The Evolution of the PVM Concurrent Computing System*

G. A. Geist

Mathematical Sciences Section
Oak Ridge National Laboratory
Oak Ridge, TN 37831

V. S. Sunderam

Department of Math & Computer Science
Emory University
Atlanta, GA 30322

Abstract

Concurrent and distributed computing, using portable software systems or environments on general purpose networked computing platforms, has recently gained widespread attention. Many such systems have been developed, and several are in production use. This paper describes the evolution of the PVM system, a software infrastructure for concurrent computing in networked environments. PVM has evolved over the past three years; it is currently in use at several hundred institutions worldwide for applications ranging from scientific supercomputing to high performance computations in medicine, discrete mathematics, and databases, and for learning parallel programming. We describe the historical evolution of the PVM system, outline the programming model and supported features, present results gained from its use, list representative applications from a variety of disciplines that PVM has been used for, and comment on future trends and ongoing research projects.

1 Introduction

High-performance computing, particularly in scientific domains, is increasingly based on parallel processing. One form of parallel or concurrent computing, using portable software systems or environments on general purpose networked computing platforms, has gained tremendous popularity in recent years. Several such software systems that support this model are in widespread use. Two important factors contributing to their success are cost effectiveness (as compared to traditional hardware multiprocessors), and high usability as a result of widely portable implementations, support tools, and straightforward interfaces. Examples of such systems are Isis [1], Linda [2], Express [3], and PVM [7]. It should be noted that these systems, referred to hereafter as "environments", are distinct from distributed operating systems, which are native kernel level implementations. Wide-ranging experiences (in terms of quantity and diversity of use) during the past several years has firmly established the viability and effectiveness of concurrent computing tools such as those mentioned above, and it is the general consensus of the high-performance scientific computing community that these systems will continue to play an active role in concurrent computing in the near-term future.

Concurrent computing environments offer several programming interfaces, but with the exception of a few (notably Linda), they are uniformly based on some variant of message passing. While early transitions to this model from traditional vector machines and share-memory multiprocessors met with some resistance, message passing is now widely accepted as the normal programming and computing model for most parallel machines, as well as for many of the software environments that emulate parallel computers. Essentially, the model is based on collections of asynchronously executing processes that interact and cooperate via the exchange of messages. On many multiprocessors, the process-processor binding is one-to-one, but software systems such as PVM remove this restriction for greater flexibility. Typical manifestations of this model, both on hardware multiprocessors and software environments, consist of user-level libraries that are bound into application programs—these libraries provide facilities for process management, communication, synchronization, and certain auxiliary functions.

The PVM system is a software infrastructure that permits collections of heterogeneous machines to be used as a general-purpose, message-passing, parallel computer. Under PVM, a user defined collection of serial, parallel, and vector computers appears as one

*This work was supported by the Applied Mathematical Sciences program, Office of Basic Energy Sciences, U. S. Department of Energy, under Grant No. DE-FG05-91ER25105, and Contract DE-AC05-84OR21400 with Martin Marietta Energy Systems, Inc.

large distributed-memory computer. Throughout this paper the term *virtual machine* will be used to designate this logical distributed-memory machine. The PVM user library contains functions to start up tasks on the virtual machine and allows the tasks to communicate and synchronize with each other. Applications, which can be written in Fortran77 or C, can be parallelized by using message-passing constructs common to most distributed-memory computers. By sending and receiving messages, multiple tasks of an application can cooperate to solve a problem in parallel.

PVM supports heterogeneity at the application, machine, and network level. Users may exploit this feature in several ways; for instance, PVM allows application tasks to exploit the architecture best suited to their solution. PVM handles all data conversion that may be required if different computers in the virtual machine use different data representations, in addition to making transparent other inconsistencies between machines in terms of architecture and operating system facilities, thereby providing for machine heterogeneity. PVM also permits the virtual machine to be interconnected by a variety of different networks.

The PVM project started in the summer of 1989 at Oak Ridge National Laboratory (ORNL) and is now an ongoing collaborative research project at Emory University, ORNL, and the University of Tennessee (UT), It is a basic research effort aimed at advancing science, and is wholly funded by research appropriations from the U.S. Department of Energy, the National Science Foundation, and the State of Tennessee. Owing to its experimental nature, the PVM project produces software that is of utility to researchers in the scientific community and to others. This software is, and has been distributed freely in the interest of advancement of science and is being used in computational applications around the world.

In the next section, we discuss the PVM computing model and describe the programming interface as well as an operational overview, noting aspects that have undergone evolutionary changes over the duration of the project. We then discuss our experiences with scientific computing using PVM, and present some performance figures based on its use. In order to highlight the diversity of applications that PVM has been used for, Section 4 presents cullings from user feedback obtained in the last several months, and the concluding section discusses the value of PVM and similar systems, and lists ongoing and future projects.

2 The PVM System

PVM (Parallel Virtual Machine) is a software system that permits the utilization of a heterogeneous network of parallel and serial computers as a unified general and flexible concurrent computational resource. The PVM system [7] initially supported the message passing, shared memory, and hybrid paradigms, thus allowing applications to use the most appropriate computing model, for the entire application or for individual sub-algorithms. However, support for emulated shared-memory was omitted as the system evolved, since the message-passing paradigm was the model of choice for most scientific parallel processing applications. Processing elements in PVM may be scalar machines, distributed- and shared-memory multiprocessors, vector supercomputers and special purpose graphics engines, thereby permitting the use of the best suited computing resource for each component of an application. This versatility is valuable for several large and complex applications including global environmental modeling [5], fluid dynamics simulations [6], and weather prediction applications. However, the full effectiveness of the PVM system can be realized, with significant benefits, on common hardware platforms such as a local network of general purpose workstations.

The PVM system is composed of a suite of user-interface primitives supporting software that together enable concurrent computing on loosely coupled networks of processing elements. Some of the prominent advantages of the system are:

- the ability to execute in *existing* network environments without the need for specialized hardware or software enhancements or modifications.
- support for multiple parallel computation models, particularly useful in conjunction with support for multiple hardware architectures.
- integral provision of debugging and administrative facilities, using interactive graphical interfaces.
- support for fault-tolerance and partially degraded execution in the presence of machine or network failures.
- auxiliary profiling and visualization tools that permit post-mortem analysis of program behavior.

2.1 Architectural Description

PVM may be implemented on a hardware base consisting of different machine architectures, including single CPU systems, vector machines, and multiprocessors. These computing elements may be interconnected by one or more networks, which may themselves be different (e.g. one implementation of PVM operates on Ethernet, the Internet, and a fiber optic network). These computing elements are accessed by applications via a standard interface that supports common concurrent processing paradigms in the form of well-defined primitives that are embedded in procedural host languages. Application programs are composed of *components* that are subtasks at a moderately large level of granularity. During execution, multiple *instances* of each component may be initiated. Figure 1 depicts a simplified architectural overview of the PVM system.

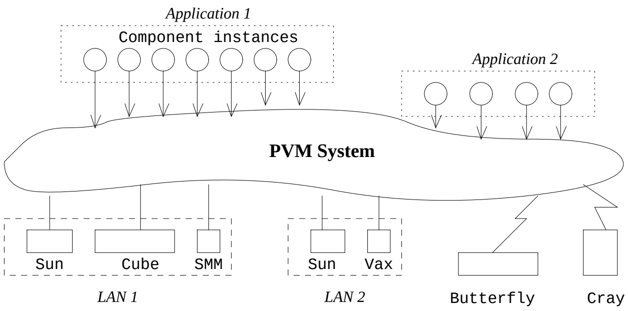

Figure 1: PVM System Overview

Application programs view the PVM system as a general and flexible parallel computing resource. This resource may be accessed at three different levels: the *transparent* mode in which component instances are automatically located at the most appropriate sites, the *architecture-dependent* mode in which the user may indicate specific architectures on which particular components are to execute, and the *low-level* mode in which a particular machine may be specified. Such layering permits flexibility while retaining the ability to exploit particular strengths of individual machines on the network. The PVM user interface is strongly typed; support for operating in a heterogeneous environment is provided in the form of special constructs that selectively perform machine-dependent data conversions where necessary. Inter-instance communication constructs include those for the exchange of data structures as well as high-level primitives such as broadcast, barrier synchronization, mutual exclusion, and rendezvous.

Application programs under PVM may possess arbitrary control and dependency structures. In other words, at any point in the execution of a concurrent application, the processes in existence may have arbitrary relationships between each other and, further, any process may communicate and/or synchronize with any other. This is the most unstructured form of crowd computation, but in practice a significant number of concurrent applications are more structured. Two typical structures are the tree and the "regular crowd" structure. We use the latter term to denote crowd computations in which each process is identical; frequently such applications also exhibit regular communication and synchronization patterns. Any specific control and dependency structure may be implemented under the PVM system by appropriate use of PVM constructs and host language control flow statements.

Multiprocessing on loosely coupled networks provides facilities that are normally not available on tightly coupled multiprocessors. Debugging support, fault tolerance in the form of checkpoint-restart, uniprocessor level I/O facilities, and profiling and monitoring to identify hot-spots or load imbalances within an application are examples. On the other hand, several obstacles and difficulties are also associated with networked concurrent computing. Among these are generating and maintaining multiple object modules for different architectures, considerations of security and intrusion into personal workstations, and a number of administrative and housekeeping functions. In its present form, PVM supports two auxiliary components that provide some desirable features and overcome several of the obstacles. First, the HeNCE interface is a graphical tool that eases many of the application tasks of specifying components, handling input and output, interacting with PVM during execution, managing multiple objects, and providing a debugging interface. Second, PVM is undergoing extensions to provide a uniform programming interface, in a vein similar to the PICL library [12] that supports portable parallel programming and profiling.

2.2 Operational Overview

The PVM system is composed of two parts. The first part is a daemon, called *pvmd*, that resides on all the computers comprising the virtual machine. Pvmd is designed so any user with a valid login can install this daemon on a machine. To run a PVM application, the user executes pvmd on one of the machines, specifying a list of other hosts that together form the virtual machine for this particular session. This first pvmd in turn starts up the daemons on each of the others, and the set of daemons cooperate via distributed algorithms to initialize the virtual machine. The PVM application can then be started by executing a program on any of these computers; the usual method is for this manually started program to spawn other application processes, using PVM facilities. Multiple users may configure overlapping virtual machines, and each user can execute several PVM applications simultaneously.

The second part of the system is a library of PVM interface routines (*libpvm.a*). This library contains user callable routines for message passing, spawning processes, coordinating tasks, and modifying the virtual machine. Application programs must be linked with this library to use PVM.

2.3 The HeNCE subsystem

HeNCE is a tool that greatly simplifies the writing of parallel programs [13]. In HeNCE, the programmer explicitly specifies parallelism between subroutines by drawing a graph where nodes in the graph represent subroutines written in either Fortran or C. The programmer must supply the HeNCE graph and the source code for the subroutine nodes. HeNCE will automatically execute the subroutines in parallel (whenever possible) across a network of heterogeneous machines. The programmer does not need to use any parallel programming primitives and does not need to write any code beyond that supplied for the node subroutines. HeNCE relies on the PVM system [11] for process initialization and communication. Programmers wishing to write explicit message passing parallel programs on a network of machines should explore using the PVM system directly.

HeNCE is composed of 5 integrated graphical tools. The compose tool allows a programmer to specify the parallelism of an application by drawing a graph describing dependencies between user defined procedures. HeNCE uses the graph to automatically write the parallel program. With the configure tool the user specifies a network of heterogeneous computers to be used as a parallel virtual machine and defines a cost matrix between machines and procedures. The build tool uses the configuration and cost matrix to compile and install the procedures written by the first tool on the appropriate machines. The execute tool dynamically maps procedures to machines for execution of the application and collects tracing information. Finally, there is a trace tool that reads the trace information and displays an animation of the execution, either in real time for debugging or later for performance analysis.

2.4 PVM Version 3.0

The PVM model, interfaces, implementation, and portability have proven sound enough that several hundred sites are actively using the system. Such widespread use has aided system evolution enormously, a substantial part of these changes being attributable to user feedback. As a result, the third major version (3.0) of PVM is imminent; prior versions included 1.0 (not publicly released) and 2.0-2.4 (each with a few minor release versions). The basic computing model remains the same in PVM 3.0; however, the system core has been completely redesigned for improved performance, scalability and fault tolerance while maintaining PVM's high standards of portability and robustness. Important modifications are briefly outlined in this section. Specific syntax and detailed explanations of the 3.0 system are omitted from this paper since the specification is subject to change prior to system release scheduled for 2Q 1993.

Based on user feedback and internal changes deemed necessary, the user interface has been somewhat modified in 3.0. Most changes are syntactic, although a few semantic changes were also required. One fundamental change involves process identification and addressing; processes in 3.0 are represented by an integer task identifier (called the *tid*), while in previous versions, processes were identified by a component name and instance number pair. The tid is the primary and most efficient method of identifying processes in PVM. Since tids must be unique across the entire virtual machine, they are supplied by the system. PVM 3.0 contains several routines that return tid values so that the user application can identify other processes in the system. Although less efficient, processes can still be identified by a name and instance number by joining a group. A user defines a group name and PVM returns a unique instance number for this process in this group.

A significant new feature in PVM 3.0 is support for certain forms of failure resilience. If a host (i.e. a computer in a virtual machine) fails, PVM will automatically detect this and delete the host from the virtual machine. The status of hosts can be inspected by the application, and if required a replacement host can be added. It is still the responsibility of the application developer to make his application tolerant of host failure. PVM makes no attempt to automatically recover processes that are aborted because of a host failure. Another use of this feature would be to add more hosts as they become available, for example on a weekend, or if the application dynamically determines it could use more computational power.

Version 3.0 also contains provisions for dynamic process groups. In PVM, functions that logically deal with groups of processes such as broadcast and barrier use the user's explicitly defined group names as arguments. Groups may overlap, and a process can belong to multiple groups. Routines are provided for processes to join or to leave a group. Processes can query for information about other group members.

Finally, version 3.0 contains facilities that enable straightforward integration of hardware multiprocessors into a virtual machine. PVM was originally developed for concurrent computation across multiple, interconnected, but independent, machines. As a result, in order to exploit true multiprocessors such as the Intel iPSC/860, it was necessary to write a PVM program for the "host" processor that received messages from the external network, and routed them to individual nodes using the native message-passing routines on the machine. Similarly for messages from a node destined for an external machine, the PVM program on the host converted these messages from native format to PVM format and routed them over the network. With PVM 3.0 the dependence on network specific mechanisms is relaxed, and communication within multiple network types as well as across them is transparently handled. For example, programs written in PVM 3.0 can run on a network of SUN's, on a group of nodes on an Intel Paragon, on multiple Paragons connected by a network, or a heterogeneous combination of multiprocessor computers that are geographically distributed. PVM 3.0 is designed to use vendor specific communication calls within a multiprocessor.

3 Scientific Computing Experiences

The PVM system has been used for the execution of a variety of application codes on different networks, each with its unique mix of processing elements. Example applications that have been executed under PVM include matrix factorization, stochastic simulation of toroid networks, and Mandelbrot image computations. In this section we describe in detail two applications that are drawn from the domain of scientific computing, and are large, computationally intensive codes that are well known. The first is an application that models the physical properties of complex substitutionally disordered materials, and the second is a molecular dynamics application.

3.1 Superconductivity Studies

The first application models the physical properties of complex substitutionally disordered materials. A few important examples of physical systems and situations in which substitutional disorder plays a critical role in determining material properties include: metallic alloys, high-temperature superconductors, magnetic phase transitions, and metal/insulator transitions. The algorithm is an implementation of the Korringa, Kohn and Rostoker coherent potential approximation (KKR-CPA) method for calculating the electronic properties, energetics and other ground state properties of substitutionally disordered alloys [8].

The KKR-CPA parallel algorithm was executed under the PVM system using several combinations of different numbers of IBM RS/6000 Powerstations. For each of these experiments, no source code changes were made; only the PVM host pool and application input files were modified as appropriate. Owing to the inherent differences in processor speed between the model 320, model 530 and model 550 Powerstations that were used, the number of energies was varied so as to approximately balance the load on each processor. Table 1 shows the effective megaflop rate achieved by the PVM system for the KKR-CPA algorithm on different combinations of IBM RS/6000 Powerstations. In the 13-machine experiment, 4 model 320 systems and 7 model 530 systems were physically on a single Ethernet network, while the remaining 530 and 550 systems were geographically distant, and accessed via a 1.5Mb T1 link. In all other experiments, all RS/6000 systems were interconnected by a single Ethernet network. It can be seen from the table that near linear speedups were attained for this application, owing to the relatively high computation-to-communication ratio. More significant is the extremely high computational rate, particularly considering that speeds in excess of 250 Mflops could be achieved using existing, general purpose RS/6000 Powerstations, interconnected by existing networks that were in simultaneous use for other activities. Of particular note is the relative price-performance ratio of the PVM system on IBM RS/6000 Powerstations as compared to supercomputers and hardware multiprocessors; an abbreviated listing is shown in Table 2.

3.2 Molecular Dynamics Simulations

Another scientific computing application in which very high levels of performance have been achieved using PVM on RS/6000 Powerstations is classical molecular dynamics. Molecular dynamics (MD) simulations are commonly used to calculate static and dynamic

Model 320		model 530	
nproc	Mflops	nproc	Mflops
serial	18.2	serial	24.4
2	31.3	2	45.9
4	63.1	4	92.2
N/A	---	7	161.9
6 (530's) + 4 (320's)			206.5
7 (530's) + 4 (320's)			226.0
1 (550) + 8 (530's) + 4 (320's)			261.0

Table 1 : Performance of HiTc on various PVM - IBM RS/6000 configurations.

Machine	List Price	Mflops	Mflops/$M
PVM (4x320+6x530)	$252K	207	822
Cray YMP (8 procs)	$20M	2290	114
iPSC/860 (128 procs)	$3M	2527	842

Table 2: Price-Performance Ratios for HiTc Application

properties of liquid and solid state systems [9]. This application treats each of the N atoms (or molecules) as a point mass, and Newtons equations are then integrated to move each atom forward in time. Individual force equations are derived for each atom based on the potential energy functional for the system. The algorithms discussed in this section pertain to the general class of MD problems in which (a) only short range forces are of interest; (b) atoms diffuse, i.e. each atom's neighbors change as the simulation progresses; and (c) systems consisting of a few hundred to several thousand atoms (relatively small systems) are considered.

MD algorithms are inherently parallel, as discussed in [10]. Several parallel algorithms for MD simulations have been developed, and executed on a variety of vector and multiprocessor architectures, ranging from the Cray-XMP vector supercomputer to 1024-node hypercubes. A detailed discussion of the algorithms, speedups, and problems in parallelizing this code may be found in the literature. The results reported indicate that MD simulations for small systems (200-2000 atoms) require approximately equal times on the Cray-XMP and on a 1024-node Ncube/2 hypercube. For larger systems (2000-10000 atoms), parallel algorithms on the Ncube/2 can execute at up to twice the speed of the Cray. These findings are consistent with the observation that as the size of the system increases, the communication to computation ratio of the parallel algorithms decreases, thereby reducing the message passing overheads in the parallel implementation.

The MD simulation algorithm was implemented on the PVM system for execution on a network of RS/6000 Powerstations. This algorithm assigns to each processor, a fixed region of space, and updates the positions of all atoms within its box in a given timestep. The PVM implementation of this MD algorithm was adapted from an algorithm originally designed for the hypercube architecture. Given the regular interconnection structure of the hypercube, the control and communication structure is also regular; essentially consisting of iterative phases of PDE solutions, followed by data exchanges along each dimension of the hypercube. The data exchanged between each computational phase consists of atom velocities and force values; typical runs involved the exchange of several hundred messages, each of the order of several hundred KB in size.

The results of the molecular dynamics application for a range of processors and problem sizes are given in Table 3. The table compares the execution times of PVM using a network of RS/6000 Powerstations and the iPSC/860 hypercube. For a small number of processors, PVM over a 1.2 MB/sec Ethernet is quite competitive with a hypercube with dedicated 2.8 MB/sec channels. Load imbalances became worse on PVM when eight processors were used because the workstations had different computational rates. With an even more heterogeneous mixture of machines, the load imbalances would be expected to get much worse given this application's method of parallelization. (These load imbalances are not seen in the KKR-CPA application because its method of parallelization employs a dynamic load balancing scheme.) Nevertheless, it is encouraging to note that the PVM system performs quite well, even for the MD application which inherently has a high communication to computation ratio.

4 PVM Usage and Application Domains

As mentioned earlier, several hundred sites are actively using PVM at the time of writing. Along with Express, Linda, and Parmacs, PVM appears to be a highly viable toolkit and methodology for concurrent computing on heterogeneous networks of processing elements. In this section, we present feedback regarding PVM usage from a very small cross-section of the user community, to highlight the different application

Molecular Dynamics Simulation			
PVM RS/6000 procs	Problem size 5x5x5	8x8x8	12x12x12
1	23	146	1030
2	15	91	622
4	12	62	340
8	6	34	184
iPSC/860 procs			
1	42	202	992
2	22	102	500
4	11	52	252
8	6	27	129

Table 3: Times in seconds for MD simulations

domains and hardware environments in which PVM is being utilized.

- PVM is in use at Los Alamos National Laboratory for numerically intensive computations on clusters of 16-IBM RS6000/560 workstations connected with Ethernet, FDDI, and Socc.

- Lawrence Livermore Laboratory mention their use for computer aided tomography of the soft x-ray emission on the DIII-D tokamak on HP and Sun workstations and Cray supercomputers.

- A scientific computing consultancy uses PVM for large (100,000-500,000 line) scientific applications involving meteorology, photochemistry, particle dispersion and underground transport and aqueous chemistry.

- At the Colorado School of Mines, PVM is used for running seismic migration applications on several workstations. Seismic migration, which gives the subsurface image of the earth from the data recorded at the surface, is computationally intensive, especially in 3-D case. In real 3-D seismic migration, the data set is about 4G bytes and the computation required is of the order of several billion floating point operations. The conclude that they can run a real problem on 5 IBM/RS6000-520 and 530 in about 15 hours, which is comparable to the performance on CRAY. In this application, PVM is used not only to do the computation in parallel, but also to coordinate the disk I/O in order to make the data I/O in parallel.

- At the University of Utah, researchers are developing a parallel solver for nonsymmetric partial differential equations, which they will eventually integrate into a large-scale combustion model. Currently they use PVM on a network of IBM RS/6000 model 520 workstations, connected by token ring. These processors will soon be upgraded to model 560s, and the token ring network will be replaced by an FDDI network. They also plan to offer PVM as the basis for network parallel computing on an FDDI configuration.

- At a NASA research center, PVM is being used for (1) flow simulations for a variety of problems in hypersonic, rarefied flows using the Direct Simulation Monte Carlo method (a particle method). Some of these simulations have run more or less continuously for several months on multiple SparcStations for a single problem. They are using PVM on a regular production basis on 8 SPARC-2's and have run some benchmarks using up to 32 Sparcs. (2) for a multi-discipline design and optimization code that they are implementing in a multi-computer environment. (3) for a CFD problem as well as a coarse breakdown of an optimization problem within an aircraft design and mission analysis simulation.

- The Jefferson Cancer Institute is part of the Jefferson Medical College, Thomas Jefferson University. The institute conducts basic and applied research in areas which include pharmacology, microbiology, molecular biology, molecular genetics, and structural biology. Researchers have endeavored to take maximum advantage of information technologies in all of these areas. Technologies in wide spread use include: relational and object oriented databases, high performance 3d graphics, analytical tools for molecular modeling, emergent computation and genetic algorithms. PVM supports applications in many of these areas which share two common threads: (1) algorithms which lend themselves to emergent approaches; and (2) applications which integrate data and services from multiple machines.

- At United Technologies, PVM is being used for (1) a generalized three-dimensional, multi-block, Navier-Stokes algorithm, presently being used to analyze fundamental fluid dynamic phenomena such as shock/boundary layer interaction and vortex generation devices. A parallel version of EAGLE has been implemented with PVM on multiple singleton processors of two Alliant FX/2800's and on a network of HP9000 workstations. Selected optimal cases dominated by floating point computations have shown 88% efficiency. (2) a three-dimensional CFD algorithm which models the wake structure of helicopter rotor blades, currently being used by UTRC and Sikorsky to calculate wake influence in several aerodynamic/blade response codes. A parallel version of the kernel has been implemented with PVM on multiple singleton processors of two Alliant FX/2800's and on a network of HP9000 workstations. Selected optimal cases dominated by floating point computations have shown 99% efficiency.

- In the Biophysics department at the State University of New York in Buffalo, PVM is an integral part of the computing environment. Their main PVM applications concern kinetic analysis of ion channel currents. This problem is solved both by analytical and Monte Carlo simulation methods.
- At Stanford University, PVM is the message passing platform for one of the implementations of Jade, a language for writing serial, imperative programs that run in parallel on a heterogeneous collection of machines. They have implemented a sparse Cholesky factorization algorithm, a program that simulates water, and an implementation of the Barnes-Hut n-body solver.
- Also at Stanford, researchers are developing a method for protein structure prediction, for which they exhaustively search all possible tertiary conformations using a simplified representation of a polypeptide chain. They have constructed a parallel interface library that lets them port the code between a distributed-memory Ncube-2 system, a shared-memory Silicon Graphics 4D-240, or a network of workstations running PVM.
- At the (only) supercomputing center in Taiwan, PVM is used for various scientific applications including Computational Fluid Dynamic equations, and imcompressible Navier-Stokes equations.
- Katholieke Universiteit Leuven in Belgium uses PVM for medical image enhancement with a (time consuming) Monte Carlo algorithm.
- At the University of Chicago, researchers are running a code to calculate the primordial abundances of elements from inhomogeneous models of nucleosynthesis.
- At the University of Heidelberg, PVM is being used for molecular biology and biotechnology applications which scan databases of protein sequences for similarities that are significant in terms of protein structures or functions. The application is running on a Parsytec machine under Parix and a port to a CM5 and KSR-1 is planned.
- At Boeing, PVM is being used for a CFD code utilizing modular multizonal methods which lend themselves well to coarse grained parallelization. Initially, they are running it on a cluster of 12 SGI workstations, and they expect to extend it to include some RS6000's, some HP 700 series machines and a Cray YMP.
- Cerfacs in France use PVM for several applications, including a project that will couple a model of the ocean and a model of the atmosphere. PVM is being used to send the boundary information between the two models. The second application is a parallel algorithm for solving the global optimization problem. This algorithm is very coarse grained and extremely well suited for a heterogeneous computation environment. They are using the algorithm to solve optimization problems for planning satellite trajectories for satellites dedicated to collecting altimeter information.
- The Canadian National Research Council EGS (Electron Gamma Shower) developers, with a user base of about 5000, use PVM extensively. EGS is a public-domain Monte Carlo code that does electron gamma transport in the energy range 1 keV – 20 TeV. It is ideally suited to a network operating system like PVM since it requires little internode communication.

5 Conclusions and Ongoing Work

In this paper, we have described the PVM system and its evolution, experiences, and uses. The PVM system is a valuable environment for the concurrent execution of applications on heterogeneous networked platforms. PVM is attractive from both technical and economic viewpoints; experiences have been very encouraging and have demonstrated PVM's potential for achieving supercomputer level performance at a fraction of the cost, as well as for providing highly effective emulations of parallel machines. The latter point is especially important; literally everyone has access to high performance workstations which, if used collectively can be as effective as true hardware multiprocessors. The PVM software infrastructure is instrumental in enabling a collection of these machines to be seamlessly integrated into a coherent and flexible concurrent computing resource. Its ease of use, coupled with support for multiple concurrent programming paradigms and auxiliary tools for debugging and monitoring support, significantly increases the effectiveness of network based computing. Ongoing and future work on the PVM system includes support for high speed fiber optic networks, enhancing the program development facilities, support for object-oriented concurrent computing, process-level fault tolerance, and increasing the efficiency of internal algorithms. We believe that the PVM approach is a viable alternative or complement to conventional supercomputing.

References

[1] K. Birman and K. Marzullo, "ISIS and the META project" *Sun Technology* , pp. 90-94, Summer 1989.

[2] M. Arango, D. Berndt, N. Carriero, D. Gelernter, and D. Gilmore, "Adventures with Network Linda", *Supercomputing Review*, Vol. 3 No. 10, Oct. 1990.

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

[4] V. Sunderam, "PVM: A Framework for Parallel Distributed Computing", *Concurrency: Practice and Experience* Vol. 2 No. 4, Dec. 1990.

[5] H. Narang et al. "Design of a simulation interface for a parallel computing environment", *Proc. ACM Southeastern Conf.*, April 1990.

[6] D. W. Lozier and R. G. Rehm, "Some Performance Comparisons for a Fluid Dynamics Code", *Parallel Computing*, 11(2):305-320, 1989.

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

[8] G. Stocks, et al., "Complete Solution of the Korringa-Kohn-Rostoker Coherent Potential Approximation: Cu-Ni Alloys", *Phys. Rev. Letters*, 41, 339, 1978.

[9] S. Plimpton, "Molecular Dynamics Simulations of Short-Range Force Systems on 1024-Node Hypercubes", *Proc. Fifth Distributed Memory Computing Conference*, ed. D. Walker and Q. Stout, IEEE Computer Society Press, pp. 478-483, April 1990.

[10] H. Nguyen, et. al., "A Parallel Molecular Dynamics Strategy", *J. Comput. Chem.*, 6,634, 1985.

[11] A. Beguelin, J. J. Dongarra, G. A. Geist, R. Manchek, and V. S. Sunderam. "A Users' Guide to PVM Parallel Virtual Machine", Technical Report ORNL/TM-11826, Oak Ridge National Laboratory, July 1991.

[12] G. A. Geist, M. T. Heath, B. W. Peyton, and P. H. Worley. "A machine-independent communication library", In J. Gustafson, editor, *The Proceedings of the Fourth Conference on Hypercubes, Concurrent Computers, and Applications*, pages 565–568, P.O. Box 428, Los Altos, CA, 1990. Golden Gate Enterprises.

[13] A. Beguelin, J. Dongarra, G. Geist, R. Manchek, and V. Sunderam, "Solving Computational Grand Challenges Using a Network of Supercomputers", *Proceedings of the Fifth SIAM Conference on Parallel Processing*, D. Sorensen, ed., SIAM, Philadelphia, 1991.