Development of an UltraNet Based Distributed Visualization Application

John Krystynak*
Computer Sciences Corporation
NASA Ames Research Center
Moffett Field, CA 94035

Abstract

The example application is a distributed visualization involving a supercomputer and a graphics workstation. The visualization computation is performed on a Connection Machine, and the results are rendered using a Silicon Graphics workstation. The UltraNet network installed at NAS allows high-bandwidth communication between the computers. Ideally, taking advantage of the UltraNet is no more complex than developing TCP/IP and Unix BSD socket-type applications on a single machine. In practice, there are several problems in developing an application using the UltraNet. This paper identifies potential problems and discusses techniques for overcoming them. Performance of UltraNet communication is measured and found to be 10 MB/sec for SGI VGX workstations.

1 Introduction

This paper examines the issues in creating an UltraNet TCP/IP socket-based distributed application. In order to preserve as much generality as possible, the discussion is focused on the UltraNet-related aspects of the development process. This paper provides:

- An explanation of the UltraNet.
- A summary of the socket model used in UltraNet communications.
- The steps in the development of a distributed application.
- A performance survey of the UltraNet.

The potential for effective distributed applications at the Numerical Aerodynamic Simulation (NAS) facility of NASA Ames Research Center grew with the installation of the UltraNet. The UltraNet is a hub-based network. The NAS installation currently consists of nine hubs connected with fiber optic cables, forming a ring which supports 1 Gigabit/sec transfer rates between the hubs. The hubs connect to over 80 workstations, supercomputers and other nodes with various types of interfaces and cable connections.

The UltraNet software allows existing socket-based communications applications to be relinked and run on UltraNet with a minimum of programming effort. After a socket-based program is relinked, however, there are still several ways to enhance performance, reliability and programmability.

Using a distributed visualization application as an example, this paper examines application design and implementation on the workstation and supercomputer sides, division of work between the machines and other issues. Specific details of the rendering and graphics algorithms in the example application are not emphasized except where they are relevant to the use of the UltraNet.

Distributed graphics applications involving a supercomputer, workstation and high-speed network are rapidly becoming an important tool for visualization [3, 4, 5, 6]. Computation is done on the supercomputer, and visualization can be done over the network, with the displays generated on a workstation. This type of distributed graphics application matches the right tools for the right tasks. Supercomputers have the memory and speed to solve the simulation and generate the data to be visualized. Graphics workstations have better tools for developing visualization software and their cost and size allow a researcher to have one on her desk.

UltraNet performance varies widely depending on the hardware and software using the network. The effective transfer rate of data over the UltraNet depends primarily on the slowest node involved. With a VME based interface on the SGI workstations, one may ex-

*Work Supported by NASA Contract NAS 2-12961

SERVER	*CLIENT*
`socket`	
`bind`	
`listen`	
	`socket`
`accept`	

Table 1: Server/Client Model: Steps in establishing a socket connection.

pect five to 10 megabytes/sec (MB/sec). Timings to support this expectation are presented in the section on performance.

2 Basic UltraNet Behavior

The UltraNet and the device drivers that support it rely on a UNIX device model to send and receive data. UltraNet connections behave like raw devices. Data is moved directly from or to the application memory by Direct Memory Access hardware (DMA) or by interrupt level processing. Hardware 'packet size' is transparent to users. The UltraNet can write whatever size buffer you wish to send, up to system-dependent limits. Currently, the largest SGI/VME write that succeeds is about 3.6 megabytes.

The model of file I/O is a good approximation of UltraNet behavior. In UNIX, the system calls **`read, write, open`** and **`close`** are provided to allow users to handle files. Although a file system may have some atomic size, users generally do not care what that size is. Sockets use the same system calls that file I/O uses: **`read, write`**, etc. More specifically, UltraNet sockets are stream facilities which provide full-duplex communication paths between user's processes and the kernel's interactions with the network hardware.

The speed of delivery of a read or write depends on the speed of the machines involved. The 'weakest link in the chain,' in terms of I/O throughput, will determine the overall speed of the transfer. Data is read directly to and from memory. A VME interface (e.g., SGI VGX, Convex) to the UltraNet is slower than an HiPPI interface (e.g., CM-2)

3 The Socket Model

The sockets in the example application are handled in a client/server model. One process listens for requests for connections and makes the connection when necessary ('server'). The other process ('client') asks for a connection from some server. When the client gets the connection, both server and client can read and write to the socket. The client must know the machine address and port number of the server to be able to connect. The socket library calls which establish a server and client connection are shown in Table 1. Note that the server must have completed the first three steps before the client does a **`socket`** call or the connection will fail.

The UltraNet socket compatibility library supports most of the UNIX socket calls. Some minor variations listed below are not currently supported:

- **`sendmsg, recvmesg`**
- **`shutdown`**
- **`listen(s, n)`** where **`n`** > 1
- **`send`** and **`receive`** are not interruptible

TCP/IP style connections should be used in most applications, since they are a reliable connection-oriented protocol, as opposed to connectionless UDP datagram protocols. UDP does not provide error control, flow control or sequencing. Several UNIX books give examples of UDP style sockets, which are not generally useful for applications sending large amounts of data. For more detail on sockets and TCP/IP, see [8].

When using an SGI with UltraNet, it is necessary to link with the **`ulsock`** library. The **`ulsock`** library provides socket calls that work with UltraNet. A few important differences arise when using the **`ulsock`** library. The **`ulsock`** library replaces the **`dup2`** function from C with its own **`dup2`**. A potential conflict exists with the **`mpc`** parallel programming library, which also replaces **`dup2`**. For UltraNet linked programs which use both **`ulsock`** and **`mpc`**, place **`-lulsock`** before **`-lmpc`** on the compile command line (i.e., ensure that your program uses the UltraNet **`dup2`**).

To use the UltraNet, it is necessary to attach to the host address of the UltraNet interface. Machines with UltraNet interfaces have internet addresses dedicated to an UltraNet native path, an UltraNet internet protocol path and a general internet protocol path. Table 2 shows typical NAS names of addresses on a given network. It is easy to switch between the networks by changing addresses of server connections. The performance discussion of this paper deals only with the native UltraNet addresses (i.e., hostnames in the form **`host-u`**).

HOSTNAME	*NETWORK*
`host`	Ethernet addresses
`host-uip`	UltraNet internet protocol address
`host-u`	Native UltraNet address

Table 2: Example NAS Hostnames and their Networks for UltraNet Hosts. Native UltraNet is generally the fastest; Ethernet is slowest.

UltraNet Name	*CMFS Name*
AF_INET	CMFS_AF_INET
`socket`	`CMFS_socket`
`read`	`CMFS_read_file_always`
`struct sockaddr`	`struct cm_sockaddr`
`perror`	`CMFS_perror`

Table 3: CMFS Socket Library Name differences

4 The Distributed Application: A Case Study

The production of a working UltraNet application is not extremely complex; however, network reliability and access to network resources limits how development takes place. The application examined here involves the Connection Machine (CM), and a Silicon Graphics VGX (SGI). The Connection Machine side requires use of the Connection Machine High Performance Parallel Interface (CM-HiPPI) processor, the CM DataVault, the CM front-end and at least one Sequencer on the CM itself. The SGI side of the application involves an SGI VGX running two processes accessing shared memory buffers governed by semaphores. Both sides depend on the UltraNet.

The sum of the parts forms a working UltraNet application, but it is easier to debug and develop the individual parts of the application separate from one another. The amount of hardware involved in the full application limits developing and debugging. If the network or the DataVault or the CM is not up, the full application cannot run. Separate development of parts allows work to proceed even when all the hardware elements are not available. As an example, the SGI side can operate with data coming from another process (on the same machine or from another workstation) over the Ethernet. The SGI side can also operate without the graphics process. For testing connections from the CM, a simpler socket program is used which can isolate network problems and benchmark network response.

Some other development obstacles include: availability of CM time, access to SGI VGX graphics console, CM-HiPPI and UltraNet hardware problems. A list of the major steps in the development process shows how the application progressed.

1. Developed client/server procedures between SGIs.
2. Ported client to CM.
3. Developed single process SGI application.
4. Wrote multiple process SGI producer/consumer.
5. Fused producer/consumer with sockets on SGI.
6. Wrote serial process to simulate CM for debugging.
7. Incorporated CM side.
8. Integrated and tested full application.

All communication is based on procedures written in Step 1. The code for these routines is given in Appendix A. The two major procedures are `server()` and `client()`. The client routine takes a hostname as an argument and attempts to connect over a previously agreed upon port number to the server. The port number can be defined in an include file visible to both server and client processes.

Once the basic communication routines are debugged, it is possible to test UltraNet throughput. These tests are discussed below. It is also relatively easy to port these communication routines to the Connection Machine, since Thinking Machines provides analogous library calls in their CM File System (CMFS) library [9]. The major difference is in the naming. Some examples are shown in Table 3. The `cmclient()` source code is included in Appendix B. Both the client and server codes in the appendices are based on examples in [8].

4.1 Partitioning the Application

Partitioning the work of a distributed application is an important design decision which affects the performance and utility of the application. The nature of the application, along with the relative speeds of the computers and networks involved determine how partitioning of work should be done. Several possibilities in dividing the work for interactive visualization applications have been explored [3, 5].

One technique is to do both simulation and rendering computations on one or more supercomputers and display the results on a workstation. This approach allocates heavy computation to the supercomputer and

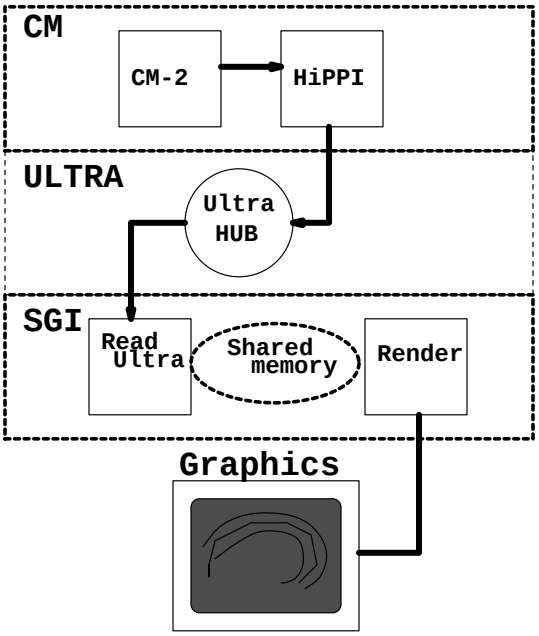


Figure 1: Basic architecture of distributed application. The CM communicates data via the CM-HiPPI to the UltraNet Hub. The SGI has a process devoted to reading buffers from the Hub into Shared memory. Another SGI process handles rendering and user interface.

relegates the workstation to displaying precomputed bitmap images. Sending a full screen color bitmap to the workstation may require an extremely fast network to support animation. This model of partitioning is analogous to the model the X windows system uses for distributing graphics. This type of partitioning is useful for computationally expensive rendering problems such as volume visualization, where supercomputers can process the image much faster than a workstation can.

The partitioning for the example application was designed to allow the workstation to handle rendering and user interaction. The amount of data sent to the workstation over the network is small. The Connection Machine calculates new data positions, then sends 3-D coordinates of the positions to the workstation for rendering. The workstation handles interaction with the user, rotation, scaling and lighting of the visualization. Figure 1 illustrates the basic architecture of the application.

It is important to balance the amount of network traffic and the computation requirements on the workstation and CM to get acceptable throughput. The CM-HiPPI is also more efficient with buffer sizes greater than 256KB (or 4 32-bit floating point values on each processor of 1 sequencer). One drawback of this behavior is that at transfer sizes where the CM-HiPPI and UltraNet are most efficient, the overall transfer time can be rather long. If one communicates in buffers of 2MB or more from the CM to an SGI, each transfer takes around 0.5 secs to complete. This is too slow for many graphics applications, so software buffering may be considered to amortize the cost of data transfers.

For the case where a small amount of data needs to be sent relatively often, similar problems arise. As shown in the performance section below, sending less than 256KB of data is non-optimal because of startup cost in buffer communication, so packaging data to be sent into larger buffers may give better UltraNet throughput. The delay time in waiting for enough data to send may not be acceptable for interactive applications. Another alternative is to use Ethernet for small buffers. These tactics are application-dependent, because the amount of buffering that will help depends on the ratio of computation speed to network speed and to workstation rendering speed. The performance section of this paper gives timings which may help developers predict what type of buffering is most appropriate for specific applications.

4.2 Workstation Application Architecture

Since the example application requires the SGI to render 3-D graphics while also reading UltraNet data into memory, a multiple process architecture was used. A producer/consumer scheme is used for efficiency. The producer is the process which reads data from the UltraNet. The producer tries to have data ready for the consumer at all times. The consumer 'consumes' this data when it needs to update the screen. Reading data from the UltraNet continuously with one processor, while rendering previously read buffers allows the application to be more interactive. This tactic is one of the suggestions in the UltraNet Network Applications Development guide [10], and is especially productive on multiple processor machines such as the VGX.

The producer/consumer code and the networking and rendering code are independent units which were integrated into one unit after they were debugged. To simplify development, the producer/consumer prototype was based on an SGI documentation example [7], which gave insight into how the producer/consumer shared memory code would work. A second prototype was a single process which did two tasks: read the network data and render data. Later, this prototype was

integrated into the two process producer/consumer code without major problems. Figure 1 shows the SGI section of the application as two processes sharing memory. The whole SGI section in Figure 1 is comprised of the integrated prototypes described above.

The serial rendering and networking code and the non-networked two process producer/consumer were combined with relative ease, because they had been independently debugged and their behavior was well defined. The basic software engineering premise of prototyping small modules to be incorporated into the larger architecture after they have been tested guided the application's development. It was necessary to proceed according to this premise because many of the hardware and software components were new and somewhat unreliable. In other words, guessing at how to integrate a large number of unfamiliar concepts would not have worked.

When using shared memory and two processes with UltraNet sockets, it is important that the processes' critical sections are handled correctly. If the control threads are not handled correctly, the reads and writes to the UltraNet will not match up and a form of deadlock will persist. If UltraNet reads and writes are not paired, both the processes and the port they use will be severely hung[1] because there is no time-out mechanism. Modifying and testing the SGI example before integrating network communication code allowed validation of the multiple process code without the problems of process and network deadlock.

4.3 Supercomputer Application Architecture

The construction of the application for the Connection Machine was analogous to the process on the workstation. Figure 1 shows the CM-2 and the HiPPI as independent blocks through which data flows. Although these are separate hardware units, they also provide a model for the partitioning of the CM section of the example application. Two modules were developed: a standalone particle tracing code (CM-2) and a CMFS socket-based communication code (HiPPI). The particle code provides the data which the HiPPI communicates to the UltraNet. The development of the particle tracing code is not discussed in this paper. The socket code, however, illustrates how the UltraNet interface may be different from machine to machine.

[1] This time-out problem was alleviated in the CMFS sockets package, which greatly reduced the number of dead processes and hung TCP ports during development.

The primary concerns when reading or writing network data on the Connection Machine are parallel/serial data format and byte-ordering. The Connection Machine stores a floating point number in *parallel* format which must be transposed into *serial* format before sending. Likewise, incoming data must be transposed from serial to parallel format before it is useful to the CM. Furthermore, the Connection Machine has a little endian byte order, while the SGI VGX has big endian byte order, which means bytes must be swapped for data to be the same on the CM and SGI. Byte swapping can be done quickly for large amounts of data in parallel on the CM. The byte-swapping transformation is given by:

$$ABCD \longrightarrow DCBA$$

If the byte swapping is done on the CM, it must be done before the data is transposed to serial format.

Currently, the `CMFS_transpose_always` call, which does the transpose from serial to parallel or vice-versa, is somewhat slow and hampers the effective performance of transfers to and from the CM. Future software releases should improve the performance of this transpose.

The CM-HiPPI handles UltraNet data for the CM. The CM-HiPPI is a Sun 4 which runs the socket server daemon. The socket server handles all socket connections to and from the CM, so the socket server must be running or the CM cannot access the UltraNet. When the CM-HiPPI was first installed there were several problems with the socket server and the hardware interfaces on the CM-HiPPI. The reliability of the software and hardware improved as bugs were found and fixed by Thinking Machines. To continue development when the CM-HiPPI was down, a "CM simulator" was written which ran on a VGX. This process simply read pre-computed data from disk and connected to the SGI process with UltraNet sockets.

5 UltraNet Performance

There are several factors which affect an application's UltraNet throughput. Most influential is the hardware interface to the UltraNet. The UltraNet as configured at NAS supports 1 gigabit/sec transfer rates between its hubs and 250 megabits/sec transfers (about 32MB/sec) between nodes. As mentioned above, the SGI VME interfaces and the CM-HiPPI interface do not provide 100% of this throughput. When transferring data over the UltraNet, the *slowest* interface sending or receiving data determines the maximum rate for the transfer. In the application discussed

in this paper, the SGI VME interface is the bottleneck in transfer speed. The VME interface is the hardware that accesses the SGI's bus and communicates data to the UltraNet hub over fiber-optic connections.

Given a fixed set of hardware, the key parameters which affect performance are:

1. Size of reads/writes.
2. Network traffic.
3. Data path from host to host.

Of these, the first — read/write size — is both user determinable and very influential in achieving optimal performance. Since users on given hosts cannot easily change network traffic or the data path of the transfer, these factors are not examined.

5.1 SGI VME Interface Performance

The UltraNet is connected to several dozen SGI's at NAS. The performance of the SGI and VME Interface to the UltraNet will affect many distributed applications written at NAS. A series of timings of transfer rates was conducted for read and write directions from SGI to SGI. Tests were also done on single machines in a 'loopback' manner. The results of these benchmarks as well as a comparison of Ethernet and UltraNet transfer rates are presented in the following discussion. Timings for reads and timings for writes are nearly equivalent so that read and write operations can be considered equivalent in performance. These timings demonstrate:

- Reading or writing buffers smaller than 256 kilobytes (KB) is inefficient, and sizes of 1 to 2 megabytes consistently give best performance for the UltraNet.
- Ethernet is more efficient than UltraNet for small data transfers.
- The DMA on the SGIs affects performance of large transfers.
- The largest size buffer the SGI VME interfaces reliably accept is about 3.6MB.

The processor and I/O facilities of a 33Mhz SGI VGX can support transfer rates of at most 12MB/sec. Figure 2 shows that VME to VME transfer rates of 10-12MB/sec can be sustained when buffer sizes are larger than 256KB. Two machines communicating to each other over UltraNet can each support this rate

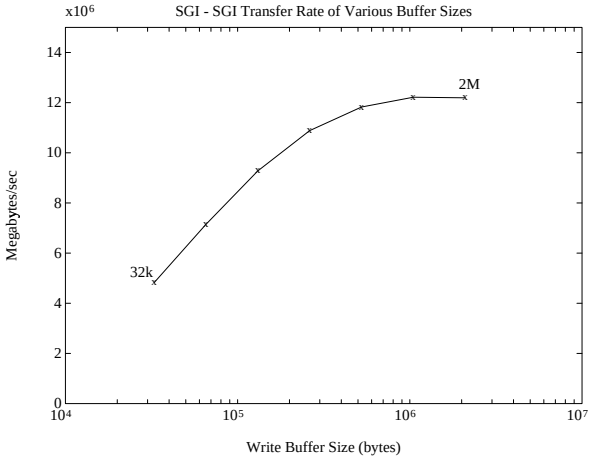


Figure 2: UltraNet transfers of 30 megabytes of data from SGI VGX to SGI VGX on the same UltraNet Hub. Points indicate size of buffer transferred, starting with 32KB , ending at 2 megabytes.

due to the combination of their processor(s) speed and the VME I/O performance. When two machines are involved, each can devote its full I/O throughput to reading or writing.

UltraNet is not as fast as Ethernet when transferring buffers smaller than 10KB. The overhead in using UltraNet is greater than Ethernet's overhead for these small buffer sizes. Figure 3 demonstrates that EtherNet is over twice as fast as UltraNet for buffer sizes between 10 bytes and 5KB. Ethernet's maximum speed is reached at 1KB. UltraNet betters Ethernet's maximum speed at sizes above 10KB. If small buffers are being transferred over and over, Ethernet provides better throughput. It is possible to have both types of connections active at the same time. As an example, one might send control information (e.g., mouse position, visualization parameters) between two machines over Ethernet, while concurrently communicating data over UltraNet.

A 'loopback' test on a single machine shows that UltraNet uses the VME interface, even when it is reading and writing to local memory. Figure 4 shows that writing from one process to another on the same machine limits the write speed to around 6 MB/sec. This is due to the single VME interface doing both reads and writes. The two machine SGI rates of 12 MB/sec can only be achieved when a single VME I/O board is devoted to handling a single connection.

The DMA hardware on the SGI influences the UltraNet transfer rate. The DMA on the SGI does the read or write from user memory to the UltraNet de-

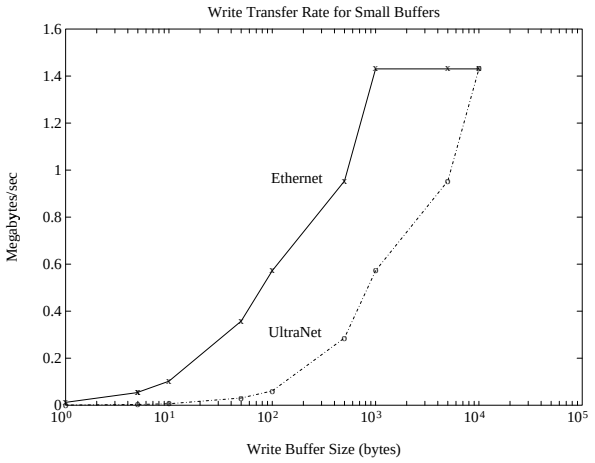


Figure 3: Transfers of 3 megabytes of data from SGI VGX to SGI VGX over Ethernet and UltraNet. Ethernet is more than twice as fast as UltraNet for buffer sizes smaller than 5KB. At buffer sizes of 10KB UltraNet is as fast as Ethernet.

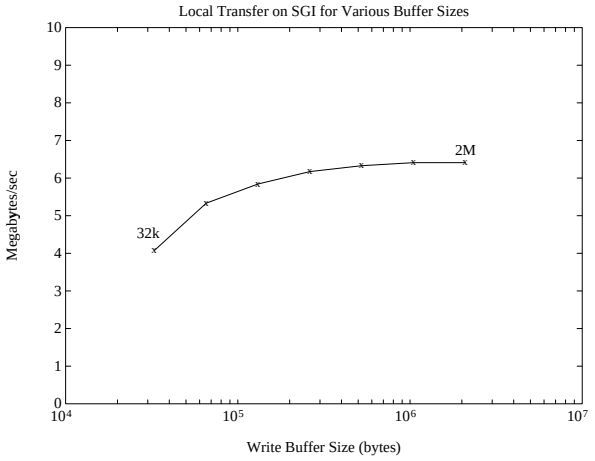


Figure 4: 30 megabytes transferred between two processes on the same workstation. Peak rates are limited by one VME I/O channel handling reads and writes for both processes simultaneously. Compared to speeds for two machine transfers, local transfers are about half as fast.

vice. The performance of reads and writes may be limited by the DMA if the DMA's address space fills up with data from the read or write.

The nature of the DMA limitation is hard to demonstrate with two machines because the sockets will not handle buffers large enough to swamp the DMA. Single machine loopback tests doing reads and writes using a single DMA at the same time reveal a performance limitation. Transfer rates suffer once the combination of read and write buffers approaches a certain size. Figure 5 shows for local buffer transfer sizes greater than 2.5 MB there is a large drop-off in buffers transferred per second at the 2.5 MB buffer size.

Since two machines communicating have twice as much DMA memory space as one machine, and one machine encounters problems at 2.5MB, problems may arise when sending buffers greater than 5MB between two machines. It is impossible to confirm this hypothesis on two machines because the current UltraNet drivers for SGI will not complete 5MB transfers between VME interfaces. Eventually, however, the DMA limitation may prove problematic since the UltraNet protocol specification does allow buffer sizes up to 64MB to be sent.

These large buffer cases are not too worrisome for many application developers, since sending this much data (4MB or more) is too slow for interactive use and is a large amount of data for a workstation to process. For example, if an application needed to send 4MB per frame for graphics, the UltraNet and VME interfaces would be too slow to support animation frame rates.

The hardware design of the UltraNet VME board also influences throughput, but very few specific details are available about the board's architecture. UltraNet is a proprietary system, and documents explaining the interface board are not generally available.

UltraNet performance between various workstations, minisupercomputers and supercomputers was measured by an Ultra Network Technologies employee before the installation of the full UltraNet configuration at NAS [2]. This work contains more detailed explanation and analysis of performance for Sun, Alliant and Convex VME Interfaces, but does not cover SGI performance.

5.2 CM to SGI Timings

The Connection Machine has an HiPPI based interface and two 32MB/sec I/O busses able to connect to the UltraNet. Since the UltraNet supports 32MB/sec sustained, while the CM supports at most 64MB/sec

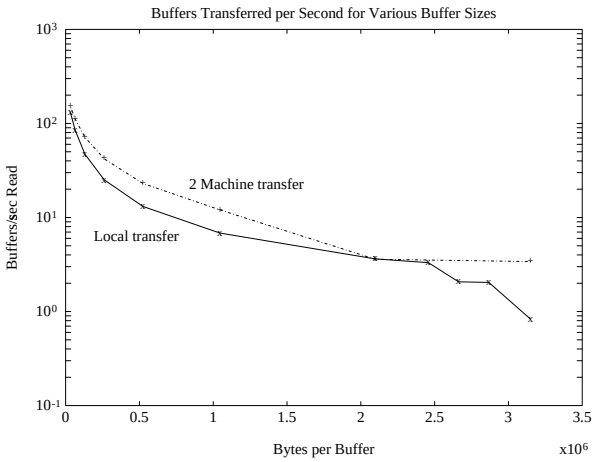


Figure 5: Buffers per second transfer rate for SGI between two machines and between two processes on same machine. Two machines are about twice as fast as local machine transfers for buffers smaller than 2.5 MB. Above 2.5 MB, two machine transfers are significantly faster.

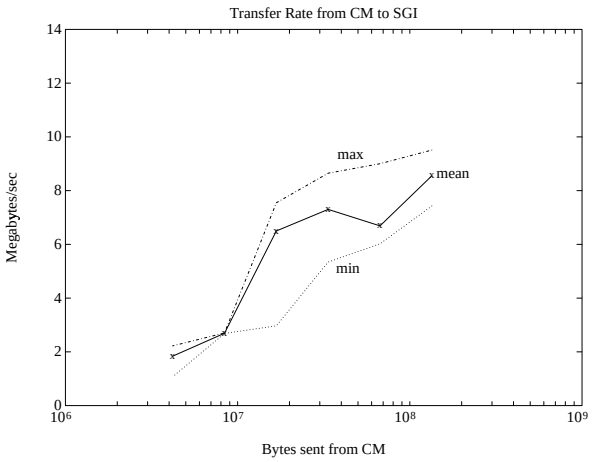


Figure 6: One Connection Machine sequencer writing to SGI VGX over UltraNet. Mean, min and max of 10 tests at each size. VGX read buffer size is held constant at one megabyte. Lowest rate is for 512 bytes sent from each of 8,192 processors or a total of 4MB. Highest rate is for 16k from each processor (128MB total).

(2 busses × 32MB/sec per bus), the maximum transfer rate from the CM to any UltraNet connected device is 32MB/sec. Transfer at these rates are theoretically possible between the CM and the Cray. Becker and Dagum investigated this type of connection for relatively small transfers [1].

For applications involving workstations, the workstation I/O will generally determine the transfer rate. Figure 6 illustrates average throughput from CM to a VGX. In these tests, conducted using CM system software version 6.0, the SGI read 30 megabytes of data from the CM in one megabyte chunks. The time to byte-swap and convert from parallel CM representation to serial representation (both done on the CM) is included in the timings. The timings show performance under 10-12MB/sec for all sizes of writes. The parallel to serial transpose is relatively slow, and limits overall performance. In release 6.1 of the CM System software, the efficiency of the parallel to serial transpose should be much better and throughput should therefore be closer to the SGI/VME maximum 10–12MB/sec.

A transfer of a 32-bit floating point number from each processor of one sequencer (8,192 processors) on the CM contains a total of 32KB of data. This is the minimum effective size the CM-HiPPI can transfer. This limitation may affect performance for codes which need to transfer a smaller amount of data. The CMFS sockets package is used when transferring data from each processor to the UltraNet. Currently, CMFS sockets are only compatible with the fieldwise model of the CM. Using the UltraNet from any slicewise program will be highly inefficient until the software directly supports the slicewise data layout. Further investigation into CM-HiPPI and CM to UltraNet performance is being postponed until the more efficient parallel to serial transpose software arrives, since the transpose currently dominates other factors in determining CM transfer rates.

6 Summary

Taking advantage of the UltraNet for distributed applications is not a trivial task; however, it is becoming easier. This paper has examined the development of one such application. This application demonstrated that development for a relatively new network and combination of computer architectures is made easier when it is done piece by piece. Comprehension of UltraNet behavior was gained by running simple benchmarks. The communications primitives used for these benchmarks were later modified to form the basis for the application's communication code. Later, single process prototypes were modified to run in the full networked, multiple process application. By isolating and prototyping, software could be developed

even when hardware was unavailable.

The maximum throughput of the UltraNet depends on the hardware attached. The SGI VGX can utilize 10–12MB/sec of UltraNet bandwidth. The hardware of the SGI/VME interface performs as well as other workstations' interfaces but is limiting compared to the speeds of HiPPI and supercomputer UltraNet interfaces. Furthermore, the DMA of the SGI VGX may limit throughput when handling buffer sizes larger than 2.5MB on one machine, or 5MB between two machines. It is easy to adapt current socket programs to use UltraNet sockets, although to attain maximum throughput, devoting a process to network handling is a good idea.

The Connection Machine is also easily accessible to and from the UltraNet. The CM can take advantage of greater bandwidth to the UltraNet, because it is not limited like the workstations or the Convex with a VME interface. CM-HiPPI software is well-designed but currently has several limitations: there is no slicewise interface, a slow transpose is required, and language support is limited. Thinking Machines is working to remedy these problems.

The current level of UltraNet performance is high enough to support a variety of applications. For a given application, making good choices concerning how much data to transfer and when to transfer data will improve application throughput. It must be kept in mind that UltraNet and the related pieces are relatively new and immature. Undoubtedly, as UltraNet matures, software toolkits will incorporate features to make its use transparent to the programmer.

Acknowledgements

The author would like to thank Leo Dagum, Katy Kislitzin and Pam Walatka for their many helpful comments and suggestions.

References

[1] Becker, J., Dagum, L., *Distributed Particle Simulation Using Heterogeneous Supercomputers*, NAS Technical Report RND-91-012, 1991.

[2] Clinger, M., *Very High Speed Network Prototype Development: Measurement of Effective Transfer Rates,* Report to NASA Ames Research Center, October 25, 1989.

[3] Gerald-Yamasaki, M. J., *Cooperative Visualization of Computational Fluid Dynamics*, RNR Technical Report RND-91-011, March 1991.

[4] Malamud, C., *The Top-Secret Virtual Chicken,* Communications Week, October 28, pg. 12, 1991.

[5] Robertson, D. W., Johnston, W.E., Tierney, B. L., Loken, S. C., Jacobson, V. L., Theil, E. H., *Distributed Visualization Using Workstations, Supercomputers and High-Speed Networks,* Proceedings Visualization '91, October 22 - 25, San Diego, CA, 1991.

[6] Schneider, M., *Pittsburgh's Not-So-Odd Couple,* Supercomputing Review, Vol. 4, No. 8, pp. 36–38, 1991.

[7] Silicon Graphics Inc., *Parallel Programming on Silicon Graphics Computer Systems,* Version 1.0, Document # 007-0770-010 Mountain View, CA, 1989.

[8] Stevens, W. R., *Unix Network Programming,* Chapter 6., Prentice-Hall, Englewood Cliffs, NJ 1990

[9] Thinking Machines Corp., *CM HiPPI User's Guide,* Cambridge, MA, 1991.

[10] Ultra Network Technologies, *Network Applications Development Guide,* San Jose, CA, 1990.