The PVM Concurrent Computing System: Evolution, Experiences, and Trends [*]

V. S. Sunderam
Department of Mathematics and Computer Science
Emory University, Atlanta, GA 30322, USA

G. A. Geist
Mathematical Sciences Section
Oak Ridge National Laboratory
Oak Ridge, TN 37831, USA

J. Dongarra & R. Manchek
Computer Science Department
University of Tennessee, Knoxville, TN 37996, USA

Abstract

The PVM system, a software framework for heterogeneous concurrent computing in networked environments, has evolved in the past several years into a viable technology for distributed and parallel processing in a variety of disciplines. PVM supports a straightforward but functionally complete message passing model, and is capable of harnessing the combined resources of typically heterogeneous networked computing platforms to deliver high levels of performance and functionality. In this paper, we describe the architecture of PVM system, and discuss its computing model, the programming interface it supports, auxiliary facilities for process groups and MPP support, and some of the internal implementation techniques employed. Performance issues, dealing primarily with communication overheads, are analyzed, and recent findings as well as experimental enhancements to are presented. In order to demonstrate the viability of PVM for large scale scientific supercomputing, the paper includes representative case studies in materials science, environmental science, and climate modeling. We conclude with a discussion of related projects and future directions, and comment on near and long-term potential for network computing with the PVM system.

[*]Research 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.; and the National Science Foundation, under Award Nos. CCR-9118787 and CCR-8809615.

1 Introduction

The past several years has witnessed an ever-increasing acceptance and adoption of parallel processing, both for high-performance scientific computing as well as for more "general purpose" applications. Furthermore, the message passing model appears to be gaining predominance as the paradigm of choice, from the perspective of number and variety of multiprocessors (especially massively parallel processors), and also in terms of applications, languages, and software systems for its support. This paper concerns one such message passing system – PVM (Parallel Virtual Machine), that is a software infrastructure that emulates a generalized distributed memory multiprocessor in heterogeneous networked environments. Such an approach, which obviates the need to possess a hardware multiprocessor, has proven to be a viable and cost-effective technology for concurrent computing in many application domains. Owing to its ubiquitous nature – a virtual parallel machine may be constructed using PVM with any set of machines one has access to – and also due to its simple but complete programming interface, the PVM system has gained widespread acceptance in the high-performance scientific computing community. In this paper, we present the rationale and motivations for this project, the model supported, important design decisions and performance considerations, and case studies in the use of PVM for scientific supercomputing.

1.1 Heterogeneous Network Computing

Heterogeneous, network-base, concurrent computing refers to an evolving methodology for general purpose concurrent computing where

- The hardware platform consists of a collection of multifaceted computer systems of varying architectures, interconnected by one or more network types. A special case (albeit the most common at present) is a collection of similar or identical workstations on a single local area network, although a more specific term for such environments is "cluster".
- Applications are viewed as comprising several sub-algorithms, each of which is potentially different in terms of its most appropriate programming model, implementation language, and resource requirements

Heterogeneous network computing refers to models, techniques and toolkits to match heterogeneous environments on the one hand with complete applications, consisting of different subtasks, on the other. While explicit attention to the heterogeneous aspects and "functionality exploitation" in such scenarios has only recently received formalized attention, the concept, in some form has been explored previously, e.g. [2], although usually with a narrow focus or based on a specialized architecture. The PVM system was designed to realize a more general and encompassing interpretation of heterogeneous computing, and had a pragmatic bias aimed at providing a working system that could be used in existing environments — PVM supports heterogeneous machines, applications, and networks. Research is continuing towards the eventual goals of the project, namely to propose a heterogeneous application development model and associated programming frameworks, to enable optimal mapping between application subtasks and the best-suited machines, and to provide adequate infrastructure for heterogeneous debugging, visualization, profiling and monitoring. In the meantime however, current realizations of PVM, supporting basic heterogeneous features and robust emulations of heterogeneous concurrent machines, have proven to be valuable and effective for more traditional applications, especially in high-performance scientific computing. The facilities that are currently available, as well as recent results and ongoing work, are discussed in the following sections.

1.2 The Evolution of PVM

The PVM project started in the summer of 1989, and has evolved through three versions of the software, the latter two of which have been publicly distributed. The original version of the system [3] was ambitious, in that it attempted to be heterogeneous in terms of programming model as well – support for emulated shared memory, in addition to message passing, was incorporated. The basic computing model, which has remained semantically unchanged, 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. The unit of concurrency in PVM is a process, and dependencies in the process flow graph are implemented by embedding appropriate PVM primitives for process management and synchronization within control flow constructs of the host programming language. The implementation model, also unchanged from the original version, uses the notion of a "host pool", a collection of interconnected computer systems that comprises the virtual machine, on which *daemon* processes execute and cooperate to emulate a concurrent computing system. Applications request and receive services from the daemons; the facilities supported essentially fall into the categories of process management and virtual machine configuration, message passing, synchronization, and miscellaneous status checking and housekeeping tasks.

PVM is the mainstay of the Heterogeneous Network Computing research project, a collaborative venture between Emory University, Oak Ridge National Laboratory, and the University of Tennessee. In addition to the authors, Keith Moore, and Weicheng Jiang of UT, and Adam Beguelin of CMU are co-investigators. This project 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. However, 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.

1.3 Related Work

A number of projects based on the same principle, namely utilizing a collection of interconnected machines as a concurrent computing platform, have been developed and several enjoy widespread adoption and following. While there is some commonality with PVM, other similar systems offer (sometimes radically) different programming and implementation models, and present varied functionality and performance. The more widely adopted of these systems are described in detail elsewhere in this volume; in this subsection, we mention a few representative systems and comment on differences in functionality and performance.

Linda [5] is a concurrent programming model based on the concept of a “tuple-space”, a distributed shared memory abstraction via which cooperating processes communicate. P4 and (its derivative Parmacs) [6] are libraries of macros and subroutines developed at Argonne National Laboratory and GMD, for programming a variety of parallel machines. They support both the shared-memory model (based on monitors) and the distributed-memory model (using message-passing). Express is a collection of tools, including a message passing interface, for programming distributed memory multiprocessors, including network clusters [7]. Various other systems with similar capabilities are also in existence; a reasonably comprehensive listing may be found in [8].

With the exception of Linda, whose programming model is not based on conventional message passing, most other systems support very similar facilities, the core primitives being system-specific variants of `send` and `receive`. PVM supports dynamic process and virtual machine management unlike other systems where the process structure is statically defined. The PVM message passing primitives are oriented towards heterogeneous operation, involving strongly typed constructs for buffering and transmission; some other systems provide for untyped data transfer. Compared to other systems, the suite of interface primitives supported by PVM is small; for example P4 and Express provide global combining routines, and Parmacs supports process topologies. These omissions are deliberate to a certain extent; the PVM philosophy is to support a core kernel of primitives above which auxiliary layers may be added (e.g. a PICL [9] port to PVM [10]), while ensuring that facilities that can only be provided at the system level are comprehensive and functionally complete.

In terms of performance, most message passing systems exhibit only marginal differences, although systematic comparative studies have not been undertaken. In the context of PVM and similar systems, performance is usually measured by (1) end-to-end communications speeds delivered by the software system; and (2) overall execution times of various complete applications as measured by elapsed wall clock times. Neither of these parametrizations is very meaningful however, since in typical networked environments, a number of dynamically varying external influences continually affect these measures. Given these qualifications, and the fact that network-based systems for the most part utilize the same underlying transport mechanisms, performance variations tend to be both insignificant and inconsistent. In a later section, we present experimental measurements of performance for the PVM system, and propose strategies under investigation to enhance them.

2 PVM Model and Features

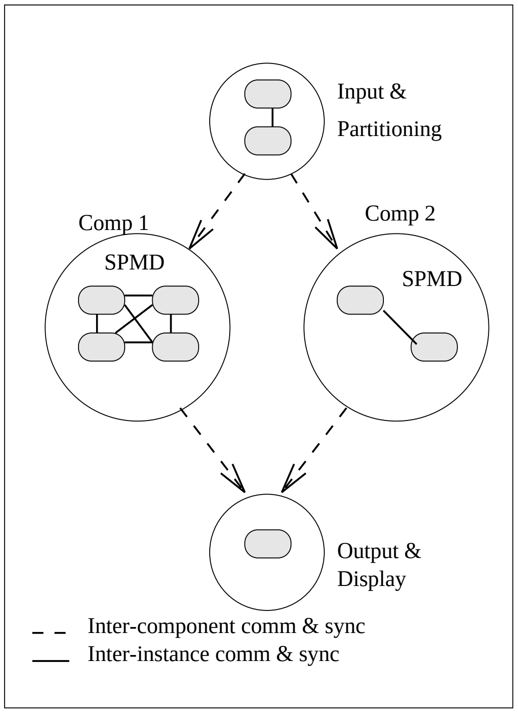
2.1 PVM Computing Model

Under PVM, a user defined collection of serial, parallel, and vector computers emulates a large distributed-memory computer. Throughout this report the term *virtual machine* will be used to designate this logical distributed-memory computer, and *host* will be used to designate one of the member computers. Multiple users can configure overlapping virtual machines, and each user can execute several PVM applications simultaneously. PVM supplies the functions to automatically start up tasks on the virtual machine and allows the tasks to communicate and synchronize with each other. A *task* is defined as a unit of computation in PVM analogous to a Unix process. It is often a Unix process, but not necessarily so. 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. Figure 1 depicts the PVM computing model as well as an architectural abstraction of the system.

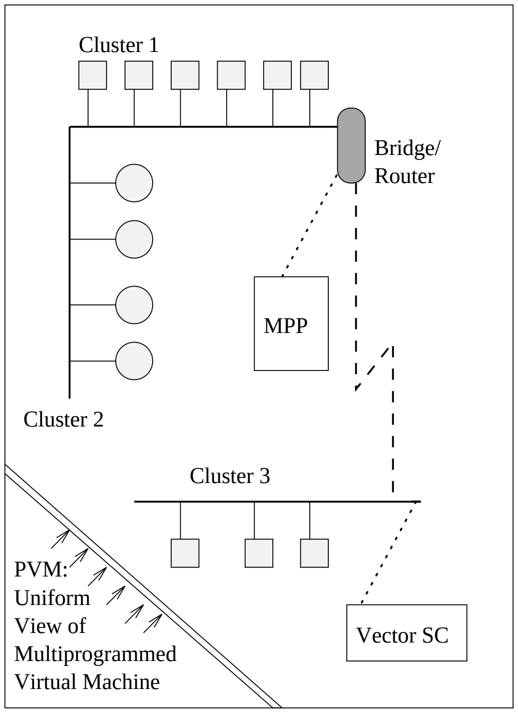
The model assumes that any task can send a message to any other PVM task, and that there is no limit to the size or number of such messages. The PVM communication model provides asynchronous blocking send, asynchronous blocking receive, and non-blocking receive functions. In our terminology, a blocking send returns as soon as the send buffer is free for reuse regardless of the state of the receiver. A non-blocking receive immediately returns with either the data or a flag that the data has not arrived, while a blocking receive returns only when the data is in the receive buffer. In addition to these point-to-point communication functions the model supports multicast to a set of tasks and broadcast to a user defined group of tasks. The PVM model guarantees that message order is preserved between any pair of communicating entities.

2.2 Core Features

PVM supplies routines that enable a user process to register/leave a collection of cooperating processes, routines to add and delete hosts from the virtual machine, to initiate and terminate PVM tasks, to synchronize with and send signals to other PVM tasks, and routines to obtain information about the virtual machine configuration and active PVM tasks. Synchronization may be achieved in one of several ways, e.g. by sending a Unix signal to another task, or by using barriers. Another method notifies a set of tasks of an



(a) PVM Computation Model



(b) PVM Architectural Overview

Figure 1: PVM System Overview

event occurrence by sending them a message with a user-specified tag that the application can check for. The notification events include the exiting of a task, the deletion (or failure) of a host, and the addition of a host.

PVM provides routines for packing and sending messages between tasks. The core communication routines include an asynchronous send to a single task, and a multicast to a list of tasks. PVM transmits messages over the underlying network using the fastest mechanism available e.g. either UDP, TCP on networks based on the Internet protocols, or other high-speed interconnects available between the communicating processors. One example of this third option is described in section 2.3. Messages can be received by filtering on source or message tag (both of which may be specified as wildcards), with either blocking or non-blocking receive routines. A routine can be called to return information about received messages such as the source, tag, and size of the data. Message buffers are allocated dynamically, thereby permitting messages limited in size only by native machine parameters. There are routines for creating and managing multiple send and receive buffers. This feature allows the user to write PVM math libraries and graphical interfaces that can be called inside other PVM applications without communication conflicts. The user can switch context from one set of buffers (for example used by the application) to another set of buffers (for example used inside a math library call).

2.3 Auxiliary Features

Dynamic process groups are layered above the core PVM routines. A process can belong to multiple groups, and groups can change dynamically at any time during a computation. Routines are provided for tasks to join and leave a named group. Group members are uniquely numbered from zero to the number of group members minus one. If gaps appear in this numbering due to tasks leaving the group, PVM attempts to fill these gaps with subsequently joining tasks. Tasks can also query for information about other group members. Functions that logically deal with groups of tasks such as broadcast and barrier use the user's explicitly defined group names as arguments.

PVM version 3 is designed so that native multiprocessor calls can be compiled into the source. This allows the fast message-passing of a particular system to be realized by the PVM application. Messages between two nodes of a multiprocessor use its native message-passing routines, while messages destined for an external host are routed via the user's PVM daemon on the multiprocessor. On shared-memory systems the data movement can be implemented with a shared buffer pool and lock primitives. The MPP subsystem of PVM consists of a daemon that manages the allocation and deallocation of nodes on the multiprocessor. This daemon is implemented in terms of PVM 3 core routines. The second part of the MPP port is a specialized libpvm library for this architecture that contains the fast routing calls between nodes of this host.

2.4 Implementation

The PVM system is composed of two parts. The first is a daemon, called *pvmd3* (sometimes simply *pvmd*), that executes on all the computers making up the virtual machine. Pvmd3 is designed so any user with a valid login can install this daemon on a machine. A user wishing to use PVM first configures a virtual machine by specifying a host-pool list; the daemons are started on each, and cooperate to emulate a virtual machine. The PVM application can then be started from a shell command line prompt on any of these computers.

The second part of the system is a library of PVM interface routines (`libpvm3.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.

The PVM system components have been compiled and tested on the architectures shown in table 1. This table includes hosts ranging from 386 laptop computers to Cray C90s and MPP computers. In addition several vendors are supplying and supporting optimized versions of PVM for their multiprocessor systems including: Cray Research, IBM, Convex, Intel, SGI, and DEC.

ARCH	Machine	Notes
AFX8	Alliant FX/8	
ALPHA	DEC Alpha	DEC OSF-1
BAL	Sequent Balance	DYNIX
BFLY	BBN Butterfly TC2000	
BSD386	80386/486 Unix box	BSDI
CM2	Thinking Machines CM2	Sun front-end
CM5	Thinking Machines CM5	
CNVX	Convex C-series	
CNVXN	Convex C-series	native mode
DGAV	Data General Aviion	
CRAY	C-90, YMP, Cray-2	UNICOS
CRAYSMP	Cray S-MP	
HP300	HP-9000 model 300	HPUX
HPPA	HP-9000 PA-RISC	
I860	Intel iPSC/860	link -lrpc
IPSC2	Intel iPSC/2 386 host	SysV
KSR1	Kendall Square KSR-1	OSF-1
NEXT	NeXT	
PGON	Intel Paragon	link -lrpc
PMAX	DECstation 3100, 5100	Ultrix
RS6K	IBM/RS6000	AIX
RT	IBM RT	
SGI	Silicon Graphics IRIS	link -lsun
SUN3	Sun 3	SunOS
SUN4	Sun 4, SPARCstation	
SYMM	Sequent Symmetry	
TITN	Stardent Titan	
UVAX	DEC MicroVAX	

Table 1: Architectures tested with PVM 3.

3 Performance Considerations

PVM and similar systems normally operate in general purpose networked environments, where neither the CPU's of the individual machines nor the interconnection network is dedicated. As a result, raw performance or speedup of a given application is hard to measure. Even in a dedicated networked environment, with no external use, the above is true since operating system activity, window and filesystem overheads, and administrative network traffic can contribute to deviated measurements. If these factors are ignored, network computing systems behave in a manner that is reasonably predictable [11]. In such a scenario, most of the focus is on communications overhead; CPU optimizations can be approached independently using traditional methods, since parallelism granularity is at the process level.

Network	Message Length							
Type	0	128	512	1K	4K	16K	64K	1M
Ethernet	1.2	1.5	2.1	3.2	7.2	24.5	82.3	1211.2
FDDI	1.2	1.5	1.9	2.5	5.9	16.1	60.3	665.7

Table 2: Data transfer times (milliseconds)

Operation	No. of Procs				
Type	2	4	8	16	32
Barrier	2.2	10.5	28.1	53.2	107.2
Broadcast	3.2	5.5	15.9	28.5	65.9
Opt. Bcast	1.2	3.2	11.5	18.2	35.1

Table 3: Global operation times (milliseconds)

3.1 Raw Communication Performance

Given the above factors, performance evaluation of PVM and similar systems normally begins with an analysis of data transfer costs. The time required for processes to exchange messages is dependent on several factors, including the host machines, network speeds, and most predominantly, the message size. In Table 2 below, we show message passing times in milliseconds for PVM, for varying message lengths and two different network types; these experiments were conducted on unloaded workstations rated at 40-50 MIPS.

Two important observations pertain to Table 2. The first concerns latency, or the minimal time required to send a zero-length message. Irrespective of network type, this measure is of the order of a millisecond, and depends largely on the speed of the host machines, as a significant fraction of this overhead is incurred in within-host protocol processing. The second factor is throughput. As the table shows, the Ethernet network could be driven at near theoretical peak capacity for large messages; similar ratios are conjectured to be possible for fast networks with increases in host speeds and protocol optimizations.

Apart from point-to-point data transfer, group communication facilities are also an important measure of communications performance. Table 3 shows times in milliseconds for barrier synchronization and broadcasting (1K messages) using the release version of PVM that uses naive but robust algorithms, as yet untuned for optimality. Also shown in Table 3 is an experimental multicast (1K messages) facility that exploits the broadcast medium of networks such as Ethernet.

3.2 Improving Communications Performance

As shown in the previous section, communication throughput approaching the medium capacity can be achieved in PVM, provided large messages are transferred. This has also been demonstrated in situations involving multiple, simultaneous message passing; i.e. a high percentage of the *aggregate* bandwidth of the medium is utilizable by PVM application processes. The factor that is difficult to optimize however, is latency, implying poor efficiency and speedup when message exchanges are short and intermittent.

One approach that is under investigation is to enable PVM operation directly above the data link layer (rather than the transport layer); a feasible option when operating in a local network environment. An experimental version of PVM that operates in this fashion has been designed, and latency improvements of the order of 50% have been observed. Work is in progress to integrate this into the release version of PVM, incorporating intelligence to determine on a `send`-by-`send` basis when it is appropriate to utilize this data transfer mechanism.

4 Scientific Supercomputing

PVM is being increasingly adopted at numerous institutions worldwide for distributed scientific computing. Many scientific, industrial, and medical applications are being deployed under PVM in clustered workstation networks. An important motivation (for the use of PVM and other cluster computing systems) is price performance – generally, clusters are about 10 times as cost-effective as supercomputers for a given performance capability, for several classes of applications. Other motivations for the increasing use of PVM include a high degree of portability and a straightforward, robust interface that is well suited for scientific application development.

Three computational grand challenges being addressed by ORNL as well as several other applications important to DOE's mission have been converted to PVM. The grand challenges are in groundwater transport to assist in waste site clean up, first principles materials calculations to assist in the design of new alloys and ceramics, and global climate modeling to predict the effects of things such as ozone depletion, and global warming. In the next sections we briefly describe these three applications and the effort involved in converting them to PVM.

4.1 Groundwater

ORNL is part of a consortium of groundwater researchers whose goal is to develop state of the art parallel models for high performance parallel computers. These computer models will enable researchers to model flow with higher resolution and greater accuracy than previously possible. As a first step researchers at ORNL have developed a parallel 3-D finite element code called PFEM that models water flow through saturated-unsaturated media. PFEM solves the system of equations

$$F \frac{\partial h}{\partial t} = \nabla \cdot [K_s K_r (\nabla h + \nabla z)] + q,$$

where h is the pressure head, t is time, K_s is the saturated hydraulic conductivity tensor, K_r is the relative hydraulic conductivity or relative permeability, z is the potential head, q is the source/sink and F is the water capacity ($F = d\theta/dh$, with θ the moisture content) after neglecting the compressibility of the water and of the media.

PFEM was parallelized by partitioning the physical domain into p pieces and statically assigning one subdomain to each of p tasks. The present version uses only static load-balancing and relies on the user to define the partitioning, but other consortium members are working on ways to automate these operations. At each timestep, each task solves the above equation for its subdomain and then exchanges its boundary region with its neighboring regions. Originally developed on an Intel iPSC/860 multiprocessor, a PVM version of PFEM was straightforward to create requiring an undergraduate student less than 3 weeks to complete. Presently, the PVM version of PFEM has been delivered to members of the groundwater consortium for validation testing using networks of workstations while they await the availability of parallel supercomputers. No performance tests had been done at the time of this writing.

4.2 Materials

ORNL material scientists are developing algorithms for studying 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: high-strength alloys, high-temperature superconductors, magnetic phase transitions, and metal/insulator transitions. One of the algorithms being developed 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. The KKR-CPA method extends the usual implementation of density functional theory to substitutionally disordered materials. In this sense it is a completely first principles theory of the properties of substitutionally disordered materials requiring as input only the atomic numbers of the species making up the solid.

Starting with the original 20,000 line serial KKR-CPA code, it required about three months to produce a PVM version of the code. After profiling the code and studying the potential sites for coarse-grain parallelism, a master/slave paradigm was chosen for implementation. The master task performs all the I/O and coordinates all the slaves. The slave tasks perform the majority of the computational work. This split reduced the amount of memory any one task required and also allowed the master task to do dynamic load balancing. Several megabytes of data is transferred between master and slaves, but no data is shared between slaves. Moreover the data transfers are organized as a few large messages rather than many small ones to reduce message latency overhead.

Using PVM the KKR-CPA code is able to achieve over 200 Mflops utilizing a network of ten IBM RS/6000 (6 model 530's + 4 model 320's) workstations; estimated to be about 82% of the maximum achievable for this code. Given this capability, the KKR-CPA application is being used as a research code to solve important materials science problems. Since its development the KKR-CPA code has been used to compare the electronic structure of two high temperature superconductors, $Ba(Bi_{.3}Pb_{.7})O_3$ and $(Ba_{.6}K_{.4})BiO_3$, to explain anomalous experimental results from a high strength alloy, NiAl, and to study the effect of magnetic multilayers in CrV and CrMo alloys for their possible use in magnetic storage devices.

The PVM KKR-CPA code has also been used to test concepts in distributed computing. For example, with help from Cray Research the KKR-CPA code was run on a network of C90 and YMP multiprocessors. Using 27 processors scattered across several sites, the Cray-based PVM was able to achieve an average aggregate performance of over 9 Gflops while calculating superconductor properties. In a test of the portability of PVM, the KKR-CPA code was run across a virtual machine composed of two Intel Paragons, a CM-5, an Intel i860, and IBM workstations. These hosts are geographically distributed at several sites. In this test the master task ran on one of the IBM workstations and slave tasks ran on nodes of the various MPP hosts. The performance of this test was consistent with the small number of nodes used on each host.

4.3 Climate

A collaboration of researchers from ORNL, Argonne National Lab, and the National Center for Atmospheric Research (NCAR) was formed to address atmospheric modeling. Their first task has been to develop parallel algorithms and implement the recently developed version of the NCAR Community Climate Model (CCM2) in a message passing version for Intel parallel supercomputers. This will be one of the first codes to run on the Intel Paragon and ORNL will support its use by the climate research community using the Paragon. Work now in progress seeks to improve on the parallel efficiency of the code and to develop more comprehensive models with improved capabilities.

The climate model solves the nonlinear PDE's for mass, momentum and energy which govern the general circulation of the atmosphere. Horizontal advection couples columns of the atmosphere while in the vertical direction a large number of processes are coupled. On the shortest time scales the interaction of radiation with the earth's surface, clouds and absorption by the atmosphere couples a vertical column. Surface moisture, latent heat exchange, convective overturning and precipitation processes are also represented within each vertical column. The ocean surface temperature and the ability of the ocean surface layer to store heat during a diurnal cycle are represented without including general ocean circulation.

Since a large portion of the calculations (radiation, absorption, clouds, etc.) in a vertical column of atmosphere are independent, the parallelization strategy used in the climate code is across columns. The columns of atmosphere are divided into adjacent patches and distributed among the processors of an MPP. These independent calculations comprise about 50% of the parallel execution time of the code but exhibit some load imbalance. The coupling in the horizontal direction, the solution of the flow equations, uses a parallel spectral transform algorithm for the approximation of horizontal derivatives and a semi-Lagrangian treatment of the advective term for the moisture equation.

The PVM version was created from a working parallel message passing code running on the Intel iPSC/860 in a couple of weeks. The message passing calls were replaced with equivalent PVM calls. Some special functions that take advantage hypercube or mesh connectivity were replaced with simpler PVM routines. For example, a global maximum is calculated on a single process. Similarly, all the I/O is performed by a single process. The climate researchers report that PVM has offered an excellent debugging and development tool as well as portability across machines and networks.

5 Discussion

In this paper, we have attempted to present overview descriptions of some of the more interesting and important facets of the PVM system, including the design philosophy, computing model, performance issues, and application experiences. Detailed expositions as well as pedagogical material on various aspects of the PVM system may be found, for example, in [12, 13, 14], in addition to the papers already cited. At the time of writing, PVM continues to be a popular and widely used system for concurrent computing, and it is expected that the project will mature and evolve even further in the future. In this section we briefly discuss ongoing auxiliary projects, future plans, and comment on the long term potential and scope of PVM and similar technology.

5.1 Ongoing and Future Work

As mentioned, PVM is an ongoing experimental research project, and continually evolving new ideas are investigated both by the project team and at external institutions; successful experimental enhancements or subsystems eventually become part of the software distribution. One example of a relatively concise enhancement that is undergoing investigation concerns system level optimizations for operating in shared memory environments. Small-scale SMM's are re-emerging, and a version of PVM that utilizes physical shared memory for interaction between the daemon and all user processes on such machines is being developed. Another project is aimed at providing fail-safe capabilities in PVM [15]. This enhanced version uses checkpointing and rollback to recover from single-node failures in an application-transparent manner, provided the application is not dependent on real-time events. Several other enhancements are also in progress, including load balancing extensions, integrating debugging support, and task queue management [16].

One somewhat different and more extensive subsystem under development is a generalized distributed computing (GDC) layer for PVM. While scientific applications have provided the technical impetus for the PVM project, more general and commercially oriented uses are now evolving. In order to support such applications, the GDC layer is being designed, and will support (1) parallel input-output with enhanced filesystem semantics for shadowing, interleaved access, and rollback; (2) access control and authentication, and abstract mutual exclusion mechanisms; (3) support for the client-server model of distributed computing, with facilities for the transparent exporting and invocation of services; and (4) distributed transaction processing primitives. Early results indicate that these facilities can be provided at high levels of efficiency, and that the enhanced functionality will prove beneficial for many new classes of application domains.

5.2 Auxiliary Projects

Apart from the experimental work described in the previous section, a number of other projects related to PVM are in various stages of progress; by nature, these are less intertwined with the internals of the core system, and to a large extent, are auxiliary tools or external subsystems. A few are mentioned below:

- HeNCE is a graphical programming system for PVM; this toolkit generates PVM programs, from depictions of parallelism dependencies as directed graphs, and provides an interactive administrative interface for virtual machine configuration, application execution, and animated visualization.
- Xab [1] is a graphical tool for the run time monitoring of PVM programs. It gathers monitoring events from applications, and displays this information, which can be useful for profiling, error detection, and optimization.
- The DoPVM subsystem [17] is aimed at supporting the "shared object" paradigm in PVM. By writing C++ programs in which objects derived from built-in classes can be declared, this PVM extension permits a shared address space concurrent computing model, thereby alleviating the inherent complexity of explicit message passing programming.

5.3 Long-term Outlook

It is important to realize that PVM (and other similar systems) is not merely a software framework for network-based concurrent computing; it is an integrated methodology for concurrent, distributed, and parallel processing, and more importantly, it is an interface definition for portable application development. From the portability point of view, PVM applications may be migrated not just from one machine (parallel or serial) to another but across different collections of machines. Given that PVM can operate within a parallel machine, across networks, and combinations thereof, significant generality and flexibility exists, and it is hard to imagine a computing environment where such a model would *not* be appropriate. From the point of view of performance, PVM delivers a significant proportion (of the order of 80 – 90%) of the capacity available from the underlying hardware, operating system, network, and protocols – and we expect to retain this characteristic as network and CPU speeds increase, and as protocol and OS software becomes more efficient. It should be pointed out, while on this topic, that in measuring the worth of systems such as PVM, comparisons of clusters vs. MPP's are inappropriate; rather, a meaningful metric is the value added by PVM to a given hardware environment, whether this is a multiprocessor or a collection of workstations. In terms of functionality, the PVM system currently supports an adequate suite of features, and with the integration of extensions described earlier, will be in a position to cater to a much larger realm of distributed and concurrent applications. Heterogeneous network-based concurrent computing systems like PVM are therefore likely to remain viable technologies for concurrent and distributed computing.

References

[1] Adam Beguelin, "Xab: A Tool for Monitoring PVM Programs", *Workshop on Heterogeneous Processing*, IEEE Computer Society Press, Los Alamitos, California, pp. 92-97, April 1993.

[2] H. J. Siegel, et. al., "An Overview of the PASM Parallel Processing System", in it Computer Architecture, D. D. Gajski, et. al. (eds), IEEE Computer Society Press, Washington, DC, pp. 319-407, 1987.

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

[4] A. Beguelin, et. al., "HeNCE Users Guide", University of Tennessee Technical Report, May 1992.

[5] N. Carriero and D. Gelernter, "Linda in Context", *Communications of the ACM*, **32**(4), pp. 444-458, April 1989.

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

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

[8] 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.

[9] 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*, pp. 565-568, P.O. Box 428, Los Altos, CA, 1990. Golden Gate Enterprises.

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

[11] B. Schmidt and V. S. Sunderam, "Empirical Analysis of Overheads in Cluster Environments", *Journal of Concurrency: Practice and Experience*, (to appear), 1993.

[12] 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.

[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.

[14] G. A. Geist and V. S. Sunderam, "The Evolution of the PVM Concurrent Computing System", *Proceedings – 26th IEEE Compcon Symposium*, pp. 471-478, San Fransisco, February 1993.

[15] J. Leon, et. al., "Fail Safe PVM: A Portable Package for Distributed Programming with Transparent Recovery", School of Computer Science Technical Report, Carnegie-Mellon University, CMU-CS-93-124, February 1993.

[16] J. Dongarra, et. al., *Abstracts: PVM User's Group Meeting*, University of Tennessee, Knoxville, May 1993.

[17] C. Hartley and V. S. Sunderam, "Concurrent Programming with Shared Objects in Networked Environments", *Proceedings – 7th Intl. Parallel Processing Symposium*, pp. 471-478, Los Angeles, April 1993.