Opening the Door to Heterogeneous Network Supercomputing

BY: ADAM BEGUELIN, OAK RIDGE NATIONAL LABORATORY AND UNIVERSITY OF TENNESSEE; JACK DONGARRA, OAK RIDGE NATIONAL LABORATORY AND UNIVERSITY OF TENNESSEE; AL GEIST, OAK RIDGE NATIONAL LABORATORY; ROBERT MANCHEK, UNIVERSITY OF TENNESSEE; AND VAIDY SUNDERAM, EMORY UNIVERSITY

Heterogeneous networks of computers are becoming commonplace in high-performance computing. Systems ranging from workstations to supercomputers are linked by high-speed networks. Until recently, each computing resource on the network remained a separate unit. Now, however, 20 to 30 institutions worldwide are writing and running truly "heterogeneous" programs utilizing multiple computer systems to solve applications.

The ultimate success of heterogeneous computing will depend largely on work such as that being conducted by the Heterogeneous Network Project (HNP). HNP is a collaborative effort by researchers at Oak Ridge National Laboratory, the University of Tennessee and Emory University to design and develop software to facilitate heterogeneous parallel computing.

Two components have been constructed so far. Parallel Virtual Machine (PVM) is a set of tools designed with heterogeneity and portability as the primary goals. It is one of the first software systems that allows machines with widely different architectures and floating-point representations to work together on a single computational task. PVM can be used on its own or as a foundation upon which other heterogeneous network software can be built.

The second component, HeNCE (for Heterogeneous Network Computing Environment), is being built on top of PVM. HeNCE simplifies the tasks of writing, compiling, running, debugging and analyzing programs on a heterogeneous network.

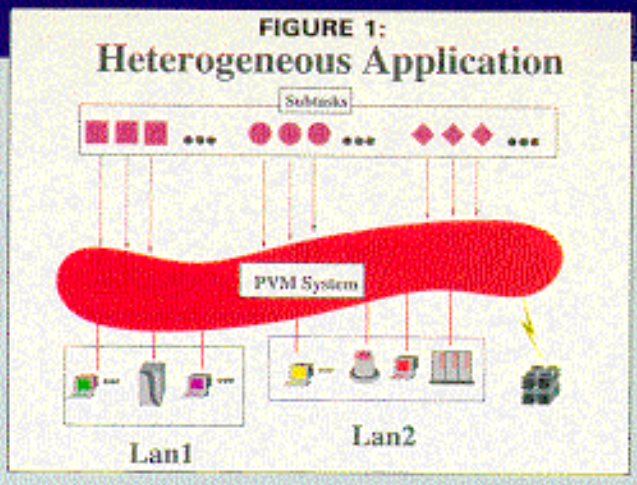
The goal of the Heterogeneous Network Project is to make network computing accessible to scientists and engineers without the need for extensive training in parallel computing. PVM and HeNCE will allow researchers to use those resources best suited for a particular phase of the computation.

PVM

PVM provides a unifying computational framework for a network of heterogeneous parallel and serial computers. An architectural overview of PVM is shown in Figure 1. The user views PVM as a loosely coupled, distributed-memory computer programmed in C or Fortran with message-passing extensions. The hardware that composes the user's personal PVM may be any Unix-based machine accessible over some network and on which the user has a valid login.

PVM may be configured to contain various machine architectures including sequential processors, vector processors and multicomputers. The present version of the software has been tested with various combinations of the following machines: Alliant FX/8, CRAY Y-MP, DECstation, IBM RS/6000, Intel iPSC/2, Intel iPSC/860, Sequent Symmetry, Silicon Graphics IRIS, SPARCstation, Sun 3 and Thinking Machines CM-2. In addition, the user can port PVM to new architectures by modifying a generic "makefile" supplied with the source and recompiling.

FIGURE 1:
Heterogeneous Application



Using PVM, each user can configure his own parallel virtual computer, which can overlap with other users' virtual computers. Configuring a personal parallel virtual computer involves simply listing the names of the machines in a file that is read when PVM is started. Several different physical networks can co-exist inside a virtual machine. For example, a local Ethernet, Internet and a fiber-optic network can all be a part of a user's PVM configuration. While each user can have only one virtual machine active at a time, PVM is multitasking. Thus, several applications can run simultaneously on a parallel virtual machine.

The PVM package is small (less than 300 KBytes of C source code) and easy to install. It needs to be installed only once on each machine to be accessible to all users. Moreover, the installation does not require special privileges on any of the machines.

Application programs that use PVM are composed of subtasks at a moderately large level of granularity. The subtasks can be generic serial codes, or can be specific to a particular machine. In PVM, resources may be accessed at three different levels: the "transparent" mode, in which subtasks are automatically located at the most appropriate sites; the "architecture-dependent" mode, in which the user may indicate specific architectures on which particular subtasks are to execute; and the "machine-specific" mode, in which a particular machine may be specified. Such flexibility allows different subtasks of a heterogeneous application to exploit particular strengths of individual machines on the network.

The PVM user-interface requires that all message data be explicitly typed. Support for operating in a heterogeneous environment is provided in the form of special constructs that selectively perform machine-dependent data conversions. All communication done between PVM processes uses the external data representation standard (XDR), thus allowing machines with different integer and floating-point representations to pass data.

Applications access PVM resources via a library of standard interface routines. These routines allow the initiation and termination of processes across the network, as well as communication and synchronization between processes. Communication constructs include those for the

exchange of data structures and high-level primitives such as broadcast, barrier synchronization 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; further, any process may communicate and/or synchronize with any other.

HeNCE

While PVM provides low-level tools for implementing parallel programs, HeNCE provides the programmer with a higher-level environment for using heterogeneous networks. The HeNCE philosophy of parallel programming is to have the programmer explicitly specify the parallelism of a computation and to automate — as much as possible — the tasks of writing, compiling, executing, debugging and analyzing the parallel computation. Central to HeNCE is an X-Windows interface that the programmer uses to perform these functions.

The HeNCE environment contains a *compose* tool that allows the user to specify parallelism by drawing a graph of the parallel application. If an X-Windows interface is not available, then textual graph descriptions can be input. Each node in a HeNCE graph represents a procedure written in either Fortran or C. HeNCE is designed to enhance procedure reuse. The procedure can be a subroutine from an established library or a special-purpose subroutine supplied by the user.

Five types of arcs can be used in a HeNCE graph:

- A *dependency* arc from one node to another represents the fact that the tail node of the arc must run before the head of the arc. A section of the graph can be executed or bypassed based on a conditional statement that will be evaluated at run time.
- A *conditional dependency* arc is used to specify the conditional statement.
- A variable *fan-out* (and subsequent *fan-in*) construct is available while composing the graph. The width of the fan-out is specified as an expression that is evaluated at run time. This construct is similar to a parallel-do construct found in several parallel Fortrans.
- *Loops* around subgraphs can be used in the graph to execute a variable number of times.
- *Pipelining* can be used for connecting sections of the graph. In pipelined sections, when a node finishes with one set of input data, it reruns with the next piece of pipelined data.

Once the dynamic graph is specified, a *configuration* tool in the HeNCE environment can be used to specify the configuration of machines that will compose the user's parallel virtual machine. The configuration tool also assists the user in setting up a cost matrix. The cost matrix allows the user to describe what machine can perform what task; priority can be given to certain machines. HeNCE will use this cost matrix at run time to determine the most effective machine on which to execute a particular procedure in the graph.

The HeNCE environment also contains a *make* tool that performs three tasks. First, the parallel program is automatically generated, using PVM calls for all the communication and synchronization required by the application. The node procedures for the various heterogeneous architectures then automatically are compiled. Finally, the executable modules are installed on the particular machines in the PVM configuration.

The *execute* tool in the HeNCE environment starts up the requested PVM and begins execution of the application. During execution, HeNCE automatically maps procedures to machines in the heterogeneous network based on the cost matrix and the HeNCE graph. Trace and scheduling information saved during the execution can be displayed in real time or replayed later.

To further help the programmer, the HeNCE environment has a *trace* tool that allows visualization of the parallel run. The trace tool is X-Windows-based and consists of two windows. One window shows a representation of the network and machines underlying PVM. Icons of the active machines are illuminated with different colors, depending on whether they are computing or communicating. Under each icon is a list of the node procedures mapped to this machine at any given instant. The second window displays the user's graph of the application, which changes dynamically to show the actual paths and parameters taken during a run. The nodes in the graph change colors to indicate the various activities going on in each procedure.

Results

One of the computational Grand Challenges being addressed at Oak Ridge National Laboratory is the calculation of the electronic structure of superconductors (see *Supercomputing Review*, October 1990). This application has been modified to run using PVM. Initially, a heterogeneous network of workstations was used to achieve execution rates exceeding 250 MFLOPS. This performance is comparable to the performance of the application on a single processor of a CRAY Y-MP.

A second set of experiments was then devised to show the capability of PVM to connect several supercomputers. In one test, an Intel iPSC/860, a CRAY X-MP and an IBM RS/6000 were configured into a metacomputer. In another test, two CRAY Y-MP/8s and a CRAY Y-MP/2 were configured into a metacomputer. The PVM version of the electronic structures code ran on both of these metacomputers, but the input data set was too small to drive either of the metacomputers to near their peak rates. Nevertheless, our experiments do demonstrate the viability of using a collection of supercomputers.

Several other PVM applications are being developed at Oak Ridge National Laboratory in areas such as molecular dynamics, statistical modeling and climate modeling.

Current Status and Availability

PVM was originally developed at Oak Ridge National Laboratory two years ago and was made publicly available in March of this year. The current version, Version 2.2, is available through netlib. To obtain a description of the components available, including the PVM User's Guide and source code, send e-mail to netlib@ornl.gov with the message: send index from pvm. A prototype version of HeNCE is expected to be available near the end of summer.

PVM and HeNCE open the door to heterogeneous network supercomputing. It is time for researchers to step inside and explore this new area of computing. SR