeps in the j and k directions can be done on the data initially assigned to each processor but for the sweep in the l-direction the array must first be transposed, involving communication as shown in Figure 4. The PVM implementation of the 3-D DFT requires two array transposes; therefore, the total communication volume is $2n_1n_2n_3(n_p - 1)/n_p$ complex numbers sent and received. For the $256 \times 256 \times 128$ problem on 8 nodes the communication volume for one 3-D DFT is 224 MByte (28 MByte / node), totalling 1568 MB for 6 timesteps. The memory requirement for this problem size is 49 MB for each of the 8 processes.

4 Preliminary Experiments

In the previous section, we have described the parallelization aspects of the five NPB kernels and outlined the structure of the PVM implementation of each. In this section we present preliminary results of our performance experiments with these five kernels in three different PVM environments. The release version (3.2) of the PVM software, was used, and the default "daemon-based" communication scheme was selected. In the next section we describe a significant performance enhancement to the PVM communications scheme[1], and present updated results. We wish to emphasize that the results reported herein are for *unoptimized* implementations of the NPB kernels – neither processor/memory optimizations for the specific machines involved nor partitioning and scheduling optimizations believed to be essential for efficient heterogeneous execution were incorporated. A few changes that do have an impact on performance were made, and these are described later, but by and large, the structure of the PVM implementations are based upon the corresponding Intel versions distributed by NAS.

4.1 Testbed Environments

The benchmark exercises discussed in this paper are unconventional, in the sense that they are not aimed at measuring performance of a specific machine, but rather of a methodology, *viz* network computing. In practical terms, this measures the effectiveness of the software infrastructure (PVM) and the particular network environment on which the experiments are conducted. In order to obtain an understanding of the effect of different hardware platforms while utilizing the same software system, we conducted our benchmark experiments on three environments, each with its unique characteristics. While these were selected based on availability, they do nevertheless represent major classes of cluster platforms, especially with respect to the communications network.

Low-end workstations on low-speed shared medium networks: This environment consists of 16 diskless Sun SS1+ workstations, each with 16 MB of memory and 32 MB of virtual

[1]Since the time of this exercise, the release version of PVM (3.2.6) has also incorporated a fast communication scheme.

memory(swap space), at the Emory University Parallel Processing Lab. They are isolated from the remainder of the department/campus network, and it is possible to use them in dedicated mode – thus achieving a reasonably well controlled environment, where the only external influences are background OS and network activities. These machines are connected by 10Mbps Ethernet on which the maximum achievable throughput is about 1000 KB/s, and the minimal latency (process-to-process time, including PVM overhead, for a zero length message) is of the order of 2 milliseconds.

Medium-power workstations on high-speed shared networks: This environment consists of seven IBM RS/6000 model 560 computers, and one RS/6000 model 320 computer, each with 32 MB of memory, and 64 MB of swap space, connected by FDDI. FDDI is an optical token-ring network that theoretically offers 100 Mbps, but because it is a shared-ring topology, achievable throughput is usually substantially lower. Minimal latencies are of the order of 1 millisecond with IBM RS/6000 model 560's. This platform was made available to us by the Utah Supercomputing Institute, and was subject to external loads.

Medium-power workstations on high-speed switched medium networks: This platform consisted of SGI R4000 workstations interconnected by FDDI Gigaswitch. The workstations are configured with 64 MB of memory and 128 MB of swap space; minimal message latency is about 1 millisecond. The switch is rated at 100 Mbps across 11 simultaneous connections. While this testbed was not available in dedicated mode, it was possible to access it at times of low external loads.

4.2 Performance Results

In this subsection we present consolidated performance results for the five kernel benchmarks with brief discussions; detailed analyses are deferred to the section on improved results. Table 1 shows the the total *elapsed time* in seconds for each platform, the total communication volume, and the time for communication related activities alone for each benchmark on the three testbed platforms, using an unmodified version of PVM version 3.2. Also shown for comparison are the reported times on a single processor of the Cray Y-MP, and a 128-node Intel iPSC/860 hypercube. These latter results were obtained from a recent NAS report [3], and represent the best, optimized, times available as of the time of writing of that report. We emphasize again that the PVM results are for unoptimized versions, except for a few gross changes described in the previous section. In all cases, native Fortran and/or C compilers were used with only the "-O2" optimization flag. The experiments were conducted during idle periods, with background CPU and network loads estimated at 5—10%, although the environments were not controlled.

The first row of table 1 shows the performance of the EP benchmark. The PVM version used an initial 10-second pilot computation to determine processor capacities

Bench-mark	16 SS1+ Enet			8 RS6000 FDDI			8 SGI Gswitch			Cray Y/MP-1	i860 128
	Time (secs)	Com. (MB)	Vol. time	Time (secs)	Vol. (MB)	Com. time	Time (secs)	Vol. (MB)	Com. time	Time (secs)	Time (secs)
EP	1603	NA	NA	342	NA	NA	446	NA	NA	126	26
MG	198[1]	96	154	229	192	162	264	192	112	22	8.6
CG	701	370	480	285*	130	192	130+	250	101	12	8.6
IS	607[2]	150	595	674	560	610	770	560	720	17	14
FT	717[1]	420	502	645	1500	395	1070	1500	516	29	10

[1] Reduced problem size ($128 \times 128 \times 128$); [2] Reduced problem size (2^{21} keys, range 0 to 2^{19})

* 4 RS 6000 processors; + 9 SGI R4000 processors

Table 1: NPB Kernels on unmodified PVM

on the basis of which workload is allocated. While these results are as expected, two facts are noteworthy. The first is that in the absence of communication, clusters of eight workstations are capable of delivering a third of Cray Y-MP/1 performance. The other point (not explicit in the table), is that despite a dedicated environment and and the use of a "one-time dynamic" load-balancing scheme, load imbalances of the order of 10–15% were observed in all PVM environments. As far as we were aware, the environments were "dedicated"; therefore, background operating system activity and the *nice* policy are the only explanations for this behavior.

The results shown for the MG kernel are rather disappointing, indicating that 8-processor clusters perform more than an order of magnitude slower than a single processor of the Cray Y-MP, and 25 times slower than a 128-processor Intel hypercube. This is in spite of an optimization to change the order of message passing to permit greater overlap of computation and communication — that resulted in performance improvement of the order of only 6-9%. On the SS1+ cluster, owing to memory limitations, we were only able to execute a reduced version of MG that uses a $128 \times 128 \times 128$ grid. The table entry for MG also indicates that communication time accounts for upto two-thirds of the execution time on PVM platforms.

The parallel CG algorithm results in increased communication volume with an increase in the number of processors - a serious drawback especially for shared-network based concurrent computing. However, a small number of powerful workstations works well, as the timings for the 4-processor FDDI and 9-processor switched networks indicate in row 3 of Table 1. Once again, communication times dominate the overall execution times, accounting for 77% of the time in the 9-processor switched network case, and 68%

in the other two platforms.

The IS benchmark is particularly ill-suited for networked environments, involving large communication volume and frequency, and minimal computation. As a result, IS performs poorly in all three PVM environments when using the daemon-based communication scheme. In fact, communication time comprises over 90% of the overall execution time in all cases! Results are shown for the version of IS incorporating one modification, *viz.* avoiding the exchange of array indices at the expense of relatively small extra memory and processing.

The FT kernel is most demanding in terms of communication volume. However, this communication is in the form of large messages of relatively low frequency – a characteristic that contributes to achieving high bandwidth. The parallel PVM version of FT was derived directly from the Intel version, with some portions obtained from the sequential NAS version. Results are shown in the last row of Table 1 – but exhibit rather disappointing performance, possibly indicating that even bandwidth-limited applications are unable to perform well under the unmodified PVM system.

4.3 General Comments

The results presented in the previous subsection are only mediocre at best. However, notwithstanding the general reasons of communication bottlenecks, PVM overheads, and timeshared machines for this poor showing, we should still expect to see somewhat better performance. Although too simplistic, we may use some crude approximations to determine the extent of the performance deterioration due to communication in the following manner. For example, in the MG benchmark, 192 MB of data are communicated in 162 seconds on shared FDDI and in 112 seconds on switched FDDI. These measurements suggest that an average throughput of 1.2 and 1.7 MB/sec on the two networks respectively – 10% and 15% of the rated capacities. Even accounting for the fact that rated network capacities are hard to achieve, this appears to be abysmally low.

5 Optimized Message Passing

Preliminary benchmarking of the NAS kernels on PVM, as described in the previous section, suggested that only a fraction of available network capacity was being attained, thereby resulting in poor to mediocre performance of these benchmarks in cluster environments. As this effort is aimed at evaluating and enhancing the software infrastructure to obtain better performance (as opposed to algorithm modifications or compiler optimizations), we focused our efforts on techniques to improve PVM communication performance. Message passing in the PVM system is a three-stage process by default, involving "store-and-forward" by the daemons. This scheme has its advantages in terms of scalability and statelessness, but performs poorly. In an attempt to overcome this drawback, we have designed and implemented an enhanced communication scheme for PVM; in this section, we describe this mechanism and present updated performance results[2].

5.1 Fsend and Frecv

From the programming interface point of view, the enhanced message passing scheme, accessible via the `pvm_fsend()` and `pvm_frecv()` calls, permit the direct transfer of user program data without requiring buffer initialization and packing. The `pvm_fsend()` and `pvm_frecv()` mechanisms are built on standard TCP internet protocols and are manifested as a separate and non-intrusive library in the PVM system. The `pvm_fsend()` and `pvm_frecv()` library was implemented and tested on a variety of environments and networks. Table 2 indicates the performance of this communication scheme for simple point-to-point data transfer, for a variety of message sizes, for each environment discussed in this paper. Also shown for reference, are the corresponding values for dameon-based PVM communication, and for a standalone benchmarking program, *viz.* TTCP.

From the table it can be observed that the enhanced communication scheme delivers throughputs several times as much as the daemon based communication. However, it also indicates that except for large messages, even the enhanced communication mechanism delivers only a fraction of the throughput actually attainable by software as indicated by the reference TTCP numbers which, incidentally, are of the order of 70-95% of the theoretical maxima. This exercise of evaluating communications performance and the development of an experimental enhanced communication scheme was productive for two important reasons. First, it has resulted in a robust and effective auxiliary library that performs, in some cases, several times better than the native PVM system. Second, it has highlighted the importance and benefits of tuning communications software both for specific machines and their operating systems, as well as for different types of local area networks.

[2]As mentioned, a later version of PVM (3.2.6) supports similar functionality and performance via a toggled option. Thus the results presented are also likely to be obtained using the release version of the PVM software.

Platform	Throughput (kB/sec)			
Msg. size→	1 byte	100 bytes	10kB	1MB
SS1+ Ethernet				
Daemon	0.06	12.88	263.41	358.48
Fsend	0.49	81.79	902.42	1003.87
TTCP	0.65	130.45	965.04	1125.24
RS6000 FDDI				
Daemon	0.09	20.67	374.04	711.58
Fsend	0.82	112.79	1568.40	2285.89
TTCP	1.85	325.40	2573.00	2918.20
SGI Gigaswitch				
Daemon	0.21	38.92	406.04	483.58
Fsend	1.11	102.79	3400.42	9550.87
TTCP	2.15	500.40	9203.00	9624.20

Table 2: Point-to-point communication bandwidth in PVM

5.2 Updated Performance Results

In this section we report on performance measurements obtained by re-executing the NPB kernels on PVM, using the enhanced communications scheme[3]. In the tables, we also include the total time and communication time for the previous experiments (in parentheses) for convenient comparison. We also include the number of messages, and an additional measure, *viz* the "maximum idle time". The time that each process was blocked on receives, i.e. waiting for incoming messages that were a precondition to performing further computation, was measured and are reported for the process whose value is maximal. This metric approximately reflects the load imbalance in a given application algorithm or the parallelization thereof.

It may be seen from table 3 that the enhanced communication mechanism resulted in an improvement of 50% for the SS1+ platform, 100% for the RS6000's, and 60% for the SGI's. In two environments, all the gains were due to communication time, which was reduced by 50% to 300%. Focusing on the idle time column, it may be seen that unproductive time spent in blocked receives are a significant fraction of the total communication times, indicating that further optimizations may be possible by implementing a better load balancing scheme, or by eliminating serial bottlenecks.

The improvements in performance for Kernel CG obtained by using enhanced PVM

[3]EP results are unchanged and are therefore not reported

Platform	Time (secs)	Comm. Volume	Comm. Time(secs)	Number of msgs	Idle Time(secs)
16 SS1+ Enet	138* (198*)	96 MB	85 (154)	2704	48
8 RS6000 FDDI	110 (229)	192 MB	52 (162)	1808	30
8 SGI Gswitch	168 (264)	192 MB	81 (112)	1808	50
Cray Y-MP/1: 22 secs; i860/128 : 8.6 secs					

* Reduced problem size ($128 \times 128 \times 128$)

Table 3: Kernel MG on enhanced PVM

Platform	Time (secs)	Comm. Volume	Comm. Time(secs)	Number of msgs	Idle Time(secs)
16 SS1+ Enet	605 (701)	370 MB	404 (480)	37920	390
4 RS6000 FDDI	203 (285)	130 MB	101 (192)	7116	88
9 SGI Gswitch	108 (130)	250 MB	46 (101)	19756	42
Cray Y-MP/1: 12 secs; i860/128: 8.6 secs					

Table 4: Kernel CG on enhanced PVM

communications are not as dramatic as in MG for the SS1+ platform, and of the order of 35% for the other two platforms. However, communication times in the two FDDI environments are reduced by a factor of two, and as the idle time column shows, potential for further reduction exists. Another interesting observation concerns the number of messages transmitted in each case in relation to the total communication volume. For example, in the 9 processor SGI platform, CG exchanges 19756 messages for a total volume of 250 MB, while MG exchanges only 1808 messages for 192 MB. The enhanced communication mechanism attempts to optimize performance for small messages as well, thereby achieving performance gains even for latency-limited applications. Yet, idle times of the order of 60% to 90% of total communications persist.

Kernel IS shows a uniform twofold (or better) performance with the enhanced communication scheme, as compared to the native PVM measurements. This improvement is definitely attributable to increased communication efficiency, especially in the case of the two FDDI networks. On Ethernet, the difference is not so marked, perhaps owing to the fact that collisions (as observed by network monitoring tools) on the network continue to cause degradation. Once again, idle times are exremely high.

The results for FT are also illustrative in that they indicate the degree to which per-

Platform	Time (secs)	Comm. Volume	Comm. Time(secs)	Number of msgs	Idle Time(secs)
16 SS1+ Enet	398* (607*)	150 MB	375 (595)	10115	370
8 RS6000 FDDI	318 (674)	560 MB	171 (610)	2491	129
8 SGI Gswitch	258 (770)	560 MB	140 (720)	2491	105
Cray Y-MP/1: 11 secs; i860/128: 14 secs					

* Reduced problem size (2^{21} keys in the range 0 to 2^{19})

Table 5: Kernel IS on enhanced PVM

Platform	Time (secs)	Comm. Volume	Comm. Time(secs)	Number of msgs	Idle Time(secs)
16 SS1+ Enet	520* (717*)	420 MB	385 (502)	3450	266
8 RS6000 FDDI	412 (645)	1500 MB	220 (395)	826	112
8 SGI Gswitch	228 (1070)	1500 MB	130 (516)	826	34
Cray Y-MP/1: 29; i860/128: 10 secs					

* Reduced problem size ($128 \times 128 \times 128$)

Table 6: Kernel FT on enhanced PVM

formance can be improved by using faster communication schemes for very-high message-volume applications. Execution time reductions are most dramatic for this benchmark, especially in the case of the switched FDDI network, where a four-fold improvement in both overall time and communication time may be observed. This marked improvement in performance and high network utilization is attributable to the fact that infrequent, large messages are the norm in this application. This effect is highly pronounced in a switched network but not as remarkable in shared networks; understandable considering that the FT benchmark is synchronous in nature – all processes perform approximately equal communication and then simultaneously exchange large messages. Another effect of exchanging large messages at relatively infrequent intervals is that idle times attributable to blocked receives and load imbalances are not as severe.

5.3 Communications Efficiency

From the foregoing results, it can be seen that the enhanced communication scheme reduced communication times significantly, often achieving near-maximal communications efficiency. For example, in MG, the SS1+ system transfers 96MB in 85 seconds; an aggregate rate of 1.12MB/sec, or 95% of the theoretical maximum. The Gigaswitch version of CG transfers 250MB in 46 secs for medium size messages, also close to the best attainable in software. IS on Ethernet runs at 91% communication efficiency. FT on FDDI gets over 60aggregate utilization (twice the software max for a point-point raw test) due to high offered load on a shared-access network. However, there is still room for improvement, as discussed in the next section.

6 Discussion

In this paper, we have described porting exercises and performance measurements of the NAS parallel benchmark kernels to distributed computing environments based on the PVM system. This work has shed light on a number of issues that influence high-performance concurrent computing in networked environments. Many of these issues deal with communications performance. By analyzing data transfer bottlenecks in the release version of PVM and devising enhancements to the system to overcome these drawbacks, we have obtained significantly better performance in all three network platforms *viz* shared-Ethernet, shared-FDDI, and switched FDDI. In addition, these experiments have also highlighted other factors that should be addressed in order to obtain optimal performance, especially load balancing, and we intend to investigate strategies to address them in future research.

6.1 NPB Codes on PVM

From the results presented in this paper, it may be seen that the clusters we have used are generally an order of magnitude slower than the standard yardstick for CFD codes, namely one processor of the Cray Y-MP. Eight processor clusters of RS6000's over FDDI and SGI's over switched FDDI perform about 7 to 10 times slower than a Cray Y-MP/1, with better ratios when communication volume and frequency is lower. This is to be expected of course, but certain other aspects are also worthy of note. First, it is apparent that for codes such as the NPB kernels that exchange large volumes of data, fast networks are essential for good performance. However, shared-FDDI networks perform as well as switched networks when the communication volume is below a certain threshold, as indicated by the superior performance of kernel MG in FDDI ring networks [4]. When communication volume is large, as in the case of FT, switched fiber networks appear to perform twice as well for the same number of processors.

[4] some of this better performance is of course due to the faster processors used

It should also be kept in mind that our implementations were naive, first-cut efforts. Being based on the Intel versions, they use uniform partitioning and workload allocation and assume a dedicated interconnection network of specific topology — neither of which is appropriate for cluster settings. We believe that a "first-principles" implementation of the NPB kernels for PVM, with specific tailoring of the computational algorithms as well as partioning and communication schemes to the cluster processors, the network, and the PVM system, will achieve significantly better results. Further, both the RS6000 network and the SGI platform were subject to external influences and various configuration and operating parameters (e.g. memory, swap space, spontaneous OS activity, *nice* policy, network service daemons etc) that we had no control over. In some sense however, such variability and lack of control is a *feature* of the network computing methodology, not a drawback. Therefore dynamic load balancing schemes and better partitioning methods are probably required to exploit the full potential of network computing.

6.2 Future Work

As can be seen from the foregoing discussions, several issues still remain to be addressed with regard to network computing infrastructures to improve their efficacy for high-performance computing. We intend to continue to work on the following critical aspects as part of our research in heterogeneous concurrent computing.

- Profiling and Visualization: In order to gain deeper insight into the behavior of the applications and software framework, and to identify the primary bottlenecks, we are currently working on enhancements to the PVM system to permit visualization of network applications using the ParaGraph tool [11].
- Partitioning and Workload Allocation: High and dynamic variability in the amount of resources available in different parts of networked environments (different processors, network connections, software) strongly suggest that alternative partitioning and load balancing schemes are a pre-requisite to optimal performance. We are investigating the use of threads and objects to address this issue.
- Alternative Protocol Architectures: In the present implementation of PVM, standard network protocols (i.e. TCP/UDP/IP)are used; these are thought to be overly general and in many respects, unsuitable for parallel distributed computing. We are evolving an alternative protocol scheme where PVM and application programs directly access the data link interface of high-speed local networks, thus bypassing much of the protocol processing and resultant overheads.

7 Acknowledgements

We would like to thank Rod Fatoohi and Horst Simon of NASA Ames for their assistance with the NPB codes; the administrators of the three testbeds for permitting access; and the anonymous referees whose comments led to significant improvements to the paper.

References

[1] D. Bailey, J. Barton, T. Lasinski, and H. Simon, eds. "The NAS Parallel Benchmarks". Report RNR-91-002, Moffett Field, CA: NASA Ames Research Center, 1991.

[2] D. Bailey, E. Barszcz, et al. "The NAS Parallel Benchmarks." *International Journal of Supercomputer Applications*, Vol. 5, No. 3, pp.63-73, Fall 1991.

[3] D. Bailey, E. Barszcz, L. Dagum, and H. Simon, "NAS Parallel Benchmark Results", Report RNR-92-002, NASA Ames, Moffett Field, California February, 1993.

[4] V. S. Sunderam, "PVM : A Framework for Parallel Distributed Computing", *Journal of Concurrency: Practice and Experience*, **2**(4), pp. 315-339, December 1990.

[5] R. Butler and E. Lusk, "User's Guide to the P4 Programming System", Argonne National Laboratory, Technical Report ANL–92/17, 1992.

[6] A. Kolawa, "The Express Programming Environment", *Workshop on Heterogeneous Network-Based Concurrent Computing*, Tallahassee, October 1991.

[7] Louis Turcotte, "A Survey of Software Environments for Exploiting Networked Computing Resources", Draft Report, Engineering Research Center for Computational Field Simulations, Mississippi State, January 1993.

[8] G. A. Geist and V. S. Sunderam, "Network Based Computing on the PVM System", *J. Concurrency: Practice and Experience*, **4**(4), pp. 293-311, June 1992.

[9] B. Schmidt and V. S. Sunderam, "Empirical Analysis of Overheads in Cluster Environments", *J. Concurrency: Practice and Experience*, **6**(1), pp. 1-32, Feb. 1994.

[10] A. Beguelin, et. al., "Solving Computational Grand Challenges Using a Network of Supercomputers", *Proceedings of the Fifth SIAM Conference on Parallel Processing*, D. Sorensen, ed., SIAM, Philadelphia, 1991.

[11] M. T. Heath and J. A. Etheridge, "Visualizing the Performance of Parallel Programs", IEEE Software, 8(5), pp. 29-39, September 1991.