A Requirement Analysis for High Performance Distributed Computing over LAN's

Manish Parashar and Salim Hariri
Electrical and Computer Engineering
Syracuse University
121 Link Hall, Syracuse, NY 13244

A. Gaber Mohamed and Geoffrey C. Fox
Northeast Parallel Architectures Center
Syracuse University
111 College Place Syracuse, NY 13244

Abstract

With the proliferation of high performance workstations and the current trend towards high speed communication networks, the cumulative computing power provided by a group of general purpose workstations is comparable to supercomputers. However a number of obstacles have to be overcome before the full potential of these network-based distributed systems can be exploited.

This paper investigates the requirements of current workstation clusters interconnected by local area networks (LAN's) which would allow them to be used as platforms for high performance distributed computing. The blocked **LU** *decomposition of dense matrices is used as the the running example in the presented study. Performance of this algorithm is measured on the iPSC/860 hypercube and on a set of homogeneous workstations (SUN SPARCstation 1+) interconnected by ethernet. These measures are analyzed and a set of requirements are identified which would enable a network of workstations to deliver high performance distributed computing.*

1 Introduction

With the proliferation of high performance workstations and the current trends towards high speed computer networks, network based distributed computing has attracted a lot of interest. The aggregate computing power of a group of general purpose workstations is comparable to that of supercomputers. Furthermore, it has been shown that the average utilization of a cluster of workstations is only around 10% [1]; 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. However, a number of obstacles have to be overcome before the full potential of network-based distributed systems can be exploited. The primary bottleneck continues to be the limited communication bandwidth available at the application level. In current LAN's, only about 10% of the physical bandwidth is available at the application level [2]. For example, of the physical bandwidth of 10 Mbits/sec available at the medium level of the ethernet, only around 1.2 Mbits/sec is available to the application. The reasons for this inefficiency are: (1) the excessive overhead of the host-to-network interface characterized by heavy usage of timers, interrupts and memory read/writes and (2) the overheads incurred by standard transport protocols which are implemented as a stack of software layers and consume most of the medium capacity [3]. As a result, even though physical bandwidths are moving to the Gigabit range, the bandwidth available to the application continues to be a bottleneck.

The objective of the presented research is three fold: (1) to develop distributed algorithms for solving computationally intensive applications on different parallel computer architectures and on a network of high performance workstations and to make them available as benchmarking tools for Fortran-D and High Performance Fortran (HPF) [4, 5], (2) to compare their performance (measured in MFLOP's) in-order to highlight the limitations of current network-based distributed systems which prevent them from delivering high performance distributed computing, and (3) to identify the requirements that must be satisfied before the full computing potential of network-based distributed systems can be utilized, allowing them to deliver high performance distributed computing.

The application chosen for the study presented in this paper is the **LU** decomposition of dense matrices which forms the computational core of many linear algebra applications. The blocked, pipelined, *kji*-SAXPY variation of the **LU** decomposition algorithm

0-8186-2970-3/92 $03.00 © 1992 IEEE

is selected due to its suitability to distributed memory MIMD systems [6]. Experimentations with the $kji-SAXPY$ and other variations on shared memory MIMD and distributed memory MIMD architectures can be found in [7] & [6] respectively. The computing platforms used in our study are a cluster of SPARCstation 1+ workstations from SUN Microsystems and Intel's i860 based iPSC/860 hypercube, available at the Northeast Parallel Architectures Center at Syracuse University.

The rest of the paper is organized 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 **LU** factorization problem and specifically, the blocked, pipelined, kji-SAXPY algorithm. Section 4 discusses the results obtained. Section 5 highlights the limitations of current network-based distributed systems and identifies a set of requirements that must be satisfied to obtain performance comparable to supercomputers. Section 6 presents our conclusions and is followed by a list of references.

2 Experimental Environment

PVM (Parallel Virtual Machine) [8, 9] 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 [10] 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 [11] (Portable Instrumented Communication Library) was used for communication on the iPSC/860 and provided a simple and portable communication structure. In addition, the implementation on the iPSC/860 made use of the BLACS [12] (Basic Linear Algebra Communication Subprograms) library for data movement. BLACS is a portable, high level communication library, developed for linear algebra applications.

The matrices used in the experiments were dense matrices where each matrix element was a randomly generated real number between 0 and 1.

LAPACK [13, 14, 15] is a portable, public, linear algebra library based on the use of parallelized BLAS (Basic Linear Algebra Subprograms) kernels and is provided by vendors on different machines. BLAS kernels provide efficient library routines for a large number of vector-vector(Level 1), matrix-vector (Level 2) and matrix-matrix (Level 3) computations. Level 1 BLAS provides computations of order $O(n)$, Level 2 BLAS provides computations of order $O(n^2)$ and Level 3 BLAS provides computations of the order $O(n^3)$. Level 3 BLAS routines provide a high ratio of operations to data movement and are suitable for computers which have a hierarchical memory structure. The blocked implementation of kji-SAXPY makes use of the Level 3 BLAS routines. Specifically, the following LAPACK and Level 3 BLAS routines are called:

- **SGETF2:** The non-block LU factorization routine called to factorize a block column.
- **STRSM:** The triangular solve with multiple right hand sides (result is a matrix).
- **SGEMM:** The matrix-matrix multiply routine.

3 LU Decomposition

The blocked **LU** decomposition of dense matrices is used as a running example in the presented study. Solutions to a system of linear equations are required in many scientific applications. [16, 17, 18, 19]. 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 [20]. 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 [6] and will be used in the presented study. The blocked *kji*-SAXPY algorithm is described below.

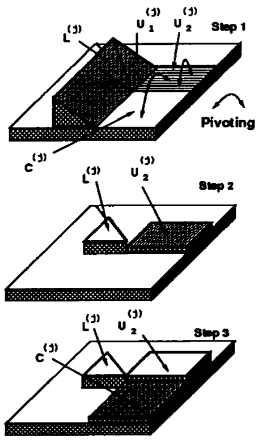
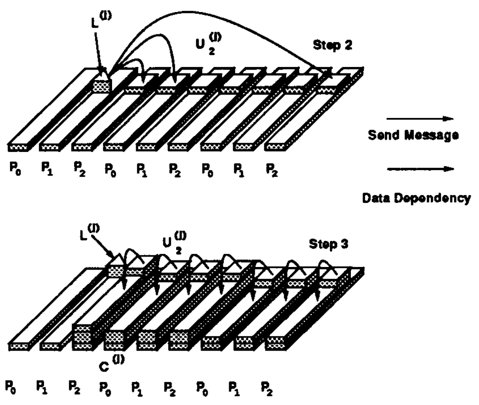


Figure 1: *kji*-SAXPY



$L^{(j)}$
$U_2^{(j)}$
Step 2
Send Message
Data Dependency
P_0 P_1 P_2 P_0 P_1 P_2 P_0 P_1 P_2
$L^{(j)}$
$U_2^{(j)}$
Step 3
$C^{(j)}$
P_0 P_1 P_2 P_0 P_1 P_2 P_0 P_1 P_2

Figure 2: Parallel SAXPY

3.1 Parallel Blocked *kji*-SAXPY

Blocked *kji*-SAXPY is an iterative algorithm wherein, in the j^{th} step, one block column of **L** and one block row of **U** are computed and the corresponding transformations are applied to the remaining submatrix. The basic steps involved in the j^{th} iteration are shown in Figure 1. The data dependencies at each step can be visualized from the height of the matrix elements in the figure. If a datum is higher than another, then this datum needed in the current iteration and must be calculated first. Thus, in the j^{th} iteration, the j^{th} block depends on $j-1^{th}$ factorized block. The operations performed in each iteration of the algorithm are described below:

0. Initialize Start with the first block; $j \leftarrow 1$

1. Factorize $C^{(j)}$ The j^{th} block column is factorized into **LU** factors, $L^{(j)}$ & $U_1^{(j)}$, using the *jik*-noblock algorithm (SGETF2). The row interchanges (pivoting) are applied to blocks on both sides of the current block.

2. Update $U_2^{(j)}$ The j^{th} block row of **U** is updated using STRSM: $U_2^{(j)} \leftarrow (L^{(j)})^{-1} U_2^{(j)}$

3. Update $C^{(j)}$ The remaining matrix is updated using a block outer product (SGEMM): $C^{(j)} \leftarrow C^{(j)} - L^{(j)} U_2^{(j)}$

4. Iterate: IF no more blocks remaining THEN stop ELSE goto Step 1.

Figure 2 shows the distribution of the matrix onto the processing nodes. The matrix is mapped using a panel-wrapped storage wherein it is divided into fixed size panels (block columns) which are wrapped onto the p nodes so that panels $i + kp$, $k = 0, 1, ...; i = 1, 2, ...$, are assigned to the node with index $i - 1$. 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 [15]. In this approach, the iterations of the algorithm are pipelined so as to overlap the factorization of the j^{th} block column (step 1) with the updates associated with the $j - 1^{th}$ factorized block column (steps 2 and 3). The pseudo-code for the pipelined version of the algorithm is given in Figure 3. Here, n = number of columns of the matrix, m = number of rows of the matrix, lda = leading dimension of the matrix, $Processors$ = number of allocated processors, $proc$ = number of the processor currently performing the factorization and shipping of the panel, and $myid$ = id of a processor. A is the matrix of dimension $lda \times n$ to be factorized. Figure 2 shows the operation of the parallel blocked algorithm in the second iteration. Here processor 1 has factorized its panel and has shipped the factorized panel to the other processors. Now in step 2, each processor uses the value of $L^{(j)}$ it received from the current processor to update its portion of $U_2^{(j)}$. In step 3, the values of $L^{(j)}$ and $U_2^{(j)}$ are used to update $C^{(j)}$, that is the sub-diagonal sub-matrix.

4 Results

The performance of the parallel blocked **LU** factorization of dense matrices is presented in this section. Matrix sizes ranging from 64×64 to $1.5K \times 1.5K$ were used to allow us to study a wide range of computational loads. The study is divided into three phases. The first phase analyzes the performance obtained by distributing the application over a network of workstations. Phase 2 studies the distribution of the same application on the iPSC/860 hypercube. In phase three, 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

In the first phase of the study, we use a set of 4 SUN SPARCstation 1+ workstations interconnected by an ethernet. The performance is measured in MFLOP's for different problem sizes and for varying block sizes for each problem size. These results are plotted below. Figure 4 shows the effect of different block sizes on the obtained MFLOP's. In the blocked algorithm used, during each iteration, one block of data is transmitted by the current processor to every other processor (as described in section 3). Hence small block sizes imply smaller message sizes . However, the total number of messages sent increases as the block size decreases. It can be seen from the figure that the optimal block-size (message size) varies with the problem size and lies in the range 10 ± 4.

Figure 5 plots the peak performance 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 6 while its effect on the peak MFLOP's obtained per processor is plotted in Figure 7.

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.

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

The second phase of the study consists of distributing the same algorithm on a parallel machine. The corresponding results were obtained using 4 nodes of a 32 node iPSC/860 hypercube. Figure 8 shows the effect of block size on the performance for different problem sizes. As seen in the case of the network of workstations, the best performance is achieved for block-sizes in the range 10 ± 4. Figure 9 shows the scalability of performance with problem size in case of the iPSC/860. The variation of the computation to communication ratio with problem size and its effect on the MFLOP's per processor can be seen from Figures 10 & 11 respectively.

4.3 Phase 3 - Extrapolation

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 (10 Mbits/sec) to FDDI (100 Mbits/sec) to Gigabit networks respectively. The obtained results are shown in Figure 12. This figure shows 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. Also, the studied algorithm was pipelined to reduce the effect of the communication latency. 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 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 13, we scale the performance of each node to match that of an i860 node (for an effective data transfer rate scaled by 1000). By comparing Figures 11 & 13 we can predict that the performance of a network of high performance workstations will be comparable to that obtained for supercomputers. For example for 512×512 matrix, we get about 17 MFLOP's on the iPSC/860, while a network of high performance workstations obtain a comparable 18 MFLOP's.

5 Requirement Analysis

Having analyzed the results (presented in Section 4) obtained by running the **LU** decomposition algorithm on a parallel computer and on a network of workstations, we can now highlight the requirements of high-speed networks and communication protocols to enable them to achieve performance comparable to that offered by supercomputers. Figures 6 & 7 show that in-order to achieve acceptable performance, communication to computation ratios must be small. This is not possible for smaller problem sizes where the communication time is comparable, if not greater than the computation time. Further, in-order to exploit maximum parallelism from any application, the parallelism should be at the finest grain which is only possible if low latency communication is available for a wide range of problem sizes. Hence, to be able to use computers interconnected by LAN's efficiently for high performance distributed computing, communication latency must be reduced. In what follows we enumerate a set of features which must be incorporated into the design of networks and their transport protocols in-order to achieve this goal.

- **High User-to-User Transfer Rates**
  As pointed out earlier, in current LAN's, user-level transfer rates are only a small fraction of the available medium bandwidth. User-to-User Transfer Rates comparable to the physical bandwidth of the communication medium must be achieved before network-based systems can be effectively used as platforms for high performance distributed computing.

- **Host-to-Network Interfaces**
  Efficient Host-to-Network Interfaces are required which will allow applications to see transfer rates close to the available physical bandwidth. Current interfaces are characterized by heavy usage of timers, interrupts and memory read/writes which result in excessive overheads in terms of processor cycles and bus capacity [3]. Host-to-Networks interfaces for high performance distributed computing must be able to overcome these overheads and in addition, must be capable of handling highly bursty traffic which is characteristic of LAN's [21]. Furthermore, as network bandwidths increase, there is an increased possibility of mismatches in bandwidth between interconnected networks or between slower hosts and the network. Congestion which can occur on the networks due to these mismatches must be efficiently handled.

- **High Speed Transport Protocols**
  New High Speed Transport Protocols need to be developed to reduce current overheads. Current protocols are implemented as stacks of software layers. Each of these layers adds to the overheads and hence reduces the effective transfer rates available to the user. Every message has to be processed by each layer at the source as well as the destination nodes, thereby increasing the communication latency experienced by the application. The new protocols must be simple and must achieve maximum throughput. Host processors should be off-loaded by implementing the transport protocol on an interface processor.

- **Parallel Interprocess Communication**
  Existing LAN protocols are inherently sequential, i.e. only one node can transmit at a time (e.g. in a token ring network, only the node that has the token can transmit). High performance distributed computing warrants Parallel Interprocess Communication wherein multiple nodes can communicate simultaneously. This will result in a more efficient utilization of the network resource.

- **Support Standard and High Speed Transport Protocols**
  Another desired feature is the ability to support standard as well as high speed transport protocols. This is required to ensure compatibility with existing software and computing resources in addition to providing high speed transport protocols for distributed computing. Since we envision future computer networks to be heterogeneous in nature and to include both standard as well as non-standard high speed transport, maintaining such compatibility is essential.

We are currently working on a High Performance Interface Processor (HiP) at Syracuse University [3]. This system is designed to achieve most of the requirements listed above. HiP is a dedicated communication processor which provides architectural support needed to improve the user-level transfer rates. It supports both standard and non-standard, high speed protocols by providing two modes of operation; (1) a Normal Speed Mode (NSM) for standard protocols and (2) a High Speed Mode (HSM) where it bypasses the standard transport layers to achieve transfer rates comparable to the medium speed. HiP allows parallel interprocessor communications thereby providing support for efficient distributed computing over a cluster of workstations. A HiP based LAN (HLAN) is shown in Figure 14. In this figure, the HSM is used over the high speed channel while the NSM speed mode is used for the normal speed channel.

Figure 15 shows a block diagram of the main functional units of the HiP architecture. The major functional units if the HiP design are as follows:

1. **A Master Processing Unit (MPU)**
   The master processing unit controls and manages all the activities of the master/slave HiP multiprocessor system. The Common Memory (CM) is used for Host-HiP communication. MPU runs the HiP software layer and supports the two modes of operation (HSM & NSM). Its tasks include: (1) HiP management, (2) HLAN management and (3) synchronization

2. **Transfer Engine Unit (TEU)**
   The transfer engine unit handles all aspects of the communication process thereby relieving the host of the overhead. It is configured by the MPU and can be implemented as a simple DMA controller.

3. **Switch**
   This is a 2×2, non-blocking, crossbar switch that provides maximum connections among the MPU, TEU, RTU-1 and RTU-2 subsystems.

4. **Receive/Transmit Unit**
   RTU off-loads the host from the task of transmitting/receiving data over the two channels. At any given time each RTU can be involved in the following concurrent activities: (1) Receive and/or transmit data over the normal speed channel; (2) Receive and/or transmit data over the point-to-point channel according to the HiP transport protocol;

6 Conclusion

In this paper we presented a requirement analysis for high performance distributed computing over LAN's. We distributed the **LU** decomposition of dense matrices on the following platforms: A cluster of SUN SPARCstation 1+ workstations interconnected by an ethernet and the iPSC/860 hypercube. The results obtained on these platforms are then compared to highlight the limitations of current network-based distributed systems which prevent them from being used as platforms for high performance distributed computing. Extrapolation is used to demonstrate how, by increasing the application level transfer rates, a network of workstations can deliver performance comparable to that offered by supercomputers. Furthermore, we study the current computer networks and their supported protocols to identify the requirements needed to achieve low latency communication. The design features of a High Performance Interface Processor (HiP) which meet the proposed requirements, are presented.

A proposed project at Syracuse University will use HiP and the high speed communication protocol to transform high performance computers like the CM5, DECmpp 12000, nCUBE 2 and workstations including SUN SPARCstation's, IBM RISC System/6000, DECStation's etc. available at NPAC, into a system for high performance distributed computing and a platform for the applications currently being developed at NPAC.

Acknowledgment

The presented research is sponsored by DARPA under contract #DABT63-91-k-0005. The content of the information does not necessary reflect the position or the policy of the Government and no official endorsement should be inferred.

Use of the Intel iPSC/860 was provided by the Center for Research on Parallel Computation under NSF Cooperative Agreement Nos. CCR-8809615 and CDA-8619893 with support from the Keck foundation.

We would like to thank Jack Dongarra for making a preliminary version of LAPACK available to us and to Susan Ostrouchov for her help with LU factorization on the iPSC/860. We would also like to thank Gregor von Laszewski and Jongbaek Park for their contributions in the presented work.

References

[1] 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.

[2] Greg Chesson, "The protocol engine project", *in Proceedings of the Summer 1987 USENIX Conference*, pp. 209–215, June 1987.

[3] Jongbaek Park and Salim Hariri, "Architectural support for high-performance distributed system", Technical report, Department of Electrical and Conputer Engineering, Syracuse University, 1992.

[4] Geoffrey C. Fox, Seema Hiranandani, Ken Kennedy, Charles Koelbel, Uli Kremer, Chau-Wen Tseng, and Min-You Wu, "Fortran d language specifications", Technical Report SCCS 42c, Northeast Parallel Architectures Center, Syracuse University, 111 College Place, Syracuse NY 13244-4100; CRPC-TR90079, Center for Research on Parallel Computation, Rice University, Houston, TX 77251-1892, Dec. 1990.

[5] Alok Choudhary, Geoffrey C. Fox, Sanjay Ranka, Seema Hiranandani, Ken Kennedy, Charles Koelbel, and Chau-Wen Tseng, "Compiling fortran 77d and 90d for mimd distributed -memory machines", Technical Report SCCS 251, Northeast Parallel Architectures Center, Syracuse University, 111 College Place, Syracuse NY 13244-4100; CRPC-TR92203, Center for Research on Parallel Computation, Rice University, Houston, TX 77251-1892, Mar. 1992.

[6] Geoffrey C. Fox, A. Gaber Mohamed, Gregor Von Laszewski, Manish Parashar, Neng-Tan Lin, and NangKang Yeh, "High performance scalable matrix algebra algorithms for distributed memory architectures", Technical Report SCCS-271, Northeast Parallel Architectures Center, Syracuse University, 111 College Place, Syracuse, NY 13244-4100; CRPC-TR92211, Center for Research on Parallel Computation, Rice University, Houston, TX 77251-1892, Mar. 1992.

[7] A. G. Mohamed, Geoffrey C. Fox, and Gregor von Laszewski, "Blocked lu factorization on a multiprocessor computer", Internal Report SCCS-94b, Northeast Parallel Architectures Center, Syracuse University, 111 College Place, Syracuse NY 31244-4100; CPRC-TR92212, Center for Research on Parallel Computation, Rice University, Houston, TX 77251-1892, April 1992, To appear in: Microcomputers in Civil Engineering.

[8] V. S. Sunderam, "Pvm: A framework for parallel distributed computing", Technical report, Oak Ridge National Laboratory, Oak Ridge, Tennessee 37831, 1991.

[9] Adam Beguelin, Jack Dongarra, Al Geist, Robert Manchek, and Vaidy S. Sunderam, "A users' guide to pvm: Parallel virtual machine", Technical Report ORNL/TM-11826, Oak Ridge National Laboratory, Oak Ridge, Tennessee 37831, July 1991.

[10] S. F. Nugent, "The iPSC/2 direct-connect technology", *Third Conference on Hypecube Concurrent Computers and Applications*, vol. 1, pp. pp. 51–60, 1988.

[11] 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.

[12] 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*, pp. pp. 287–290, 1991.

[13] Ed Anderson, Annamaria Benzoni, Jack Dongarra, Steve Moulton, Susan Ostrouchov, Bernard Tourancheau, and Robert 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*, 1991.

[14] E. Anderson, Z. Bai, J. Demmel, J. Dongarra, J. Du Croz, A. Greenbaum, S. Hammarling, A. McKenney, and D. Sorensen, *Preliminary LAPACK Users' Guide*, Netlib, Oak Ridge National Laboratory, 1991.

[15] Jack Dongarra and Susan Ostrouchov, "Lapack block factorization algorithms on the intel iPSC/860", Technical Report LAPACK Working Note 24, Department of Computer Science Technical Report, University of Tennessee, 1990.

[16] A. G. Mohamed and D. T. Valentine, "Taylor's vortex array: A new test problem for navier-stokes solution procedures", in J.H. Kane, A.D. Carlson, and D.L. Cox, editors, *Solution of Superlarge Problems in Computational Mechanics*, pp. 167–181, Plenum, New York, NY, 1989.

[17] G. Fox, M. Johnson, G. Lyzenga, S. Otto, J. Salmon, and D. Walker, *Solving Problems on Concurrent Processors*, Prentice Hall, New Jersey, 1988.

[18] A. G. Mohamed and D. T. Valentine, "Numerical predictions of turbulent flow in an annular pipe", in G.L. Kinzel and S.M. Rohde, editors, *Proceedings of the ASME International Computers in Engineering*, vol. II, pp. 471–479, Boston, Massachusetts, Aug 1990. ASME, New York.

[19] A. G. Mohamed, D. T. Valentine, and R. E. Hessel, "Numerical study of laminar separation over an annular backstep", *Computers & Fluids, Pergamon Press*, vol. 20, pp. pp. 121–143, 1991.

[20] J. Dongarra, F. G. Gustavson, and A. Karp, "Implementing linear algebra algorithms for dense matrices on a vector pipeline machine", *SIAM Review*, vol. 26, pp. pp. 91–112, Jan 1984.

[21] H. T. Kung, "Gigabit local area networks: A system perspective", *IEEE Communications*, vol. 30, pp. 79–89, Apr. 1992.

```
subprogram kji-SAXPY-parallel (m, n, a, lda, ipiv)
 1. Processor 0 starts the pipeline
    proc = 0
    IF ( myid = proc ) THEN
      * call noblock routine (SGETF2) to factorize 1st panel
      * SEND factorized panel & pivots to all processors using
        BLACS routines (SGESD)
    ENDIF
 2. Repeat for each panel
    DO j = 1, n, β
      IF ( proc = myid ) THEN
        * copy factorized panel into work matrix
        * apply pivoting permutations to work matrix
          and rest of my panels
      ELSE
        * RECEIVE factorized panel & pivots from
          processor proc (SGERC)
        * apply pivoting permutations to work matrix
          and rest of my panels
      ENDIF
      proc = mod(proc + 1,Processors)
      IF ( I have panels left to factorize ) THEN
        IF ( proc = myid ) THEN
          * call STRSM to compute jth block row of next panel
          * call SGEMM to update the next panel only
          * call SGETF2 to factorize next panel
          * SEND factorized panel & pivots to all processors
            (SGESD)
        ENDIF
        * all processors update their remaining panels by
          calling STRSM & SGEMM
      ENDIF
    ENDDO
```

Figure 3: Pipelined Parallel kji-SAXPY

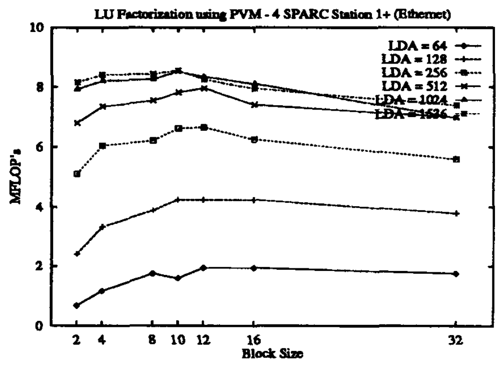


Figure 4: Effect of block-size (message size) on performance. Varying matrix sizes.

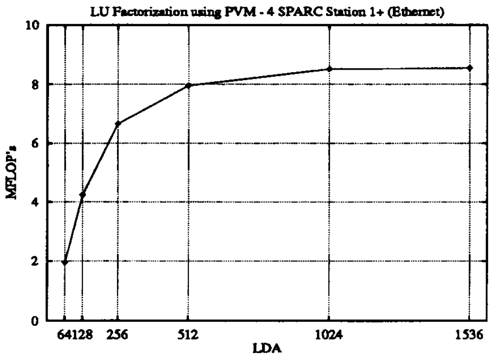


Figure 5: Peak MFLOP's obtained. Varying matrix sizes.

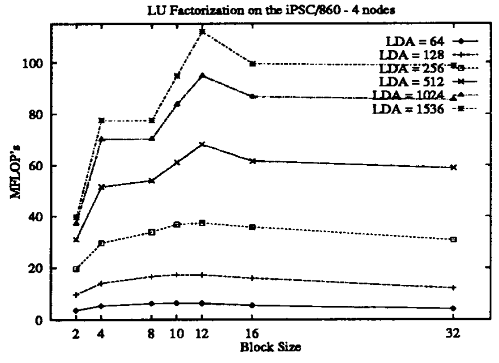


Figure 8: Effect of block-size (message size) on performance. Varying matrix sizes.

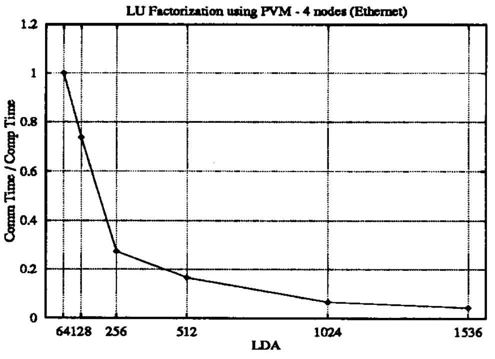


Figure 6: Variation of Communication Time/Computation Time with problem size.

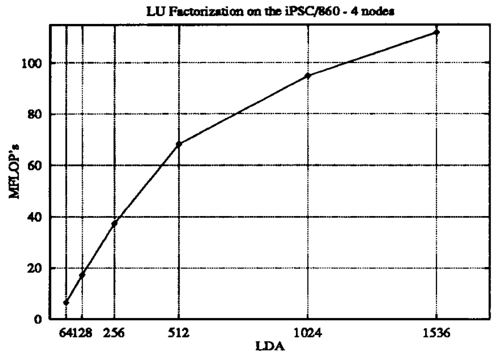


Figure 9: Peak MFLOP's obtained. Varying matrix sizes.

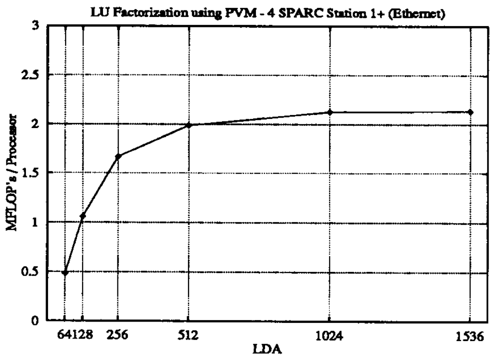


Figure 7: Peak MFLOP's/Processor obtained. Varying matrix sizes.

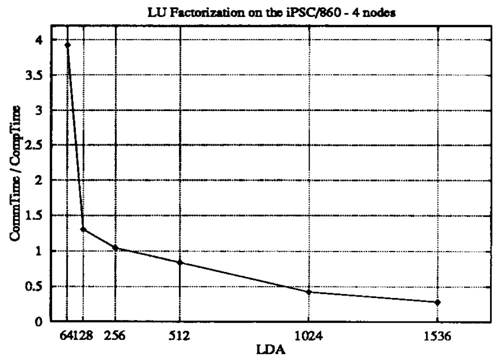


Figure 10: Variation of Communication Time/Computation Time with problem size.

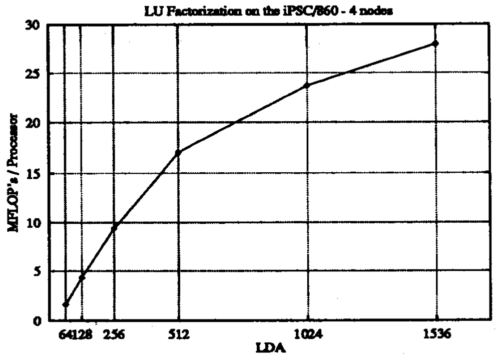


Figure 11: Peak MFLOP's/Processor obtained. Varying matrix sizes.

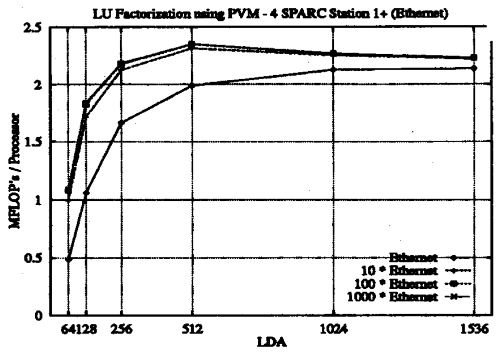


Figure 12: Extrapolated MFLOP's/Processor for effective data rates scaled up by factors of 10, 100 & 1000.

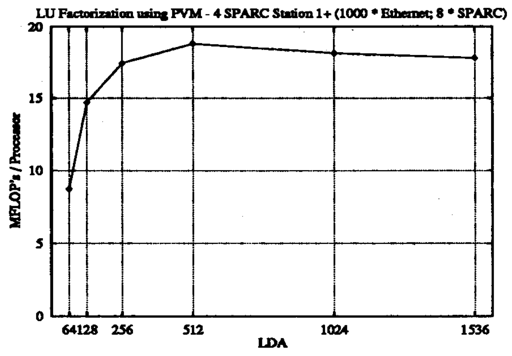


Figure 13: Extrapolated MFLOP's/Processor for processing capacity equivalent to the i860; effective data rates scaled up by a factor of 1000.

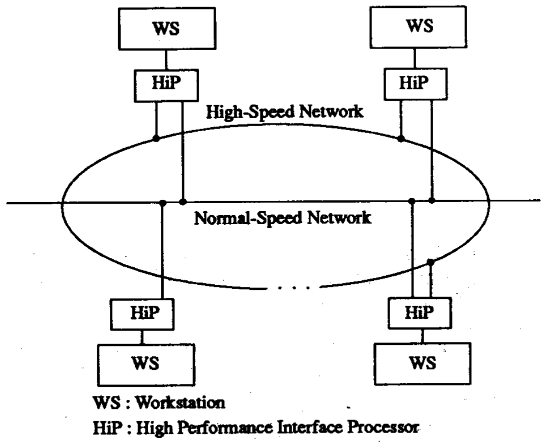


Figure 14: HiP based LAN

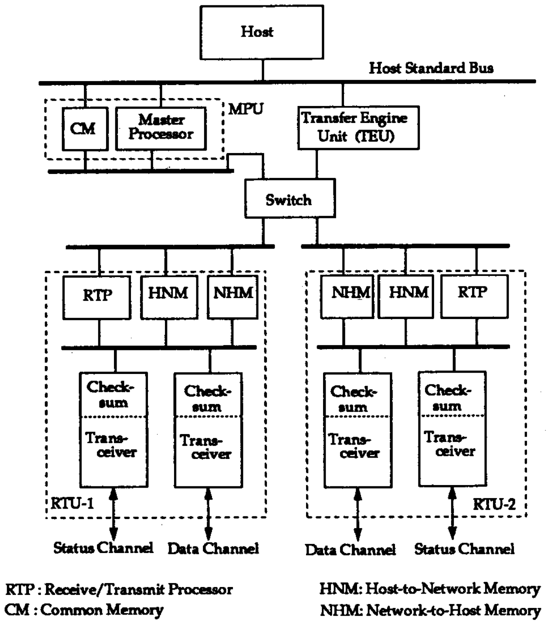


Figure 15: HiP Block Diagram