The Integrative Role of COW's and Supercomputers in Research and Education Activities

Don Morton, Ganesh Prabu, Daniel Sandholdt, Lee Slater
Department of Computer Science
The University of Montana
{*morton | gprabu | marist92 | lslater*}*@cs.umt.edu*

ABSTRACT: Experiences of researchers and students are presented in the porting of code between a cluster of Linux workstations at The University of Montana and the Cray T3E at the Arctic Region Supercomputing Center. We test the thesis that low-cost workstation environments may be utilized for training, and to develop and debug parallel codes which can ultimately be moved to the Cray T3E with relative ease for realizing high performance gains. We further present ideas on how the computing environments of supercomputers and COW's might benefit from more commonality.

1 Introduction

In the early 1990's, several pivotal events, revolving about the availability of low-cost commodity processors, occurred which forever changed the nature of scientific and high-performance computing. The improving line of Intel x86 architectures and the increasing use of the growing Internet encouraged the development of the Linux operating system, ensuring that anybody could own a Unix workstation for home or business use. During this same time-frame, researchers at Oak Ridge National Laboratory (and U. Tennessee at Knoxville) initiated development of PVM, a highly-portable library and environment for the construction of message-passing programs on clusters of workstations (COW's). In addition to supporting well-known architectures, PVM supported Linux as early as 1993. Also during this time, Cray Research, Inc. introduced its MPP Cray T3D based on the low-cost DEC Alpha chip, running its own version of Unix. In addition to supporting its own CRAFT environment, the T3D supported PVM (though somewhat different from the "standard" PVM).

The convergence of these tracks marked the beginning of a healthy, complementary relationship between COW environments and state of the art supercomputers such as the Cray MPP series. Though some Linux and Cray supporters have felt somewhat threatened by the presence of a "rival," the reality is that both environments hold important niches, and an integration of the environments may easily result in a win-win situation for everyone.

Our thesis is that the COW environment is well-suited for training users in concepts of parallel programming and in the development (which often means a lot of debugging) of parallel codes, all at relatively low equipment cost. Though some Beowulf clusters have achieved remarkable performance benchmarks, we believe the majority of parallel programmers have insufficient resources to construct such a "supercomputer." For this reason, powerful machines such as the Cray T3E will always be high-demand machines for large-scale production runs. Such environments often favor the batch user, and, in our experience, supercomputer centers encourage such long batch jobs in order to achieve high CPU utilization. Though these centers try to keep a few interactive PE's available, it is often difficult for several programmers to test their code (or debug) in an interactive manner. When it comes time to hold group training sessions in parallel computing, it becomes even more difficult. So, we maintain that COW environments should be used for this sort of interactive parallel computing, allowing trainees and developers the interactivity (and sometimes lack of network delays) they need, while freeing the MPP CPU's for the large-scale batch jobs that utilize these expensive resources best.

In this paper, we accept the above thesis and begin to explore the issues of integrating COW and MPP supercomputer resources so that users may migrate between the environments as effortlessly as possible. We begin with a case-study – a recent graduate-level course in parallel computing in which students are initially trained on a Linux cluster and, towards the end of the semester port their codes to the Cray T3E. Then, we discuss some of the research activities that have been taking place in the past four years using both Linux and Cray MPP systems. Finally, we present our own opinions on how COW's and supercomputers may be better integrated for the benefit of all.

2 Computing Environments

The author has been working with Linux since 1991 and with the Cray T3D/E series since 1993. With funding from the National Science Foundation, a poor

man's supercomputer (Linux cluster) was constructed in 1995 to provide a local environment for education and continued research in parallel computing. This funding supported summer residencies at the Arctic Region Supercomputing Center. Therefore, there has been constant emphasis placed on developing codes that run on both the Linux cluster and the Cray T3D/E, and to create similar programming environments. Sometimes this means that we don't use certain features unless they're available on both platforms. For example, until recently there was no Fortran 90 compiler on the Linux cluster, so any Fortran code was written in Fortran 77 style for both platforms. Likewise, although *shmem* is a powerful programming tool on the Cray MPP series, we do not use it in "portable" codes (though, sometimes we use conditional compilation so that if the program is compiled on the Cray, *shmem* is used, otherwise MPI or PVM).

The Linux cluster at The University of Montana (see Figure 1) consists of nine 100MHz Pentiums. One machine, possessing 128 Mbytes of memory, acts as an account and file server for the other eight machines, each possessing 64 Mbytes of memory. The machines are connected by a 100Mb Fast Ethernet. Although users may log into any machine (via NIS) and see their file systems (via NFS), the primary mode of use is to log in to the server. Since users have valid accounts on each machine through NIS, and their applications can be seen by each machine via NFS, there is no need to transfer an application to each machine for parallel computing. Thus, in many respects, users of the system can run programs transparently, much as they would on a T3E. The system supports PVM, MPI and recently, Portland Group's HPF.

The Cray T3E at the Arctic Region Supercomputing Center consists of 272 450-MHz processors. Until recently, the network connections to/from ARSC were very slow, making remote use of the system frustrating, which of course increased the need to work on code locally. Outside network connections have improved substantially since April 1999, but, due to very high CPU utilization, it still is often more convenient to perform training and development activities on the local Linux cluster.

3 Case Study – Parallel Programming Course

In the Spring 1999 semester, a graduate-level (primarily masters degree students) course in parallel processing was offered at The University of Montana with the intent of giving students an applied, hands-on introduction to parallel computing. One expected outcome of the course was to test the thesis stated in the Introduction. Students would be initially introduced to parallel programming on the existing cluster of Linux workstations, using PVM, MPI, and High Performance Fortran. Then, through the support of the Arctic Region Supercomputing Center, students would use accounts on the Cray T3E to execute and compare programs that were previously run on the Linux cluster.

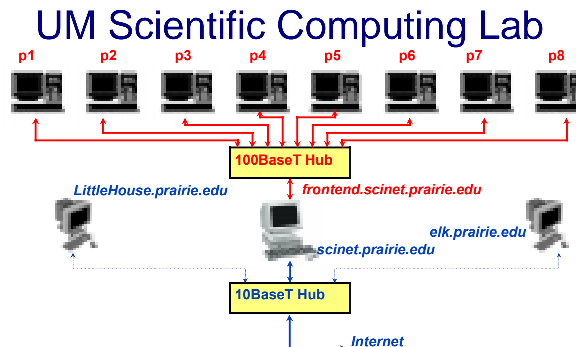

Figure 1: University of Montana Linux cluster.

One of the first parallel programming activities that students engaged in was the PVM implementation of the n-body problem. In its simplest form, we considered a set of masses in two-dimensions and calculated the net gravitational force applied to each particle. A quick and dirty parallel algorithm was introduced so that students could gain experience in writing their first parallel program that did something "useful." For most students, this was a difficult task, and they spent a lot of time learning how separate executables could interact in a nondeterministic fashion, plus they began to familiarize themselves with the mechanics of launching parallel executables. Such difficulty would have occurred on either our Linux cluster or the T3E, so it was more appropriate to use local resources for this. Though PVM doesn't seem as popular as it did in the early 1990's (a time when it had little competition), it certainly appears in much legacy code, and it served to introduce students to something other than SPMD.

Through the semester, students were also introduced to MPI and HPF, and were required to implement the n-body problem in each of these paradigms. Since the students had already been exposed to a message-passing implementation with PVM, an MPI implementation was simpler for them, and they enjoyed the higher-level constructs provided by that library. Of course, those students who didn't constrain themselves to SPMD programming with PVM had to modify their codes extensively when using MPI. Finally, students had little difficulty modifying their programs for HPF, but got a little confused playing around with HPF compiler directives for optimizing their code.

At this point, students had written and executed programs in PVM, MPI, and HPF, all on the Linux cluster, and were becoming somewhat proficient in writing simple parallel programs. The next step was to introduce them to readily-available performance analysis tools that would run on the Linux cluster, and some that would also be available on the T3E. Students were first introduced to *xpvm* and *xmpi*, which provide users with a graphical interface for running, then viewing tracefiles. Both tools are freely available and support Linux environments. Although it is assumed that a user could generate a tracefile on the T3E with a PVM or MPI program, then use *xpvm* or *xmpi* on another machine to view it, this wasn't attempted, so it remains merely an assumption.

Students were also introduced to *vampir* and *pgprof*. Vampir, a performance analyzing tool, and Vampirtrace, a library of functions for generating trace files, is a commercial product for use with MPI programs. The developing company, Pallas, was kind enough to allow us to use the product for evaluation purposes. The vampir tools had the great advantage of being portable between the Linux cluster and the T3E. For example, one could generate a trace remotely on the T3E, then transfer it to a local machine for viewing. *Pgprof*, Portland Group's profile analyzer for HPF programs was distributed with the PGHPF package for the Linux cluster, and we found that it, too, would allow for trace generation on one platform with subsequent viewing on another platform.

As a midterm exam project, students were required to develop performance models for a parallel Jacobi code (which they were provided with) implemented with MPI, then test their "theoretical" performance against actual execution on our Linux cluster. This forced them to deal with complexity analysis of parallel algorithms while providing them with more experience in running codes for performance analysis.

In addition to the previous work, each student pursued an individual project of their own choosing to either develop and analyze parallel code for a problem of interest, or parallelize existing code and then analyze the new code's performance. Topics pursued by the students included dictionary algorithms, machine learning, image processing, molecular modeling, implementation of Java-based PVM, parallelization of an existing plasma code, and finite element modeling. Students developed and tested their code on the Linux cluster, but several moved their code to the T3E for further analysis.

The final six weeks of the course were devoted to providing students with experience in dealing with the issues of portability between the Linux cluster and the Cray T3E in an effort to test our thesis and to provide material for this paper. Two lab sessions were developed – the first session was simply an introduction to the use of the T3E environment, in which students were required to compile previously-ported code, and then execute, in both interactive and batch (NQS) modes. Most students found this fairly straightforward. A second lab session of six hours was held on a Saturday, in which the students were given favorable access to the PE's at ARSC. This lab session consisted of five primary tasks (described below), and it was believed that the majority of the students would be able to finish most of the work within the six hours. In fact, only one student came close to finishing in this time frame, while others devoted a considerable amount of extra time.

Three students, listed as co-authors of this paper, performed some additional tests as special projects described later.

3.1 Description of Linux/T3E Laboratory

3.1.1 Conversion of Linux PVM *n*-body PVM Program for T3E Execution

Previous to this assignment, students were introduced to the differences in standard PVM (referred to as network PVM by Cray) and Cray MPP PVM. Since network PVM relies on the spawning of new tasks from an original task, and MPP PVM requires an SPMD mode of programming, does not support the *pvm_spawn()* operation, does not support the concept of PVM tasks having parent tasks, and so on, producing a portable code requires several tricks which are incorporated through conditional compilation (see Figure 2 for the relevant code excerpts). In general, this requires setting up a network PVM code that, once all processes have been spawned, assigns a logical task number to each process through group operations, and runs identical tasks in SPMD mode. On Cray MPP platforms, tasks are not spawned, and cannot get their logical PE number by joining a global group, as in network PVM.

```
#ifdef _CRAYMPP
// In Cray MPP, the "global" group is
//indicated by null pointer
#define GROUPNAME (char *) 0
#else
#define GROUPNAME "alltasks"
#endif

….


#ifdef _CRAYMPP
// Cray MPP does not support joining a
//"global" group, so we simply
```

```
// use the Cray-specific routine for getting
the PE number
mype = pvm_get_PE(mytid);
#else
mype = pvm_joingroup(GROUPNAME);
#endif

…..

// Determine total number of tasks
// In Cray MPP, we get this from command line
//using pvm_gsize(), in other systems, we
//specify, either in the program, on command
// line, or in a file
#ifdef _CRAYMPP
ntasks = pvm_gsize(GROUPNAME);
#else
ntasks = 4;
#endif

……..

#ifndef _CRAYMPP
// This is not executed for Cray MPP PVM -
//pvm_spawn() is not implemented - all tasks
//startup SPMD at beginning
if(mype == 0)  // I'm the master, spawn the
others
   info = pvm_spawn(argv[0], (char**) 0,
PvmTaskDefault, (char*) 0,
          ntasks-1, &tid_list[1]);
#endif

……

// Wait for everyone to join group before
//proceeding
info = pvm_barrier(GROUPNAME, ntasks);

……

// Get TID of everyone else in group
for(i=0; i<ntasks; i++)
   tid_list[i] = pvm_gettid(GROUPNAME, i);

// At this point, we know the TID's of all
//tasks, so we can communicate
```

Figure 2: Code excerpts from portable PVM program.

The goal of this task was to have the students port their Linux PVM *n*-body codes for execution on the Cray T3E. Once instructed in how to modify a code for this type of portability, most students found it to be relatively painless. However, the differences between Cray MPP and network PVM have been an impediment to portability since the introduction of the Cray T3D, and users of PVM need to be aware of this.

Additionally, Cray MPP compilers seem to be less forgiving than others (e.g. Gnu compilers) of simple programmer omissions, such as failure to initialize a variable. Other problems included the difference in the default handling of incorrect arguments within the *atan2()* function – Gnu compilers would simply return a zero from this function, but Cray MPP compilers would abort and dump core. Of course, these problems are inherent in any code, not just PVM, ported from a Linux cluster to the T3E, and one can argue that the Cray MPP environment simply forces programmers to adhere to good programming practices. Most of these problems are easily solved through the use of compiler flags. The point is that this is a problem area in the "coupling" of Linux and T3E systems, and needs to be addressed in such a scenario. The majority of the students encountered some problems in moving their code from the Linux system to the T3E. However, in almost all cases, a problem in porting was a direct result of a programmer error that the original system didn't catch. In general, experienced students found the transition from Linux to the T3E to be rather straightforward, whereas those students with less experience had a very difficult time.

Some representative timings on the Linux cluster and the Cray T3E are displayed below in Table 1. The wide variation in timings results from different implementations of a solution, and often, from varying loads on the Linux cluster. The timings are from the calculation of forces on N=1000 particles, with two and five processors. Students were also required to run larger problems on the Cray T3E so that they would appreciate the larger problem sizes facilitated by the T3E, and see the performance gains of coarse-grained problems.

Table 1: Selected Linux vs. T3E timings for PVM n-body problem.

PE's	Linux	Cray	Linux	Cray	Linux	Cray
2	5.5	1.5	126	8.2	3.8	0.79
5	3.2	1.2	1.2	2.6	1.8	0.65

3.1.2 Performance Analysis of *n*-body MPI Program (Linux and T3E)

In the next task, students were to port their Linux *n*-body MPI codes to the T3E and compare execution times. Their MPI programs used higher-level, presumably more efficient, constructs for scattering and gathering data, so any comparison of the performance with PVM is probably fruitless. Students found the port of an MPI program to be quite painless, other than the basic problems inherent in moving any code across platforms, as discussed above. Again, good programming practices can reduce such problems.

Some representative timings on the Linux cluster and the Cray T3E are displayed below in Table 2. Again, the wide variation in timings results from different implementations of a solution, and often, from varying

loads on the Linux cluster. The timings are from the calculation of forces on N=1000 particles, with two and five processors. As with the PVM component described above, students were also required to run larger problems on the Cray T3E so that they would appreciate the larger problem sizes facilitated by the T3E, and see the performance gains of coarse-grained problems.

Table 2: Selected Linux vs. T3E timings for MPI n-body problem.

PE's	Linux	Cray	Linux	Cray	Linux	Cray
2	13.8	1.7	5.4	0.72	5.4	0.60
5	12.0	1.4	3.9	0.91	1.6	0.33

3.1.3 Performance Analysis of *n*-body MPI Program Using Vampir (Linux and T3E)

In order to introduce students to performance analysis tools, and to demonstrate yet another commonality between COW's and supercomputers, Vampir was used to produce visual traces of their MPI *n*-body programs executed on both the Linux cluster and the Cray T3E. Vampir is actually made of two components – vampir, the GUI viewer, and Vampirtrace, a library of routines that are linked in to an MPI program to produce runtime traces. By linking in Vampirtrace libraries on either system, program execution will produce a trace file for viewing in the Vampir program. These traces are portable, which means that one generated on the T3E can be viewed with Vampir on another platform, and vice-versa. Students were excited about the ability to "see" their parallel program execute, and impressed with the idea that this was available on both platforms. One student remarked that the information provided was much more helpful than that from *xmpi*.

3.1.4 Performance Modeling and Analysis of MPI Jacobi Program on T3E

As part of a prior assignment, students were provided with an MPI code to solve a system of equations via the Jacobi method. The students were to construct a performance model and compare theoretical and actual execution times for the Linux cluster. This lab required that they extend their work and create a performance model for predicting T3E performance, then compare with actual execution times. Since a portable code had already been provided to them, and they had begun to gain some experience in working on both platforms, this was fairly simple for them.

3.1.5 Performance Analysis and Improvement of an MPI Code

To encourage students to think about optimization in a parallel program, they were given a fairly simple program which would read data from an input file and distribute the data equally among processors. Then, each processor would perform a series of operations on its data (statistics operations) and finally, return some results to a master process. The program was intentionally coded with inefficient communication mechanisms. For example, when the master process was to receive results from all of the other processes, it would wait for a result from PE1, then PE2, etc. Students were to produce a Vampir trace of this original program on both the Linux cluster and the T3E, as shown in Figures 3 and 4, then use this information to modify the program for less "waiting" time. Again, this program was given to them, already running on both systems, so the exercise simply allowed the students to witness the commonality of the systems.

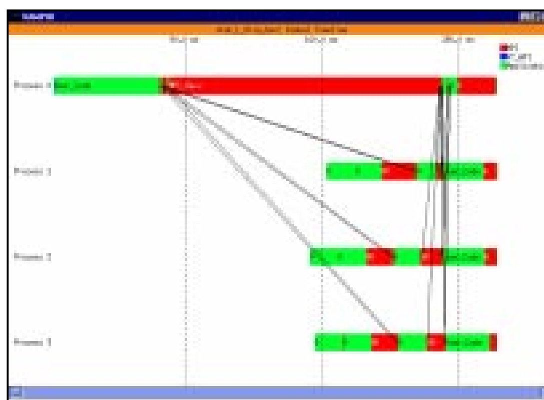

Figure 3: Vampir trace from Linux cluster.

Figure 4: Vampir trace from Cray T3E

3.1.6 Conversion of C++ MPI Jacobi Program to Fortran

As an extra project, one student was tasked with converting a C++ MPI Jacobi program provided to the class earlier in the semester, to a Fortran version for testing and evaluation on both Linux and T3E systems. Unfortunately, this exercise revealed a serious flaw in the local setup of Fortran 90 vs. MPI on the Linux

cluster. We found on the Linux cluster that we were unable to use the Portland Group's *pgf90* compiler and successfully link in MPICH library calls. MPICH was rebuilt several times with different flags in an effort to generate function names that *pgf90* would agree with, but this was unsuccessful. Although it would have been possible to use the Gnu *g77* compiler for this, it didn't contain the dynamic memory allocation features desired. Therefore, this was a scenario in which a portable code was not achieved – it was simply developed on the T3E.

Fortran codes have generally been just a little more problematic than C/C++ codes on Linux clusters. Typically, in order to insure portability, it has been necessary to use Fortran 77 as a lowest common denominator. Recently, however, Portland Group has supported Linux, providing the ability to write portable Fortran 90 and HPF codes on a wide range of systems. Though we experienced difficulties, we presume they can be remedied without too much effort. Again, this is simply a problem area that needs exploration if we are to achieve the goal of COW/workstation integration.

The C++ MPI code was converted to Fortran 90 on the T3E with minor porting problems, but, ultimately, a conversion to Fortran was successful, and timings of the C++ vs. Fortran 90 MPI codes for the Jacobi algorithm are shown in Table 3, suggesting a substantial benefit of using C++ over Fortran 90 for this particular code on the T3E.

Table 3

	N=8400		N=10080		N=12600	
PE's	C++	F90	C++	F90	C++	F90
4	1.2	6.8	1.4	10.1	NA	NA
6	0.8	4.4	1.0	6.3	1.8	10.2
8	0.6	3.1	0.7	4.6	1.3	7.4

3.1.7 Conversion of C++ MPI Jacobi Program to C++ PVM

Another student was tasked with taking the same C++ MPI Jacobi problem and converting it to PVM on both the Linux cluster and the T3E. With the experience gained from the previous assignments of the semester, it turned out to be fairly straightforward. In general, performance differences (Tables 4 and 5) between MPI and PVM on the Linux cluster were small, though there were isolated exceptions. On the T3E, at least for this small sample, it appears that PVM exhibited better performance.

Table 4: Linux timings for MPI/PVM comparison

	N=512		N=1600		N=3200	
PE's	MPI	PVM	MPI	PVM	MPI	PVM
2	0.2	0.2	1.5	0.8	6.1	3.0
4	0.2	0.2	0.8	1.0	3.2	3.4
8	0.1	0.2	0.5	0.9	2.0	2.5

Table 5: T3E timings for MPI/PVM comparison

	N=8000		N=10000		N=16000	
PE's	MPI	PVM	MPI	PVM	MPI	PVM
8	0.40	0.31	0.62	0.41	NA	NA
20	0.16	0.10	0.25	0.17	0.61	0.42
64	0.06	0.04	NA	NA	0.20	0.15

3.1.8 Porting of Linux C++ Parallel Finite Element Code to T3E

Finally, another student is pursuing an ambitious plan to write an object-oriented, parallel, adaptive finite element "toolkit." Again, the ideal development platform has been the local Linux cluster, though the intent is to ultimately implement this on the Cray T3E (the project is funded by CRI/ARSC under a University R&D grant) to encourage development of finite element codes.

To date, the student has developed software for 1D finite element situations, and his task during the course was to prototype a parallel system. The code uses extensive object oriented designs, depending on C++ features such as virtual functions, inheritance, and templates. The student performed all development on our Linux cluster using MPI, ultimately revealing the timings shown in Figure 5.

The port to the T3E was, surprisingly (to the author) relatively painless. The T3E C++ compiler doesn't by default support the local declaration of index variables within for-loops (though a compiler option provides such support), so these loops were modified. As described above, the T3E tends to be less forgiving of programmer errors, so, by porting the code, the student was able to discover some errors in his code which didn't appear when running on the Linux cluster (though under different situations, they may have revealed themselves in the future). The Cray T3E timings are shown in Figure 6. It was interesting to see that the charts of Figures 5 and 6 were quite identical, with the time scale being the only significant difference. Experience has shown, however, that communication will become more significant when moving to two and three dimensions, and the T3E will show much greater scalability in these cases.

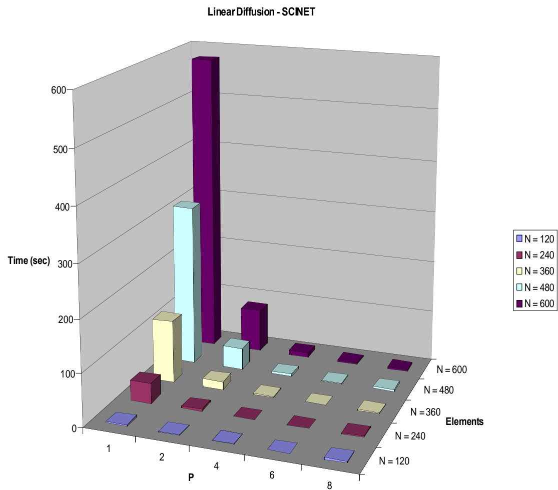

Figure 5: Linux cluster finite element timings.

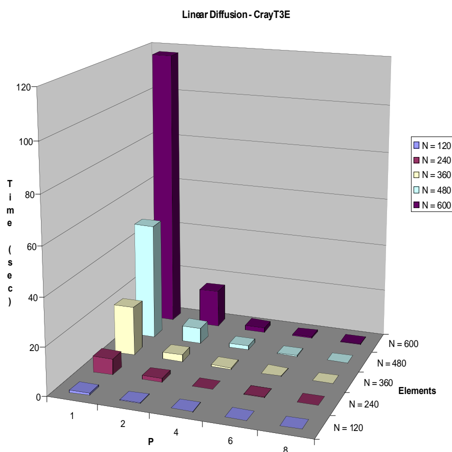

Figure 6: Cray T3E finite element timings

4 Research and Development Activities

Though the course activities described above provide a recent example of how COW's can facilitate the training aspect of high-performance computing, the cluster has in fact been used extensively, along with the Cray T3D/E in code development since 1995.

The initial application for our Linux cluster was an adaptive, parallel finite element code for fluid flow simulations, written in Fortran/PVM. It was developed on an RS600 cluster, then ported to ARSC's T3D in 1994. Back then, T3D time was easy to get, but network constraints made remote use very difficult. Once our Linux cluster was assembled, this code was ported to the system to provide a local development platform for prototyping, testing and debugging.

Other research activities have resulted from collaborations with University of Alaska's Water Research Institute to parallelize for the Cray MPP a hydrologic model for arctic ecosystems, then, to couple this hydrologic model with a parallel thermal model for execution on the Cray MPP systems. These models were written in Fortran 77, and, though a CRAFT approach was considered (and even implemented initially), portability of the code was a large issue, so MPI was utilized. In these situations, typical model runs would read in hundreds of megabytes of data, and produced just as much, which was an impediment to implementation on our small Linux cluster. However, by using very small datasets just for prototyping and development purposes, it was possible to utilize the Linux cluster extensively in the initial phases of the project. In fact, extensive debugging and analysis was necessary at times, and it was much simpler to do this work on a local workstation cluster than it would have been over a slow network on a supercomputer where we might have to wait several hours for a few PE's.

5 General Summary of COW/Supercomputer Integration Issues

The goal of integrating COW's and supercomputers should focus primarily on the creation of similar programming environments. Ultimately, code written and executed on one system should be able to run on another system without great porting problems. Users should be able to move their T3E code to a local cluster for further development and debugging, and not be overwhelmed with differences in the programming environments. Likewise, students and researchers should be able develop algorithms and codes for interesting problems on local clusters without having to deal with supercomputing centers initially, yet, when they're ready for the powerful machines, they should be able to move to such a machine with a minimal learning investment. It is our opinion that supercomputer manufacturers and supercomputing centers will benefit by the presence of numerous clusters whose sole purpose in life is to help and encourage more students and researchers to try high-performance computing, and create a larger base of experts in the field. Likewise, the supercomputing centers will benefit by not having to reserve large groups of PE's for interactive computing, and may be able to spend less time training and helping users.

The more popular and necessary tools for parallel computing – PVM, MPI, HPF – are available on a large

number of platforms, so there is already a great deal of commonality between COW's and supercomputers. Though MPI and HPF tend to be portable, the PVM differences between COW's and the Cray T3E are substantial. If we are to assume that supercomputers such as the T3E will not fully support network PVM in the near future, then the obvious solution for portable code is to insure we write it SPMD style from the beginning, and encapsulate the necessary differences within a startup routine.

Since *shmem* has proven to be so successful at reducing "communication" costs on the T3D/E, it, and other such tools should have a means to simulate their behavior via underlying PVM or MPI, so that such codes can at least be debugged and tested on COW's. Alternatively, programmers would need to be instructed in the use of preprocessor directives so that their code uses a portable message passing system on COW's, and uses *shmem* on the T3E.

The Cray T3E tends to have a simple user environment for compiling and executing parallel codes. For example, compilation of PVM and MPI programs doesn't require long flags to specify location of header files and libraries. Of course, much of this may be achieved through the creation of appropriate environment variables, and this should be implemented more on the COW environments in order to ease the task of migrating from a machine such as the T3E, where users don't have to worry about such issues. Additionally, users on the T3E can use the *mpprun* utility to run all parallel programs, whether they be PVM, MPI, or HPF. This contrasts with the typical setup on a COW where users often need to start local daemons on each machine, and they use different commands for executing different types of parallel programs. Many of these differences could be encapsulated on COW's behind scripts, and again, doing so would ease the task of migrating from the T3E.

Particularly with the presence of performance analysis tools like *Vampir* (for MPI) and *pgprof* (for Portland Groups HPF), there is already a great deal of commonality between COW/supercomputer systems. This continued dual-development should be encouraged. Of course, both of the above tools are commercial products, and may not be affordable for use in some COW environments (where low cost is often a very important factor).

With Totalview, it is possible to use a common debugging tool for MPI, PVM and HPF programs on numerous systems. This tool has been available on Cray MPP series since at least 1994, and has improved over time. It is apparently available for some COW environments, but not yet Linux. Debugging tools on Linux clusters tend to revolve about *gdb* (Gnu debugger), and it is often very difficult, and sometimes impossible to get such tools working for parallel programs. As a result, effective debugging on a Linux cluster tends to be based on *printf* statements. Availability of a common debugger (like Totalview) for Linux clusters would be highly desirable but, again, such a tool would likely be expensive.

6 Conclusions

We have provided examples and discussed ways in which COW's and supercomputer (particularly Linux vs. Cray MPP) platforms can play complementary roles in high performance computing. The insight of software developers in the past decade has resulted in numerous opportunities for a large number of researchers to enter the world of parallel computing, and to do so in an integrated environment. Users of COW's and supercomputers can now talk in a common language (e.g. PVM, MPI, HPF, Vampir) and indeed use a wide variety of systems to suit their needs.

The numerous activities described in this paper provide clear evidence that COW environments can serve as training and development platforms in parallel computing. In most cases, development and initial testing and debugging of code could take place on local COW's where interactivity is often better than that from a remote supercomputing center that emphasizes maximum utilization of expensive resources. We also suggest that even a major supercomputing center can benefit immensely by offloading this training and development work to the COW's, leaving the expensive supercomputer for the long, high-PE production runs.

Though recent reports have surfaced of remarkable performances by Beowulf systems, we maintain that these tend to be specialized systems, working on problems well-suited for them. In the general computing community, we suspect that a small Linux cluster such as ours is the norm for small institutions. Such systems are generally affordable and can be adequately maintained by someone knowledgeable in Unix system administration issues. Though the performance of these systems will never compare to that of the supercomputers, they are inherently valuable to the supercomputing world by providing a low-cost environment for training and development activities which would often decrease the CPU utilization on the supercomputers.

As an educational tool, COW's are the perfect platform for introducing students to parallel computing. This is the platform where they can work interactively, make many mistakes, and ultimately become experienced enough to start using a supercomputer efficiently. We further suggest that increased collaboration between COW institutions and supercomputing institutions will

allow for more training of HPC users and increased demand for supercomputer resources.

7 Acknowledgements

The authors gratefully acknowledge the support of SGI/CRI and the Arctic Region Supercomputing Center for their long-term support of work described in this paper and, recently, for the provision of student training accounts on their T3E. We thank the National Science Foundation for provision of the "start-up" funding for the Linux cluster and logistical support of collaboration with ARSC. Finally, we thank Pallas for their generosity in allowing us to evaluate their *Vampir* performance analysis tools on our cluster over the Spring 1999 semester.

References

Foster, Ian. *Designing and Building Parallel Programs*. Addison-Wesley, 1995.

Morton, D.J, K. Wang, D.O. Ogbe. "Lessons Learned in Porting Fortran/PVM Code to the Cray T3D*," IEEE Parallel & Distributed Technology*. Vol. 3, No. 1, pp. 4-11, 1995.

Morton, D.J. "Development of Parallel Adaptive Finite Element Implementations for Modeling Two-Phase Oil/Water Flow," in *High Performance Computing Systems and Applications*, Kluwer Academic Publishers, 1998.

Morton, D.J. "The Use of Linux and PVM to Introduce Parallel Computing in Small Colleges and Universities," *Journal of Computing in Small Colleges*. Vol. 11, No. 7, 1996, pp. 13-19.

Morton, D.J., Z. Zhang, L.D. Hinzman, S. O'Connor. "The Parallelization of a Physically Based, Spatially Distributed Hydrologic Code for Arctic Regions," in *Proceedings of the 1998 ACM Symposium on Applied Computing*, Atlanta, GA, 27 Feb – 1 March 1998.

Morton, D.J., L.D. Hinzman, E.K. Lilly, Z. Zhang, D. Goering. "Coupling of Thermal and Hydrologic Models for Arctic Regions on Parallel Processing Architectures," in *Proceedings of the 3rd International Conference on Geocomputation*, Bristol, UK, 17-19 September 1998.