# Methodologies and Systems for Heterogeneous Concurrent Computing *

V. S. Sunderam[a]

[a]Department of Math and Computer Science,
Emory University, Atlanta, GA 30322, U. S. A.

Heterogeneous concurrent computing is gaining increasing acceptance as an alternative or complementary paradigm to multiprocessor-based parallel processing as well as to conventional supercomputing. While algorithmic and programming aspects of heterogeneous concurrent computing are similar to their parallel processing counterparts, system issues, partitioning and scheduling, and performance aspects are significantly different. In this paper, we discuss critical design and implementation issues in heterogeneous concurrent computing, and describe techniques for enhancing its effectiveness. In particular, we highlight the system level infrastructures that are required, aspects of parallel algorithm development that most affect performance, system capabilities and limitations, and tools and methodologies for effective computing in heterogeneous networked environments. We also present recent developments and experiences in the context of the PVM system, and comment on ongoing and future work.

## 1. INTRODUCTION

The term concurrent computing, as used in this paper, refers to the simultaneous execution of the components of a single application on multiple processing elements. While this definition might also apply to most other notions of parallel processing, we make a deliberate distinction, to highlight certain attributes of the methodologies and systems discussed herein — namely *loose coupling*, physical and logical *independence* of the processing elements, and *heterogeneity*. These characteristics distinguish heterogeneous concurrent computing from traditional parallel processing, normally performed on homogeneous, tightly coupled platforms which possess some degree of physical independence but are logically coherent.

Concurrent computing, in various forms, is becoming increasingly popular as a methodology for many classes of applications, particularly those in the high-performance and scientific computing arenas. This is due to numerous benefits that accrue, both from the applications as well as the systems perspectives. However, in order to fully exploit these advantages, a substantial infrastructural framework is required — in the form of novel programming paradigms and models, systems support, toolkits, and performance analysis and enhancement mechanisms. In this paper, we focus on the latter aspects, namely the systems infrastructures, functionality, and performance issues in concurrent computing.

*Research supported by the National Aeronautics and Space Administration under grant NAG 2-828, the Office of Naval Research grant N00014-93-1-0278, and the National Science Foundation under award number CCR-9118787.

### 1.1. Heterogeneous, Networked, and Cluster Computing

One of the major goals of concurrent computing systems is to support heterogeneity. Heterogeneous computing refers to architectures, models, systems, and applications that comprise substantively different components, as well as to techniques and methodologies that address issues that arise when computing in heterogeneous environments. While this definition encompasses numerous systems, including reconfigurable architectures, mixed-mode arithmetic, special purpose hardware, and even vector and input-output units, we restrict ourselves to systems that are comprised of networked, independent, general-purpose computers that may be used in a coherent and unified manner. Thus, heterogeneous systems may consist of scalar, vector, parallel, and graphics machines that are interconnected by one or more (types of) networks, and support one or more programming environment/ operating system. In such environments, heterogeneity occurs in several forms:

- *System architecture* - heterogeneous systems may consist of SIMD, MIMD, scalar, and vector computers.
- *Machine architecture* - individual processing elements may differ in their instruction sets and/or data representation.
- *Machine configurations* - even when processing elements are architecturally identical, differences such as clock speeds and memory contribute to heterogeneity.
- *External influences* - as heterogeneous systems are normally built in general purpose environments, external resource demands can (and often do) induce heterogeneity into processing elements that are identical in architecture and configuration, and further, cause dynamic variations in interconnection network capacity.
- *Interconnection networks* - may be optical or electrical, local or wide-area, high or low speed, and may employ several different protocols.
- *Software* - from the infrastructure point of view, the underlying operating systems are often different in heterogeneous systems; from the applications point of view, in addition to operating systems aspects, different programming models, languages, and support libraries are available in heterogeneous systems.

Research in heterogeneous systems is in progress in several areas [1, 2] including applications, paradigm development, mapping, scheduling, reconfiguration, etc., but the primary thrust has thus far been in systems, methodologies, and toolkits. This latter thrust has been highly productive and successful, with several systems in production-level use at hundreds of installations worldwide. The body of this paper will discuss two such systems, PVM (Parallel Virtual Machine) and Eclipse, that have evolved into popular and effective methodologies for heterogeneous concurrent computing.

It is worthwhile to note at this juncture, that heterogeneous processing is a superset of similar methodologies referred to as network computing and cluster computing. While the nomenclature is as yet informal, network computing may be considered equivalent to heterogeneous computing, but with rather less emphasis on application heterogeneity, mapping, and task partitioning aspects. Cluster computing is even more restrictive, in that it generally refers to usually identical workstation clusters that are used as a substitute for hardware multiprocessors. These classes of frameworks are distinct from *distributed systems*, although several features and technologies are common to both. Distributed systems are distinguished by their goal of maximal transparency, as well as their emphasis on traditional operating systems issues such as filesystems, process management, kernel services, etc. Figure 1 depicts the relationship between various concurrent computing paradigms.

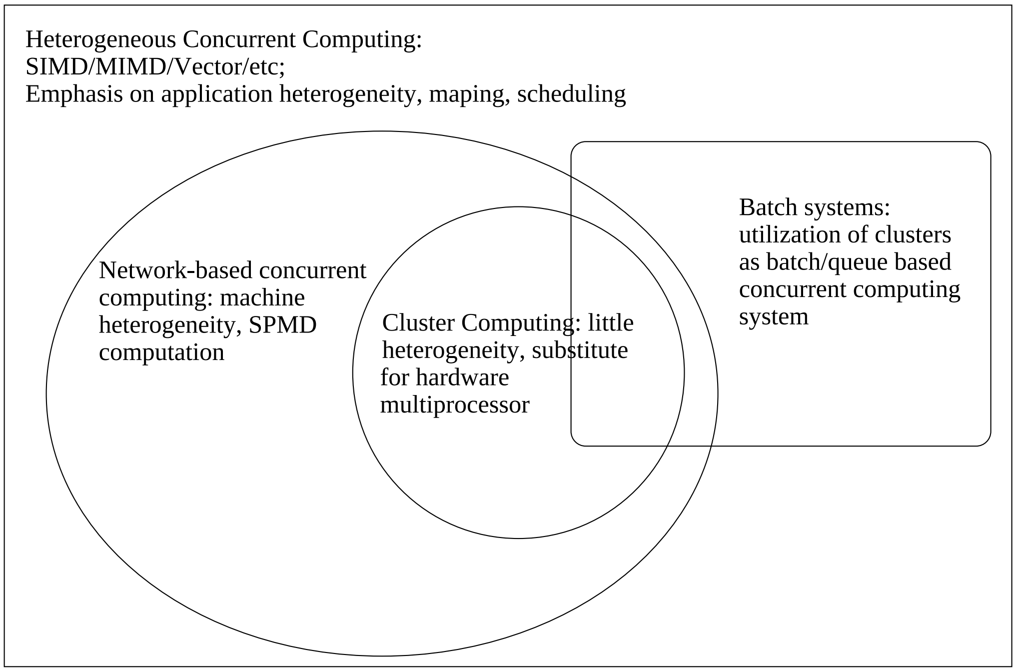


Figure 1: Heterogeneous, Network, and Cluster Computing

### 1.2. Applications Perspective

From the point of view of application development, heterogeneous computing is attractive, since it inherently supports function parallelism, with the added potential of executing subtasks on best-suited architectures. It is well known that different types of algorithms are well matched to different machine architectures and configurations, and at least in the abstract sense, heterogeneous computing permits this matching to be realized, resulting in optimality in application execution as well as in resource utilization. However, in practice, this scenario may be difficult to achieve for reasons of availability, applicability, and the existence of appropriate mapping and scheduling tools. Nevertheless, the concept is an attractive one and several research efforts are in progress in this area [3, 4].

In this respect, many classes of applications that would benefit substantively from heterogeneous computing have been identified. For example, a critically important problem which is ideally suited to heterogeneous computing is is global climate modeling. Simulation of the global climate is a particularly difficult challenge because of the wide range of time and space scales governing the behavior of the atmosphere, the oceans, and the surface. Parallel GCM codes require distinct component modules representing the atmosphere, ocean and surface and process modules representing phenomena like radiation and convection. Sampling, updating and manipulating this data requires scalar, vector, MIMD and SIMD paradigms, many of which can be performed concurrently. Another application domain that could exploit heterogeneous computing is computer vision. Vision problems generally require processing at 3 levels: high, medium and low. Low-level and some medium-level vision tasks often involve regular data flow and iconic operations. This type of computation is well-matched to mesh-connected SIMD machines. Medium-grained MIMD machines are more suitable for various high level and some medium level vision tasks which are communication-intensive and in which the flow of data is not regular. Coarse-grained MIMD machines are best matched for high-level vision tasks such as Image understanding/recognition and symbolic processing.

As previously mentioned however, the above aspect of heterogeneous concurrent computing is still in its infancy. Proof-of-concept research and experiments have demonstrated the viability of exploiting application heterogeneity, and many others are evolving. On the other hand, the systems aspect has matured significantly; to the extent that robust environments are now available for production execution of traditional parallel applications while providing stable testbeds for the evolving, truly heterogeneous, applications. We discuss the systems facet of heterogeneous concurrent computing in the remainder of the paper.

## 2. THE CURRENT SCENARIO

### 2.1. Heterogeneous Concurrent Computing Systems

Heterogeneous computing systems share some common goals and requirements:

- to effectively provide access to significant amounts of computing resources in a cost-effective manner, usually by utilizing already available resources.
- to exploit the existing software infrastructure and facilities (e.g. editors, compilers, debuggers) that are available on individual computer systems in a cluster.
- to provide an effective programming model and interface, generally based on explicit parallelism and the message passing paradigm.
- to support transparency in terms of architecture, processor type, task location, network communication, and resource allocation.
- to achieve the best possible performance, subject to the inherent limitations of the processors and networks involved; some systems also attempt to be non-intrusive by suspending execution in deference to higher priority activities.

Several of the above goals have been met, at least by the most popular network-computing/heterogeneous processing systems. Other goals, such as exploiting heterogeneity, sophisticated job and resource management, automatic parallelization, and graphical interfaces are being pursued. In recent years, there has been a proliferation of network-based concurrent computing systems, some of which also satisfy some attributes of heterogeneous systems. The common aspect of these systems is their emulation of a general purpose concurrent computing platform on interconnected but independent computer systems. Two recently published reports contain concise descriptions of over a hundred network computing and/or auxiliary systems and toolkits [5, 6]. However, recent conferences as well as informal discussion media seem to indicate that most attention is focused on a handful of these; in this section, we provide brief overviews of three such systems, while the next section will discuss the PVM and Eclipse projects in greater detail.

### 2.2. The Linda Model and System

Linda [7] is a concurrent programming model that has evolved from a Yale University research project. The primary concept in Linda is that of a "tuple-space", an abstraction via which cooperating processes communicate. This central theme of Linda has been proposed as an alternative paradigm to the two traditional methods of parallel processing, viz. that based on shared memory, and on message passing. The tuple-space concept is essentially an abstraction of distributed shared memory, with one important difference (tuple-spaces are associative), and several minor distinctions (destructive and non-destructive reads, and different coherency semantics are possible). Applications use the Linda model by embedding explicitly, within cooperating sequential programs, constructs that manipulate (insert/retrieve tuples) the tuple space.

From the application point of view Linda [8] is a set of programming language extensions for facilitating parallel programming. The Linda model is a scheme built upon an associative memory referred to as *tuple-space* It provides a shared memory abstraction for process communication without requiring the underlying hardware to physically share memory. Tuples are collections of fields logically "welded" to form persistent storage items. They are the basic tuple-space storage units. Parallel processes exchange data by generating, reading, and consuming them. To update a tuple, the tuple is removed from tuple-space, modified, and returned to tuple-space. Restricting tuple-space modification in this manner creates an implicit locking mechanism ensuring proper synchronization of multiple accesses.

The "Linda System" usually refers to a specific (sometimes portable) implementation of software that supports the Linda programming model. System software is provided that establishes and maintains tuple spaces, that is used in conjunction with libraries that appropriately interpret and execute Linda primitives. Depending on the environment (shared memory multiprocessors, message passing parallel computers, networks of workstations etc), the tuple space mechanism is implemented using different techniques, and with varying degrees of efficiency. Recently, a new *system* technique has been proposed, at least nominally related to the Linda project. This scheme, termed "Pirhana" proposes a proactive approach to concurrent computing – the idea being that computational resources (viewed as active agents) seize computational tasks from a well known location based on availability and suitability. Again, this scheme may be implemented on multiple platforms, and manifested as a "Pirhana system" or "Linda–Pirhana system".

### 2.3. P4 and Parmacs

P4 is a library of macros and subroutines developed at Argonne National Laboratory for programming a variety of parallel machines. The P4 system [9] supports both the shared-memory model (based on monitors) and the distributed-memory model (using message-passing). For the shared-memory model of parallel computation, p4 provides a set of primitives from which monitors can be constructed, as well as a set of useful monitors. For the distributed-memory model, p4 provides typed send and receive operations, and creation of processes according to a text file describing group and process structure. P4 is intended to be portable, simple to install and use, and efficient. It can be used to program networks of workstations, advanced parallel supercomputers like the Intel Touchstone Delta and the Alliant Campus HiPPI-based system, and single shared-memory multiprocessors. It has currently been installed on most uniprocessor workstations, shared memory multiprocessors, and several high-performance parallel machines.

*Process management* in the P4 system is based on a configuration file that specifies the host pool, the object file to be executed on each machine, the number of processes to be started on each host (intended primarily for multiprocessor systems) and other auxiliary information. Two issues are noteworthy in regard to the process management mechanism in P4. First, there is the notion a "master" process and "slave" processes, and multilevel hierarchies may be formed to implement what is termed a *cluster* model of computation. Second, the primary mode of process creation is static, via the configuration file; dynamic process creation is possible only by a statically created process that must invoke a special P4 function that spawns a new process on the local machine. However, despite these restrictions, a variety of application paradigms may be implemented in the P4 system in a fairly straightforward manner.

*Message Passing* in the P4 system is achieved through the use of traditional `send` and `recv` primitives, parameterized almost exactly as other message passing systems. Several variants are provided for semantics such as heterogeneous exchange, and blocking or nonblocking transfer. A significant proportion of the burden of buffer allocation and management however, is left to the user. Apart from basic message passing, P4 also offers a variety of global operations, including broadcast, global maxima and minima, and barrier synchronization. *Shared Memory* support via monitors is a facility that distinguishes P4 from other systems. However, this feature is *not* distributed shared memory; but rather, a portable mechanism for shared address space programming in true shared memory multiprocessors.

*Parmacs* is a project that is closely related to the P4 effort. Essentially, Parmacs is a set of macro extensions to the P4 system developed at GMD [10]. It originated in an effort to provide Fortran interfaces to the P4 system, but is now a significantly enhanced package that provides a variety of high-level abstractions, mostly dealing with global operations. Parmacs provides macros for logically configuring a set of P4 processes; for example, the macro `torus` produces a suitable configuration file for use by P4 that results in a logical process configuration corresponding to a 3-d torus. Other logical topologies, including general graphs may also be implemented, and Parmacs provides macros used in conjunction with `send` and `recv` to achieve topology-specific communications within executing programs.

## 2.4. Express

In contrast to the other parallel processing systems described in this section, Express toolkit is a collection of tools that individually address various aspects of concurrent computation. The toolkit is developed and marketed commercially by ParaSoft Corporation, a company that was started by some members of the Caltech concurrent computation project. A second distinction is support for PC's that Express provides, in addition to the usual high performance computing platforms and workstations.

The philosophy behind computing with Express is based on beginning with a sequential version of an application and following a recommended development life cycle culminating in a parallel version that is tuned for optimality. Typical development cycles begin with the use of VTOOL, a graphical program that allows the progress of sequential algorithms to be displayed in a dynamic manner. Updates and references to individual data structures can be displayed to explicitly demonstrate algorithm structure and provide the detailed knowledge necessary for parallelization. Related to the above is FTOOL, which provides in-depth analysis of a program including variable use analysis, flow structure and feedback regarding potential parallelization. FTOOL operates on both sequential and parallel versions of an application. A third tool called ASPAR is then used; this is an automated parallelizer that converts sequential C and FORTRAN programs for parallel or distributed execution using the Express programming models.

The core of the Express system is a set of libraries for communication, IO, and parallel graphics. The communication primitives are akin to those found in other systems, and include a variety of global operations and data distribution primitives. Extended IO routines enable parallel input and output, and a similar set of routines are provided for graphical displays from multiple concurrent processes. Express also contains the NDB tool, a parallel debugger that used commands which are based on the popular "dbx" interface. These debugging commands can be issued to single processors or groups of nodes simultaneously.

Express finally contains a set of 'back-end" tools intended to assist performance monitoring and tuning. CTOOL analyzes high level overhead issues such as the relative amount of time spent computing, performing IO and in interprocessor communication. ETOOL shows the relationships between various computing elements and may be used to understand overheads, and cause and effect relationships between actions in different processors. XTOOL profiles CPU usage on a per-processor basis, and may be used at different levels of granularity.

## 2.5. Ongoing Trends

In the next section of this paper, we describe the basic features of the PVM and Eclipse systems. PVM as well as the systems described above have evolved over the past several years, but none of them can be considered fully mature. The field of network based concurrent computing is relatively young, and research on various aspects is ongoing. Although basic infrastructures have been developed, many of the refinements that are necessary are still evolving. Some of the ongoing research projects related to heterogeneous network-based computing are briefly outlined here.

Standalone systems delivering several tens of millions of operations per second are commonplace, and continuing increases in power are predicted. For network computing systems, this presents many challenges. One aspect concerns scaling to hundreds and perhaps thousands of independent machines; it is conjectured that functionality and performance equivalent to massively parallel machines can be supported on cluster environments. A project at Fermilab has demonstrated the feasibility of scaling to hundreds of processors for some classes of problems. Research in protocols to support scaling and other system issues are currently under investigation. Further, under the right circumstances, the network based approach can be effective in coupling several similar multiprocessors, resulting in a configuration that might be economically and technically difficult to achieve with hardware.

Applications with large execution times will benefit greatly from mechanisms that make them resilient to failures. Currently few platforms (especially among multiprocessors) support application level fault tolerance. In a network based computing environment application resilience to failures can be supported without specialized enhancements to hardware or operating systems. Research is in progress to investigate and develop strategies for enabling applications to run to completion, in the presence of hardware, system software, or network faults. Approaches based on checkpointing, shadow execution, and process migration are being investigated.

The performance and effectiveness of network based concurrent computing environments depends to a large extent on the efficiency of the support software, and on minimization of overheads. Experiences with the PVM system have identified several key factors in the system that are being further analyzed and improved to increase overall efficiency. Efficient protocols to support high level concurrency primitives is subgoal of work in this area. Particular attention is being given to exploiting the full potential of imminent fiber optic connections, using an experimental fiber network that is available. In preliminary experiments with a fiber optic network, several important issues have been identified. For example, the operating system interfaces to fiber networks, its reliability characteristics, and factors such as maximum packet size are significantly different from those for Ethernet. When the concurrent computing environment is executed on a combination of both types of networks, the system algorithms have to be modified to cater to these differences, in an optimal manner and with minimized overheads. Another aspect concerns the efficient implementation of inherently expensive parallel computing operations such as barrier synchronization. Particularly in an irregular environment (where interconnections within hardware multiprocessors are much faster than network channels), such operations can cause bottlenecks and severe load imbalances. Other distributed primitives for which algorithm development and implementation strategies are being investigated include polling, distributed fetch-and-add, global operations, automatic data decomposition and distribution, and mutual exclusion.

## 3. THE PVM AND CONCH SYSTEMS

### 3.1. 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 [11] 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.

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. 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 2 depicts a simplified architectural overview of the PVM computing model as well as the system.

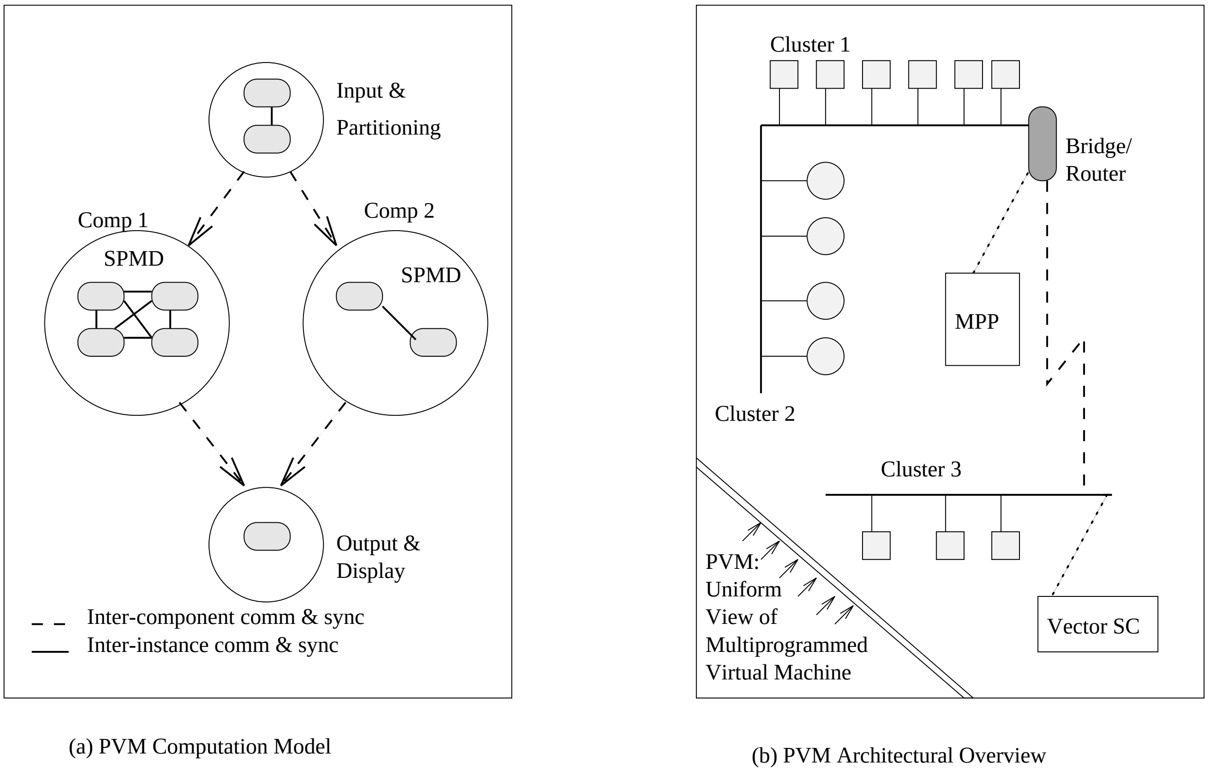


Figure 2: PVM System Overview

Application programs view the PVM system as a general and flexible parallel computing resource. A translucent 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.

The PVM system is composed of two parts. The first part is a daemon, called *pvmd*, that executes on all the computers comprising the virtual machine. PVM is designed so that any user normal access rights to each host in the pool may install and operate the system. daemon on a machine. To run a PVM application, the user executes the daemons on a selected host pool, 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 machine; 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.

### 3.2. The Eclipse System

The Eclipse system [12, 13] was originally designed as a prototype toolkit for concurrent stochastic simulation. Eclipse is also a heterogeneous, network-based, concurrent computing system, but focused on ease of parallelization and library-based implementations, rather than providing a general-purpose parallel programming framework as with more general systems such as PVM, Express, and Linda. Briefly, the Eclipse toolkit provides high level simulation-oriented primitives using which an application programmer describes the simulation process using a sequential computation model. The system transparently replicates appropriate sections of the simulation when the program is executed on a concurrent system. By binding to machine-dependent versions of the Eclipse library, the simulation program can execute without modification on a variety of architectures including uniprocessors, hypercubes, shared-memory machines, and loosely coupled networks, leading, in many classes of applications, to high performance concurrent execution with little development effort. Moreover, it appears that several classes of applications (other than stochastic simulation) also match the Eclipse paradigm, and the system has undergone enhancements to cater to such applications as well.

The Eclipse system consists of three major components. The first is a collection of interface constructs, in terms of which an application program is described. The majority of these constructs fall into two main categories — one comprising domain specific functions, such as random number generation, statistics calculation, and confidence interval testing for the stochastic simulation domain; and the other consisting of primitives that permit the specification of the control structure of the application. The second component of the Eclipse system is the translator — a tool that processes the simulation description (written in a high level language augmented by the constructs mentioned above) and generates architecture dependent variants of the program(s) depending upon the intended target machine. By binding many of the machine dependent aspects of the simulation during this phase, execution time efficiency is improved while retaining portability. The final component of the Eclipse system is a set of run-time libraries that perform the required simulation and control functions during the execution of the simulation. These libraries replicate sections of a simulation application, and manage communication, synchronization, and control of the multiple instances.

In using the Eclipse toolkit, an application programmer defines the control structure of a simulation using toolkit primitives, embedded in a procedural host language. For many applications, this involves tailoring one of several toolkit-provided templates, that are predefined and based on the "typical" structure of several classes of stochastic simulations. Even in those instances when an application does not conform to a system-provided template, the user is only required to provide a *sequential* description of the simulation, written using Eclipse primitives. The system transparently converts this specification (or a tailored template) to a parallel program, thus hiding details regarding the existence of, and communication between, multiple processes. Both C and Fortran interfaces are available in this software library.

The system supplied libraries in Eclipse contain the software infrastructure needed to concurrently execute applications developed as described above on various platforms, including networked environments. Further, the system permits the specification of a logical topology among the processing elements, thereby enabling interconnections that are natural and consistent with process interactions. The common topology for simulation and similar applications is a tree-structure, where leaf nodes perform independent computations while intermediate nodes perform both computation and allocation / re-allocation of work, and collection / consolidation of results. The Eclipse system also provides a limited form of fault tolerance — in tree-structured computations, subtree failures can be tolerated, and optionally, reconstructed at alternative sites. The Eclipse system has proved to be an effective framework for certain application classes; in the next section we describe our experiences with the use of Eclipse for a polymer physics application on large heterogeneous networks.

## 4. RECENT RESULTS

### 4.1. The NAS Parallel Benchmarks

The Numerical Aerodynamic Simulation (NAS) Program of the National Air and Space Administration (NASA) has devised and published a suite of benchmarks [14] for the performance analysis of highly parallel computers. These benchmarks are designed to substantially exercise the processor, memory, and communication systems of current generation parallel computers. They are specified only algorithmically; except for a few constraints, implementors are free to select optimal language constructs and implementation techniques.

The complete benchmark suite consists of eight applications, five of which are termed *kernels* because they form the core of many classes of aerodynamic applications, and the remaining three are simulated CFD applications. The five kernels, and their vital characteristics are listed in Table 1. NASA periodically publishes performance results obtained either from internal experiments or those conducted by third-party implementors on various supercomputers and parallel machines. The *de facto* yardstick used to compare these performance results is a single processor of the Cray Y-MP, executing a sequential version of the same application.

Table 1
NAS Parallel Benchmarks: Kernel Characteristics

| Benchmark Code | Problem size | Memory (Mbytes) | Cray time (seconds) | Operation count |
|---|---|---|---|---|
| Embarassingly parallel | $2^{28}$ | 8 | 126 | $2.67 \times 10^{10}$ |
| V-cycle multigrid | $256^3$ | 453.6 | 22 | $3.90 \times 10^{09}$ |
| Conjugate Gradient | $2 * 10^6$ | 83.2 | 12 | $1.51 \times 10^{09}$ |
| 3-D FFT PDE | $256^2 * 128$ | 343.2 | 29 | $5.6 \times 10^{09}$ |
| Integer sort | $2^{23}$ | 248.8 | 11 | $7.81 \times 10^{08}$ |

The five NPB kernels, with the exception of the embarassingly parallel application, are all highly communication intensive when parallelized for message passing systems. As such, they form a rigorous suite of quasi-real applications that heavily exercise system facilities and also provide insights into bottlenecks and hot-spots for specific distributed memory architectures. In order to investigate the viability of clusters and heterogeneous concurrent systems for such applications, the NPB kernels were ported to execute on the PVM system. Detailed discussions and analyses are presented in [15, 16]; here we describe our experiences with two representative kernels, using Ethernet and FDDI based clusters.

The V-cycle multigrid kernel involves the solution of 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 PVM version of this application was derived by substantially modifying an Intel hypercube version; data partitioning along one dimension of the grid, maintaining shadow boundaries, and performing nearest neighbor communications. Several optimizations were also incorporated, primarily to maximize utilization of network capacity, and to reduce some communication. Results for the multigrid kernel under PVM are shown in Table 2.

From the table it can be seen that the PVM implementation performs at good to excellent levels, despite the large volume of communication which accounts for upto 35% of the overall execution time. It may also be observed that the communications bandwidths obtained, *at the application level*, are a significant percentage of the theoretical limit for both types of networks. Finally, the eight processor cluster achieves one-half the speed of a single processor of the Cray Y-MP, at an estimated one fourth of the cost.

The Conjugate Gradient kernel is an application that approximates the smallest eigenvalue of a symmetric positive definite sparse matrix. The critical portion of the code is a matrix-vector multiplication, requiring the exchange of sub-vectors in the partitioning

Table 2
V-cycle Multigrid: PVM timings

| Platform | Time (secs) | Communication volume | Communication time (secs) | Bandwidth (KB/sec) |
|---|---|---|---|---|
| 4*IBM RS6000/550 (Ethernet) | 293 | 96MB | 101 | 973 |
| 4*IBM RS6000/550 (FDDI) | 185 | 96MB | 18 | 5461 |
| 8*IBM RS6000/320 (FDDI) | 110 | 192MB | 32 | 6144 |
| Cray Y-MP/1 | 54 | — | — | — |

scheme used. In this exercise also, the PVM version was implemented for optimal performance, with modifications once again focusing on reducing communication volume and interference. Results from our experiments are shown in Table 3.

Table 3
Conjugate Gradient: PVM timings

| Platform | Time (secs) | Communication volume | Communication time (secs) | Bandwidth (KB/sec) |
|---|---|---|---|---|
| 4*IBM RS6000/550 (Ethernet) | 203 | 130 | 124 | 1074 |
| 4*IBM RS6000/550 (FDDI) | 82 | 130MB | 24 | 5433 |
| 16*Sun Sparc SS1+ (Ethernet) | 620 | 370MB | 360 | 1074 |
| Cray Y-MP/1 | 22 | — | — | — |

This table also exhibits certain interesting characteristics. Like the multigrid application, the conjugate gradient kernel is able to obtain near theoretical communications bandwidth, particularly on the Ethernet, and a four processor cluster of high-performance workstations performs at one-fourth the speed of a Cray Y-MP/1. Another notable observation is that with an increase in the number of processors, the communication volume increases, thereby resulting in lowered speedups. Our results from these two NPB kernels indicate both the power and the limitations of concurrent network-based computing — i.e. that with high-speed, high-capacity networks, cluster performance is competitive with that of supercomputers; that it is possible to harness nearly the full potential and capacity of processing elements and networks; but that scaling, load imbalance, and latency limitations are inevitable with the use of *general purpose* processors and networks that most cluster environments are built from.

### 4.2. Polymer Chains and Scale-Invariant Phenomena

The particular problem studied in this work is one in which some fundamental aspects of the statistical mechanics of polymer solutions [17] are investigated. In this experiment, we focus on a linear chain which has a restricted interaction with the medium; that is, there are forbidden regions (infinite energy barrier), and the chain is confined to other parts of the medium. If the forbidden regions occur randomly and the minimum length scale of these is much smaller than the size of the polymer in homogeneous media, then this problem can be modeled, after suitable graining, by a self-avoiding walk (SAW) on a randomly diluted lattice. Thus, we first create a *percolation cluster* [18] on a finite grid (of size $L^3$) by randomly diluting the grid (i.e., removing sites as inaccessible to the chain) with probability $1-p$. The remaining sites then form connected components called *clusters*. Above a certain threshold $p_c$, there exists one cluster that spans the grid from end to end, and this is the cluster of interest. On this *disordered* cluster, a starting point of a SAW is chosen randomly, and then *all* SAWs of a predetermined number of steps $N$ are generated by a depth-first type search algorithm. At each of the $N$ steps, various conformational properties of the chain are measured, such as the moments of the end-to-end distance $R_N$ and radius of gyration $S_N$ and those of the number of chains $C_N$. These are then averaged over the ensemble of SAWs on the particular disorder configuration. This is repeated for a large number of disorder configurations, and finally both linear and logarithmic means are calculated over the disorder ensemble.

The polymer simulation problem is typical of most Monte Carlo problems in that it possesses a simple repetitive structure. The main routine initializes various arrays for statistics and calls a slave routine which computes samples, and periodically communicates them to the monitor(s). A few hours of effort were required in parallelizing the original code using one of a variety of Eclipse templates. In eleven different experiments, each corresponding to a particular network configuration of arbitrarily chosen machines, we used between 16 and 192 geographically dispersed processors; results from these experiments are reported in Table 4.

Table 4
Polymer Physics Application: Eclipse timings (seconds)

| # of procs | Sun SS | IBM RS6000 | Sun SS + RS600 | Intel i860 | SS + IBM + i860 | Average | Equiv Cray |
|---|---|---|---|---|---|---|---|
| 15 | 7101 | 1992 | 9226 | | 6106 | 12735 | |
| 31 | 3194 | 1366 | | 4661 | | 3074 | 13250 |
| 63 | | | 1015 | 2108 | | 1516 | 11552 |
| 127 | | | 492 | 893 | | 692 | 10526 |
| 191 | | | | | 574 | 574 | 11493 |

The parallelization has made an otherwise impossible interactive experimentation process possible. Previous exercises conducted on the CRAY Y-MP were only permissible in batch mode, because of the computing time and cost involved, and allowed for little interactive experimentation. Further, limited CRAY access time made the entire experimentation process difficult; restrictions that many researchers encounter on supercomputers as well as on MPP's. In contrast, our experience is that a computing environment flexibly allowing for computation nodes that are SUN/IBM workstations, Intel i860 nodes, IPSC/2 nodes or Sequent processors improves the entire interactive experimentation process by an order of magnitude. Moreover, if a subset of nodes fail, or a subnetwork goes down, the rest of the computation remains active, resulting in increased computation time but a graceful degradation of service.

## 5. DISCUSSION

Our experiences with PVM and Eclipse have been very valuable. From the research perspective, the important issues in heterogeneous network computing have been highlighted, and the benefits and limitations of this paradigm have been demonstrated. The project has also provided a deeper insight into important research issues that warrant further investigation and has led to the evolution of novel techniques that are promising and should be pursued further. Like many experimental research projects, PVM and Eclipse have also had an important side-benefit; they have produced software systems that are pragmatic and robust enough for actual production quality use by applications in a variety of domains. From the point of view of applications developers, PVM and Eclipse have provided a cost-effective and technically viable solution for high performance concurrent computing, as indicated by the large number of installations adopting these systems, often as their primary concurrent computing facility. Both these facets of the PVM and Eclipse projects have encouraged us to continue research and development, both on aspects that enhance the effectiveness and efficiency of the systems software, as well as on new features and techniques. We describe below, two new projects that we have undertaken to improve the efficacy of network-based concurrent computing.

### 5.1. Parallel I/O

In the rapidly evolving field of heterogeneous concurrent computing, and even in conventional parallel processing, an overwhelmingly large proportion of the focus has thus far been on "standalone", "in-core", problems mostly of a scientific or mathematical nature. However, as this model of computing matures and becomes an increasingly viable technology, the need to incorporate features other than fast computation and communication will become apparent — the most pressing of these is likely to be an input/output framework, preferably one that is itself parallel in nature. Parallel I/O will not only be required by scientific applications that manipulate large data sets or require out-of-core computations, but also by non-scientific applications, especially those in transaction processing, distributed data bases, and other high-performance commercial applications.

Motivated by this reasoning, we have embarked on a project to provide parallel I/O facilities in network-based concurrent computing systems, with PVM as the first target environment. PIOUS [19] is an input/output system that provides process groups access to permanent storage within a heterogeneous network computing environment. PIOUS is a parallel distributed file server that achieves a high-level of performance by exploiting the combined file I/O and buffer cache capacities of multiple interconnected computer systems. Fault tolerance is achieved by exploiting the redundancy of storage media.

To better support parallel applications, PIOUS implements a parallel access file object called a *parafile* and provides varying levels of concurrency control for process group members. PIOUS is a file service built on top of existing file systems and accessed via library calls. PIOUS is itself implemented as a group of cooperating processes within (an enhanced version of) the PVM distributed computing framework. Because of its modular design and utilization of existing standards, PIOUS is easily ported to other distributed computing environments providing similar functionality. An alpha version has been implemented and preliminary results, in our evaluation of both functionality and performance, is very encouraging.

### 5.2. Threads-based concurrent computing

At present, most heterogeneous network-based concurrent computing systems, including PVM and Eclipse, support the process-based model of computation. In other words, the unit of parallelism is a process; on the other end of the spectrum are traditional shared-memory and vector computers where parallelism can be at the loop or even instruction level. While the process based model has many advantages, there are compelling reasons to consider network concurrent computing using a finer granularity, if this is possible without sacrificing efficiency.

Our preliminary analyses suggest that an attractive option is a model where threads, or light-weight processes are the units of parallelism. We are investigating a scenario where processes, instead of possessing one thread of control, would be composed of multiple threads; threads within a process would cooperate via their common, shared, address space, while message passing would be used for communication between threads on different processes or processors. This scheme is advantageous from several viewpoints. First, there is a trend among manufacturers of almost all workstations and general purpose systems towards multiprocessing on a small scale — i.e. systems with 2 to 8 CPU's are becoming increasingly common. In the conventional use of systems such as PVM, multiple processes may execute on these systems, but will communicate via (relatively expensive) message passing, even though physical shared memory is available. With a threads approach, multiple light-weight processes on one physical machine with several CPU's would interact via shared memory, thus reducing communication overheads significantly.

In the above situation as well as on single-CPU machines, threads have another important benefit *viz.*, they are a natural and effective mechanism for latency hiding. In other words, multiple threads within a single process have the potential for increased CPU utilization (and therefore better overall application performance) because when a thread is blocked on an external communication activity, other threads could continue to perform useful work. Finally, threads are also attractive from the programming point of vies for applications that are best expressed in terms of fine-grained computations, e.g. Ising model computations, branch-and-bound algorithms, etc. Currently, programming such applications requires extensive manual housekeeping and data/control structure management; a threads based model would significantly enhance the software engineering aspect of such applications. We are currently experimenting with such a threads-based model and expect preliminary results to be available soon.

### 5.3. Other directions

In addition to the two major research efforts outlined above, several other directions are also being pursued in the context of the PVM and Eclipse systems. An effective debugging interface and mechanism is one; a preliminary concurrent debugging tool that permits simultaneous breakpoints and single stepping is under development. Fault tolerance is another important feature that is being pursued in the Eclipse system, which already provides a simple mechanism for limited failure resilience. Finally, graphical interfaces, for enhancing the development activity as well as for monitoring and profiling the execution of network-based concurrent environments are being researched and developed.

## REFERENCES


1. R. F. Freund and H. J. Siegel (eds.), IEEE Computer Special Issue on Heterogeneous Processing, 26(6), June 1993.
2. V. S. Sunderam and R. F. Freund, J. Parallel and Distributed Computing, Special Issue on Heterogeneous Processing, (to appear) January 1994.
3. R. F. Freund, SuperC or Distributed Heterogeneous HPC, 2(4):349-355, 1991.
4. J. Potter, Associative Computing, Plenum Publishing, New York, 1992.
5. L. H. Turcotte, A Survey of Software Environments for Exploiting Networked Computing Resources, Technical Report, ERCCFS, Mississippi State University, June 1993.
6. D. Y. Cheng, A Survey of Parallel Programming Languages and Tools, NAS Systems Division Technical Report RND-93-005, NASA Ames Research Center, March 1993.
7. L. Patterson, et. al., Construction of a Fault-Tolerant Distributed Tuple-Space, Proc. 1993 Symposium on Applied Computing, Indianapolis, February 1993.
8. D. Gelernter, Domesticating Parallelism, IEEE Computer, 19(8):12-16, August 1986.
9. J. Boyle, et. al., Portable Programs for Parallel Processors, Holt, Rinehart, and Winston, 1987.
10. R. Hempel, The ANL/GMD MAcros (Parmacs) in Fortran for Portable Parallel Programming Using Message Passing, GMD Technical Report, November 1991.
11. V. S. Sunderam, PVM : A Framework for Parallel Distributed Computing, Journal of Concurrency: Practice and Experience, 2(4):315-339, December 1990.
12. V. Rego and V. S. Sunderam, Experiments in Concurrent Stochastic Simulation: The ECLIPSE Paradigm, Journal of Parallel and Distributed Computing, 14(1):66-84, January 1992.
13. H. Nakanishi, V. Rego, and V. S. Sunderam, Superconcurrent Simulation of Polymer Chains on Heterogeneous Networks, Proceedings – Fifth IEEE Supercomputing Conference, Minneapolis, November 1992.
14. D. H. Bailey, et. al., The NAS Parallel Benchmarks, International Journal of Supercomputer Applications, 5(3):63-73, Fall 1991.
15. S. M. White, Implementing the NAS Benchmarks on Virtual Parallel Machines, Emory University M. S. Thesis, April 1993.
16. S. M. White, A. Anders, and V. S. Sunderam, Performance Optimization of the NAS NPB Kernels under PVM, Proc. Distributed Computing for Aeroscience Applications, Moffett Field, October 1993.
17. Flory, P. J., Statistical Mechanics of Chain Molecules, Interscience, New York, 1969.
18. Stauffer, D., Introduction to Percolation Theory, Taylor and Francis, London, 1985.
19. S. A. Moyer, PIOUS: A Parallel Input/Output System, Emory University White Paper, July 1993.