Software and Hardware Support for Workstation-based Supercomputing

Salim Hariri, Manish Parashar and JongBaek Park
Electrical and Computer Engineering
Syracuse University
Syracuse, NY 13244

Abstract

The proliferation of high performance workstations and the emergence of high speed networks have attracted a lot of interest in workstation-based supercomputing. We project that workstation-based environments with supercomputing capabilities will be available in the not-so-distant future. However a number of hardware and software issues have to be resolved before the full potential of these workstation-based supercomputing environments can be exploited. The presented research has two main objectives: (1) to investigate the limitations of communications techniques used in current workstation-based systems and to identify a set of requirements that must be satisfied to achieve workstation-based supercomputing; (2) to use these requirements to develop software and hardware support that enables workstation-based supercomputing. The performance of two applications, the **LU** *factorization of dense matrices and the calculation of FFT, on two platforms, the iPSC/860 supercomputer and on a cluster of general-purpose workstations, is used in the analysis to identify the limitations of current workstation clusters and to show that if these limitations are overcome, the clusters can provide comparable performance to that offered by an expensive supercomputer.*

1 Introduction

The proliferation of high performance workstations and the emergence of high speed networks (Gigabit networks) have attracted a lot of interest in workstation-based supercomputing. We project that workstation-based environments with supercomputing capabilities will be available in the not-so-distant future. The driving forces towards this end will be (1) the advances in processing technology, (2) the availability of high speed networks and (3) existence of software support and programming environments for workstation-based computing systems. Current workstations are capable of delivering tens and hundreds of Megaflops of computing power. A cluster of 1024 DEC alpha workstations would provide a combined computing power of 150 Gigaflops while the same size configuration (1024 nodes + control computer + i/o computers) of the CM5 from Thinking Machines Inc. has a peak rating of only 128 Gigaflops [1]. A recent report from the IBM European Center for Scientific and Engineering Computing [2] stated that a cluster of 8 RX/6000 Model 560 workstations connected with IBM serial optical channel converter, achieved a performance of 0.52 Gigaflops for the Dongarra benchmarks for massively parallel systems which consists of solutions to dense linear systems. The obtained results outperform a number of existing supercomputers like a 24 node Intel iPSC/860, 16 node Intel Delta, 256 node nCube2 or a 24 node Alliant CAMPUS/800. Hence, the aggregate computing power of a group of general purpose workstations is comparable if not greater than that of supercomputers. Further, workstations are general-purpose, flexible and much more cost-effective. The cost-performance ratio for a workstation today is about 5000 peak flops/$ while that for a conventional supercomputer like a Cray is only 500 to 1000 peak flops/$. Furthermore, it has been shown that the average utilization of a cluster of workstations is only around 10% [3]; consequently, around 90% of their computing capacity is sitting idle. This un-utilized or wasted fraction of the computing power is sizable and if harnessed can provide a cost-effective alternative to expensive supercomputing platforms.

Advances in computer networking technology have introduced high speed, reliable networks capable of providing high transfer rates. Current trend in local area networks is toward higher communica-

tion bandwidth as we progress from Ethernet networks that operate at 10 Mbit/sec to higher speed networks such as Fiber Distributed Data Interface (FDDI) networks. Furthermore, it is expected that soon these networks will operate in Gigabit/sec range. However, the application-level transfer rates on existing local area networks remain much lower and it is doubtful that they can keep pace with medium speed. For example, out of the 10 Mbit/sec available at the medium of an Ethernet network, only around 1.2 Mbit/sec bandwidth is available for applications [14]; it is therefore not enough to have a Gigabit data link if user applications could only use a small portion of that bandwidth.

Consequently, it has been established that current clusters of workstations have the aggregate computing power to provide an environment for supercomputing, while high speed networks, capable of supporting these computing rates, are becoming a standard (e.g., ATM, SONET, HUB-based LAN) [10]. However a number of hardware and software issues have to be resolved before the full potential of workstation-based supercomputing environments can be exploited. The performance of future Gigabit applications[11] (distributed computing, full motion video, video-on-demand, computer imaging, etc.) will be severely degraded by this low application-level transfer rate. Hardware issues that must be resolved involve the design of dedicated communication processors capable of delivering the medium bandwidths at the application level and relieving the host processor of communication overheads. The possibility for parallel interprocessor communication must also be supported. Software issues involve introducing new high-speed transport protocols. The general design concepts for high-speed protocols can be characterized as (1) design philosophies, (2) architecture and (3) implementation philosophies [11]. Although these projects have resulted in reducing communication overhead and improving the application transfer rates, these rates are still too slow to meet the interprocess communication bandwidth needed to solve Gigabit applications across a high-speed network; Most of these techniques support a local area network access protocol that allows only one computer to transmit at a time (sequential interprocess communication).

The presented research has two main objectives: (1) to investigate the limitations of communications techniques used in current workstation-based systems and to identify a set of requirements that must be satisfied to achieve workstation-based supercomputing and (2) to use these requirements to develop software and hardware support that meets the requirements. In this paper, we first study the performance of two applications, the **LU** factorization of dense matrices and the calculation of FFT, on the iPSC/860 supercomputer and on a conventional cluster of general-purpose workstations. We use these results to identify the limitations of current workstation clusters and show that if these limitations are overcome, the clusters can provide comparable performance to that offered by an expensive supercomputer. Using these results, we outline a set a requirements which must be satisfied for workstation-based supercomputing. We then propose a software and hardware support environment which meets all the above requirements and demonstrate how it can be used for high performance supercomputing.

The organization of the paper is as follows: Section 2 describes the experimental environment used in our study. It outlines the specifications of the hardware platforms and the software libraries used. Section 3 briefly describes the applications used in the study and outlines the algorithms. Section 4 analyses the numerical results obtained by running the same applications on the iPSC/860 hypercube and on a cluster of workstations, highlighting the limitations of current workstation-based systems. It then identifies a set of requirements that must be satisfied to obtain performance comparable to supercomputers. Section 5 describes a hardware and software support environment which meets the requirements. Section 6 provides some concluding remarks.

2 Experimentation Environment

PVM (Parallel Virtual Machine) [4] was used to distribute the application over a network of workstations. PVM provides a programming environment for the development and execution of parallel applications across a collection of possibly heterogeneous computing elements interconnected by one or more networks. For our experimentation, we used a set of SUN SPARCstation 1 workstations interconnected by an ethernet.

The Intel iPSC/860, an Intel i860 processor based hypercube, was used to compare its performance with that obtained on a cluster of SUN workstations. Each node of the iPSC/860 hypercube operates at a clock speed of 40 MHz and has a theoretical peak performance of 80 MFLOPS for single precision and 40 MFLOPS for double precision. Communication is supported by direct-connect modules

present at each node which allow the nodes to be treated as though they are directly connected. The communication time for a message is a linear function of the size of the message. Hence, the time, t_m to transmit a message of length n bytes from an arbitrary source node to an arbitrary destination node is given by:

$$t_m = t_s + t_b \times n$$

where t_s is the fixed startup overhead and t_b is the transmission time per byte. PICL [5] (Portable Instrumented Communication Library) was used for communication on the iPSC/860 and provided a simple and portable communication structure. In addition, the implementation of **LU** factorization on the iPSC/860 made use of BLACS [6] (Basic Linear Algebra Communication Subprograms) library for data movement as well as routines from Level 3 BLAS (Basic Linear Algebra Subprograms) which are a part of the LAPACK [7] linear algebra library.

3 Applications

In this section we breifly describe the two applications which will be used to study the limitations and requirements of current workstations-based computing environments. The application chosen are the **LU** factorization of dense matrices and the computation of Fast Fourier Transforms (FFT). These application form an integral part of many scientific and engineering problems (e.g. **LU** factorization is required for solving systems of linear equations while FFT's are used extensively in digital signal processing).

3.1 LU Decomposition

running example in the presented study. Solutions to a system of linear equations are required in many scientific applications. Consider the solution of the dense system of linear equations,

$$\mathbf{A}\vec{x} = \vec{b}, \tag{1}$$

where $\mathbf{A}$ is an n-by-n matrix and $\vec{b}$ and $\vec{x}$ are vectors of dimension n. One method of solving this system is to proceed by first factorizing $\mathbf{A}$ into a unit lower triangular matrix $\mathbf{L}$ and an upper triangular matrix $\mathbf{U}$, *i.e.*,

$$\mathbf{A} = \mathbf{LU}, \tag{2}$$

and then solving for $\vec{y}$ and $\vec{x}$ in two consecutive substitution steps:

$$\mathbf{L}\vec{y} = \vec{b} \quad \& \quad \mathbf{U}\vec{x} = \vec{y}. \tag{3}$$

Experimental results show that in programs which need to solve linear systems, more than 50% of the CPU time is usually spent in matrix factorization since the computational complexity of the factorization step is $O(n^3)$ while that of the substitution steps is $O(n^2)$. Hence an efficient factorization algorithms will have a significant impact on the performance of a linear system solver. Blocked algorithms increase the computations per memory access by considering the matrix as a collection of sub-matrices where each sub-matrix could be a group of columns (column blocking) or rows (row blocking). The use of block-based algorithms for matrix computations is one of the most efficient ways to improve the performance on machines with a hierarchical memory structure. In our implementation of LU factorization, we use the column blocking strategy which complements column oriented Fortran language.

There are six ways of implementing the **LU** factorization obtained by ordering the three nested loops that constitute the algorithm. Since Fortran is column-oriented, only three of the six forms namely; jik-SDOT (also known as Crout's algorithm), jki-GAXPY, and kji-SAXPY, are suitable for Fortran applications. Our previous work has shown that the kji-SAXPY algorithm is most suitable for distributed memory MIMD systems [8] and is used for the current application.

Blocked kji-SAXPY is an iterative algorithm wherein, in the j^{th} step, one block column of $\mathbf{L}$ and one block row of $\mathbf{U}$ are computed and the corresponding transformations are applied to the remaining sub-matrix. The structure of the algorithm requires that the blocks of the matrix are factorized in a sequential order. That is, panel j must be factorized and shipped before panel $j+1$ can be factorized. In order to offset this inherent bottleneck, a pipelined approach is used. In this approach, the iterations of the algorithm are pipelined so as to overlap the factorization of the j^{th} block column with the updates associated with the $j-1^{th}$ factorized block column. Details of this algorithm as well as its performance on the iPSC/860 can be found in [8, 9].

3.2 Fast Fourier Transform (FFT)

Given the input sampled signal $s(k)$, $k = 0, 1, ..., M-1$, computation of FFT gives the output $X(i)$, $i = 0, 1, \ldots, M-1$ such that $X(i) = \sum_{k=0}^{M-1} s(k) \cdot W^{ik}$, where $W = \exp(j\frac{2\pi}{M})$, $j = \sqrt{-1}$ and M = number of sample points.

Assuming we have N workstations on the net-

Figure 1: DIF FFT

using host-node programming paradigm of PVM. Host algorithm is mainly divided into 2 steps: in the first step, the input data is partitioned into N segments and distributed over node processors, and the results computed at each node are collected in the second step.

We show the node algorithm below where W is a complex coefficient involved in FFT computation ($e^{j2\pi/M}$), my_num represents identifier of each node process, and A and B represent the input data distributed to the node.

```
receive((A,B), host);    /* from host */

For step=0 to log2N − 1
      d = N · 2^(−step−1);    /* communication distance */
      k= my_num · 2^step mod N;
      X = A + B; /* computation */
      Y = (A − B) · W^k;

      if(k < N/2) /* communication */
            node = my_num + d;
            send(Y, node);
            receive(B, node);
      A = X;
      else
            node = my_num − d;
            send(X, node);
            receive(A, node);
            B = Y;

/* final step of computation */
X= A + B;
Y= A − B;
send((X,Y), host);  /* send results to host */
```

In measuring the performance of this algorithm, we assumed that multiple sets of the sampled data is provided to vary message size of interprocess communications.

4 Results

The performance of the two application examples is presented in this section. The study is divided into three phases. The first phase analyzes the performance obtained by distributing the applications over a network of workstations. Phase 2 studies the distribution of the same application on the iPSC/860 hypercube. In phase 3, we use the results of phases 1 and 2 to extrapolate the effect of increasing the available network bandwidth on the obtained performance.

4.1 Phase 1 - Algorithm Performance on a Network of Workstations

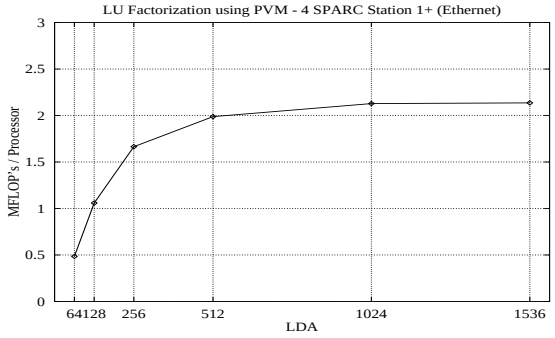

Figure 2: Peak MFLOP's/Processor obtained. Varying matrix sizes.

In the first phase of the study, we use four SUN SPARCstation 1+ workstations interconnected by

an ethernet. For the case of **LU** decomposition, Figure 2 plots the peak performance per processor for each problem size. This plot displays the scalability of performance with problem size. It can be seen that the obtained performance on a network of workstations saturates for larger matrix sizes. This saturation results from performance limitations of the processors. The variation of the communication time to computation time ratio with problem size is shown in Figure 3.

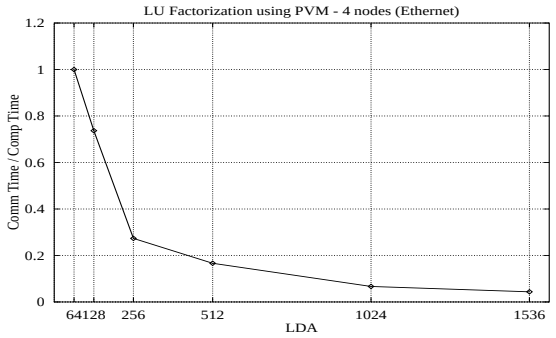

Figure 3: Variation of Comm. Time/Comp. Time with problem size.

The figures support the intuitive result that as the communication to computation ratio decreases, the performance increases. Acceptable performances ($\geq$ 1.5 MFLOPS) are obtained only when the ratio drops below 0.4; i.e. for matrix of sizes 200×200 and greater. Performance for smaller problem sizes is limited by the communication time i.e. by the network bandwidth. The performance of

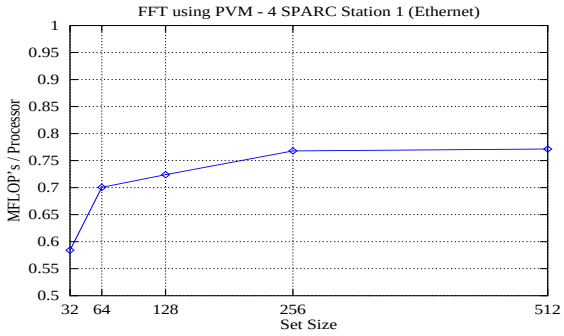

Figure 4: MFLOP's/Processor obtained. Varying set size and M = 32

FFT computation is also measured in MFLOP's by varying the set size (S) and the number of sampling points, M, is 12 as shown in Figure 4. The ratio of communication time to computation time is plotted in Figure 5. It is clear from the figure that the peak MFLOP's increases as the set size increases; however, as the message size increases, the performance saturates due to the same reason discussed above.

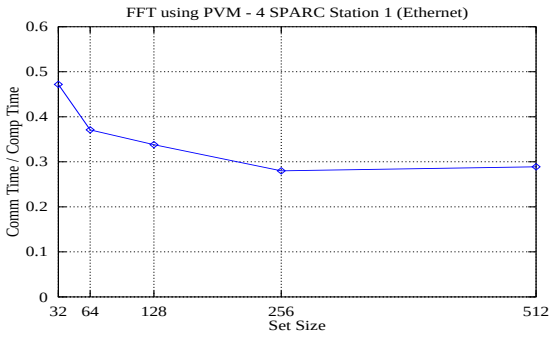

Figure 5: Variation of Comm. Time/Comp. Time. Varying set size and M = 32

4.2 Phase 2 - Algorithm Performance on the iPSC/860 Hypercube

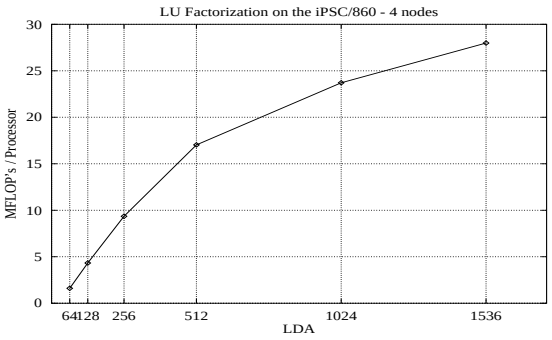

Figure 6: Peak MFLOP's/Processor obtained. Varying matrix sizes.

The second phase of the study consists of distributing the same algorithms on a parallel machine. The corresponding results were obtained using 4 nodes of a 32 node iPSC/860 hypercube. Figure 6 shows the scalability of performance with problem size in case of the iPSC/860. In Figure 8, we plotted the performance of FFT on the iPSC/860 for the case of $M = 32$. Figures 7 and 9 show the ratio of communication time to computation time on iPSC/860 for **LU** decomposition and FFT, respectively.

4.3 Phase 3 - Extrapolation

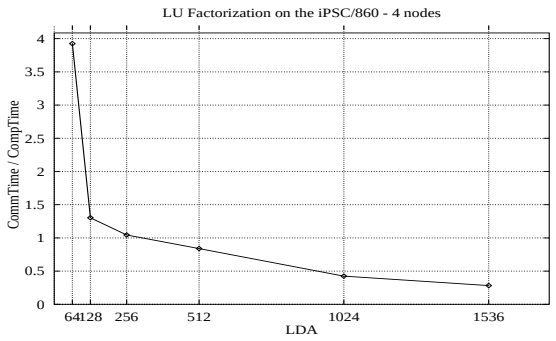

Figure 7: Variation of Comm. Time/Comp. Time with problem size.

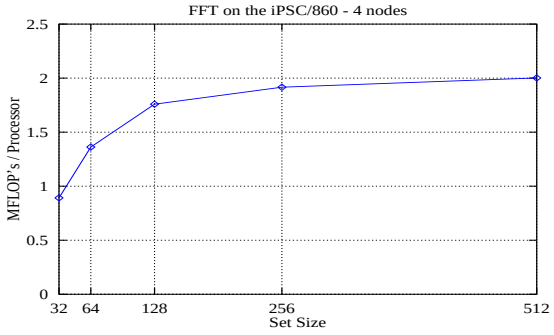

Figure 8: MFLOP's/Processor obtained. Varying set size and M = 32

In phase 3 of our study, we used the results obtained in phases 1 and 2 to extrapolate the effect of scaling up the data transfer rates available to the application. This is achieved by increasing the available bandwidth so as to effectively reduce the communication time by factors of 10, 100 and 1000 and could be thought of as moving from ethernet

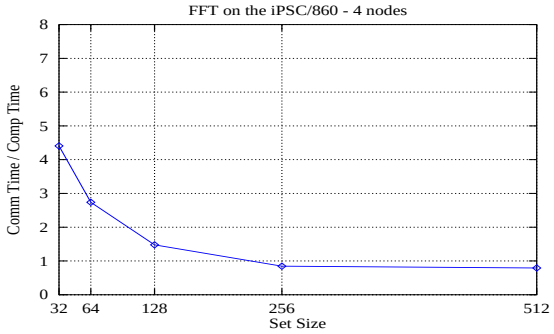

Figure 9: Variation of Comm. Time/Comp. Time. Varying set size and M = 32

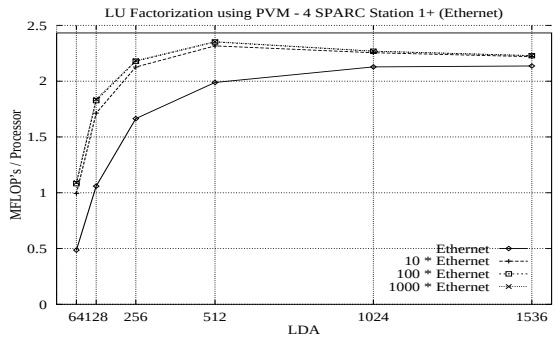

Figure 10: Extrapolated MFLOP's/Processor for effective data rates scaled up by factors of 10, 100 & 1000.

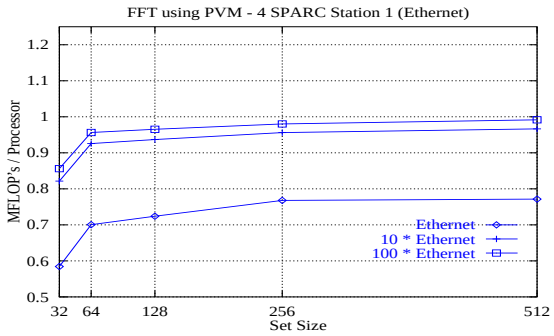

Figure 11: Extrapolated MFLOP's/Processor for effective data rates scaled up by factors of 10 & 100.

(10 Mbits/sec) to FDDI (100 Mbits/sec) to Gigabit networks respectively. The obtained results are shown in Figure 10 for **LU** case and in Figure 11 for FFT case. These figures show the effect of scaling up the effective available data transfer rates. It can be seen that the obtained performance still saturates for large problem sizes due to the limited capacity of the processing nodes. As a result, there is a very small increase in performance as the network speed moves from 100 Mbits/sec to 1 Gigabit/sec. The effect of such a scaling of available transfer rates would be stronger and more prevasive for applications which require intense interprocessor communication. For smaller problem sizes, however, there is a steeper increase in performance. Hence, increasing the data transfer rates overcomes the overhead of large communication to computation ratios encountered for smaller problem sizes. In Figure 12 and 13, we scale the performance of each node to match that of an i860 node (for an effective data transfer rate scaled by

Figure 14: Structure of HLAN

HLAN adopts two types of protocol stacks as shown in Figure 15. We present briefly the architecture of a High-speed Communication Protocol (HCP) and

Figure 15: A configuration of protocol stack with HCP

5.1 High-speed Communication Protocol (HCP)

As network speed increases to Gigabit/sec range, communication latency between computers is becoming comparable to that between internal components of a computer. We envision that Gigabit LANs will allow its computers to interact and collaborate with latency comparable to that now exist between the internal components of a computer.

The key design concepts of HCP are as follows:

- **Simplicity:** One of the main design goals of HCP is to provide low latency data transfer capability between computers in networks operating in Gigabit range. In order to achieve this goal, the communication protocol should be simple and can be processed in an order of magnitude less than that is currently achievable with existing standard protocols. Current protocols were designed to be robust in the face of adverse network conditions and this makes protocols too complex to operate at Gigabit data rate.

- **Concurrency:** Another key design concept in HCP lies in providing concurrency. Existing LAN protocols allow only one station to transmit data at a time (sequential interprocess communication); for example, in an FDDI network, concurrency is not provided since only one station that holds the token can transmit data. HCP allows multiple computers to transmit at the same time (parallel interprocess communication); for example, in HLAN with N computers in the network, we could have all N computers transferring data simultaneously to their immediate neighbors, and thus maximizing network throughput.

- **Scalability:** HLAN is scalable in a similar manner to that used in the bus system; HLAN bandwidth can be scaled up by increasing the number of parallel channels in the D-net and corresponds to increasing the number of parallel data lines in the bus system.

HCP can cope and handle networks operating in Gigabit or even Terabit range; in existing standard protocols, at high transmission rates, the network interface processor should process an incoming packet within a very short time interval. For example, let us assume that 1000 instructions are needed for processing 1 Kbyte packets [13]. For a network operating in 100 Mbit/sec or 1 Gigabit/sec, the interface should process the incoming packet in 80 μsec, and 8 μsec, respectively. Consequently, the network interface processor speed must be 12.5 MIPS and 125 MIPS, respectively. It is clear from this simple analysis that the existing protocols can not be scaled up to Gigabit or Terabit rates and new protocols such as HCP must be developed.

5.2 Host-Interface Processor (HIP)

In this subsection, we briefly describe the main features of the Host Interface Processor(HIP). HIP is a communication processor capable of operating in two modes of operation such that one or both of these modes can be active at a given time. In HSM, HIP provides applications with data rate comparable to medium rate. This high speed transfer rate is achieved by (1) using simple communication protocol, HCP, (2) decomposing the transmit/receive tasks into several subtasks that can run concurrently on a separate engine and (3) using dedicated channels that allow all nodes to transmit and receive data concurrently. In NSM, the standard transport protocols can run efficiently on HIP and thus offload the host from running these protocols. Figure 16 shows a block diagram of the main functional units of the proposed HIP. The HIP design consists of five major subsystems: a Master Processing Unit (MPU), a Transfer Engine Unit (TEU), a crossbar switch, and two Receive/ Transmit units (RTU-1, RTU-2). The architecture of HIP is highly parallel and uses hardware multiplicity and pipeline techniques to achieve high-performance transfer rates.

Figure 16: Blockdiagram of HIP

For example, the two RTUs can be configured to transmit and/or receive data over high-speed channels while the TEU is transferring data to/from the host. In what follows, we describe the main tasks to be carried out by each subsystem.

5.2.1 Master Processing Unit (MPU)

HIP is a master/slave multiprocessor system where MPU controls and manages all the activities of HIP subsystems. The Common Memory(CM) is a dual-port shared memory and can be accessed by the host through its standard bus. Furthermore, this memory is used to store control program that runs on MPU. The MPU runs the kernel software that provides an environment in which two modes of operation can be supported (HSM and NSM), and several parallel activities (receive/transmit from/to the host, receive and/or transmit over the D-net, and receive/transmit over the normal network). The main tasks of the kernel system are outlined as follows.

- **HIP manager** : This involves configuring the subsystems to operate in certain configuration, allocating and deallocating processes to HIP processors (TEU, RTUs),and arranging the execution order of HIP processes on RTUs and TEU such that their asynchronous parallel executions will not result in erroneous results and any deadlock scenarios.

- **HIP-based LAN (HLAN) manager** : This involves setting up a cluster of computers to cooperate in a distributed manner to execute a compute-intensive application using the D-net dedicated to the HSM of operation. The computer that owns this application will use the D-net to distribute the decomposed subtasks over the involved computers, synchronize their computations, and collect the results from these computers.

- **HLAN general management** : This involves collecting information about the network activities to guarantee certain performance and reliability requirements. These tasks are related to network configuration, performance management, fault detection and recovery, accounting, security and load balancing.

5.2.2 Transfer Engine Unit (TEU)

The host is separated from controlling all aspects of the communication process which is handled entirely by the TEU. The initiation of data transfer is done by the host through the Common Memory (CM) and the completion of transfer is notified through an interrupt to the host. The TEU can be implemented simply as a Direct Memory Access Controller (DMAC).

5.2.3 Switches

This is a 2x2 nonblocking crossbar switch that provides maximum connections among the TEU, MPU, and RTUs. The use of local buses in MPU and RTUs allow any component of these subsystems to be accessed directly through the switch.

5.2.4 Receive/Transmit Unit (RTU)

The main task of the RTU is to off-load the host from getting involved in the process of transmitting/receiving data over the two channels. At any given time, the RTU can be involved in several asynchronous parallel activities: receive and/or transmit data over the point-to-point channel according to HCP protocol.

6 Conclusion

In this paper, we analyzed the current advances in computing technology, network technology, and

software tools for developing parallel and distributed computing applications. We also demonstrated the potential of workstation-based computing through two experiments of compute-intensive applications; a cluster of high performance workstations can be used to build a cost-effective supercomputing environment.

We proposed a High-speed LAN architecture to efficiently implement the network-based supercomputing environment. In this environment, HCP and HIP provide the software and hardware support needed for high-performance network-based computing.

We are currently planning to build a distributed system to interconnect a cluster of high performance workstations available at NPAC (Northeast Parallel Architecture Center) like SUN SPARCstation's, IBM RISC System/6000, DECStation's etc. This network will provide the testbed environment to experiment and validate the performance of the communication protocol HCP when it is implemented on a special host-interface processor such as HIP.

References

[1] Gordon Bell, "Ultracomputers A Teraflop Before Its Time", *Communications of the ACM*, vol. 35, pp. 27–47, Aug. 1992.

[2] IBM European Center for Scientific and Engineering Computing, ", Usenet news item, 1992.

[3] P. Kruegeer and R. Chawal, "The Stealth Distributed Schedular", *in Proceedings of the 11th International Conference on Distributed Computing Systems, Texas*, pp. 336–343, May 1991.

[4] V. S. Sunderam, "PVM: A Framework for Parallel Distributed Computing", Technical report, Oak Ridge National Laboratory, Oak Ridge, Tennessee 37831, 1991.

[5] G. A. Geist, M. T. Heath, B. W. Peyton, and P. H. Worley, "A User's Guide to PICL, A Portable Intrumented Communication Library", Technical Report Tech. Rep. ORNL/TM-11616, Oak Ridge National Laboratory, Oak Ridge, Tennessee 37831, Oct 1991.

[6] E. Anderson, A. Benzoni, J. Dongarra, S. Moulton, S. Ostrouchov, B. Tourancheau, , and R. van de Geijn, "Basic Linear Algebra Communication Subprograms", *Sixth Distributed Memory Computing Conference Proceedings, IEEE Computer Society Press*, vol. , pp. pp. 287–290, 1991.

[7] E. Anderson, A. Benzoni, J. Dongarra, S. Moulton, S. Ostrouchov, B. Tourancheau, and R. van de Geijn, "LAPACK for Distributed Memory Architectures: Progress Report", *To appear in the Proceedings of the Fifth SIAM Conference on Parallel Processing for Scientific Computing, Houston*, vol. , 1991.

[8] G. Von Laszewski, M. Parashar, A. G. Mohamed, Geoffrey C. Fox, "On the Parallelization of Blocked LU Factorization Algorithms for Distributed Memory Architectures", To be presented at Supercomputing 1992, Nov. 1992.

[9] M. Parashar, S. Hariri, A. G. Moahamed, and G. C. Fox, "A Requirement Analysis for High Performance Distributed Computing over LAN's", *in Proceedings, First International Symposium on High Performance Distributed Computing*, pp. 142–151, Sep. 1992.

[10] H. T. Kung, " Gigabit Local Area Networks: A Systems Perspective," *IEEE Communications Magazine,* pp. 79-89, April 1992.

[11] T. F. La Porta and M. Schwartz," Architectures, Features, and Implementation of High-Speed Transport Protocols," *IEEE Network Magazine*, pp. 14-22, May 1991.

[12] K. Hwang and F. A. Briggs, *Computer Architecture and Parallel Architecture*, McGraw-Hill 1984.

[13] C. Partridge," How Slow Is One Gigabit Per Second ? " *Symposium on Applied Computing*, 1990.

[14] G. Chesson, " The Protocol Engine Project," *Proceedings of the Summer 1987 USENIX Conference,* pp. 209-215, June 1987.