The Measured Network Traffic of Compiler–Parallelized Programs

Peter A. Dinda Brad M. Garcia
Kwok-Shing Leung

July 1998

CMU-CS-98-144

School of Computer Science
Carnegie Mellon University
Pittsburgh, PA 15213

Abstract

Using workstations interconnected by a LAN as a distributed parallel computer is becoming increasingly common. At the same time, parallelizing compilers are making such systems easier to program, Understanding the traffic of compiler–parallelized programs running on networks is vital for network planning and for designing quality of service interfaces and mechanisms for new networks. To provide a basis for such understanding, we measured the traffic of six dense-matrix applications written in a dialect of High Performance Fortran and compiled with the Fx parallelizing compiler. The traffic of these programs is profoundly different from typical network traffic. In particular, the programs exhibit global collective communication patterns, correlated traffic along many connections, constant burst sizes, and periodic burstiness with bandwidth dependent periodicity. The traffic of these programs can be characterized by the power spectra of their instantaneous average bandwidth. These spectra can be simplified to form analytic models to generate similar traffic.

Keywords: network traffic characterization, networks of workstations, workstation clusters, parallelizing compilers

1 Introduction

As the performance of local area networks grows, it is increasingly tempting to use a cluster of workstations as a parallel computer. At the same time, presentation layer APIs such as PVM [21] and MPI [22], and parallel languages such as High Performance Fortran [10] are being standardized, greatly enhancing the portability of parallel programs to workstation clusters. Further, the parallel computing community has developed extremely efficient implementations of these APIs and languages [19, 1, 4].

As implementations continue to become more efficient, the performance of the network will be increasingly important. In addition to significantly increased connection and aggregate bandwidths, next generation LANs, such as ATM [3, 2], will supply quality of service (QoS) guarantees for connections. Parallel programs may be able to benefit from such guarantees. However, to extract a QoS guarantee from a network, an application must supply a characterization of its traffic [8]. Much of the work in traffic characterization has concentrated on media streams [9, 11], although some work on ATM call admission for parallel applications has assumed correlated bursty traffic [7]. In this paper, we detail measurements of the traffic of dense matrix parallel programs written in a dialect of High Performance Fortran and compiled with the Fx parallelizing compiler [13].

In all, we measured the network behavior of six Fx parallel programs on an Ethernet. Five of these programs are kernels which exhibit global communication patterns common to Fx programs. Fx parallelizes dense matrix codes written in a dialect of High Performance Fortran. Fx targets the SPMD machine model, as do many other parallelizing compilers. We also look at a large scale example of an Fx application, an air quality modeling application which is being parallelized at CMU in a project related to Fx [14].

The outgrowth of these measurements is the observation that the traffic of Fx parallel programs is fundamentally different from those of media streams. Specifically, parallel programs exhibit

- Global collective communication patterns
- Correlated traffic along many connections
- Constant burst sizes
- Periodic burstiness
- Bandwidth dependent periodicity

We characterize the programs' bandwidth demands by the power spectra of their instantaneous average bandwidths. These spectra directly correspond to the Fourier series coefficients needed to reconstruct the instantaneous average bandwidth at any point in time. Interestingly, these spectra are rather sparse and "spiky", which means the Fourier expansion can be limited to important spikes, forming a simple analytic model that approximates the instantaneous average bandwidth.

The paper begins by describing common communication patterns exhibited by Fx parallel programs. The next section describes each of the six programs we measured, in particular explaining how its communication pattern arises. Following this, we describe the PVM communications library used by the the Fx run-time system. Next, we describe our methodology in considerable detail. The main part of the paper presents our measurements, including the power spectrum of the instantaneous bandwidth for each of the programs. The power spectra of the programs makes their periodicity absolutely clear. Following the measurements, we discuss the results, and comment on how the power spectra can be used to build simple analytical models of the bandwidth requirements of the programs. We also discuss a QoS negotiation scheme that is more amenable to parallel programs. Finally, we conclude with an overview.

2 Communication Patterns of Fx Programs

The Fx [13] compiler parallelizes dense matrix codes based on parallel array assignment statements and targets distributed memory parallel computers using the Single Program, Multiple Data (SPMD) model. This model is the ultimate target of many parallel and parallelizing compilers. In the SPMD model, each processor executes the same program, which works on processor-local data. Frequently, the processors exchange data by message passing, which also synchronizes the processors. This message exchange is referred to as a *communication phase*. The parallel program executes as interleaved communication and local computation phases.

A communication phase can be classified according to the pattern of message exchange among the processors. In general, this pattern can be *many-to-many*, where each processor sends to any arbitrary group of the remaining processors. However, certain patterns are much more common than others, especially in dense matrix computations such as those typically coded in High Performance Fortran and Fx. For example, the *neighbor* pattern, where each processor p_i sends to processors p_{i-1} and p_{i+1} is common. Another common pattern is *all-to-all*, where each processor sends to every other processor. A third pattern is *partition*, where the processors are partitioned into two or more sets and each member of a set sends to every member of another set. Fourth, a single processor may *broadcast* a message to every other processor. Finally, the pattern can be a *tree*, where every second processor sends to its left neighbor and then drops out. This is repeated until one processor remains. Sometimes this is followed with a “down-sweep”, reversing the process. These communication patterns are summarized in Figure 1.

3 Program descriptions

The six Fx [13] programs chosen for investigation fall into two classes. Five of the programs, SOR, 2DFFT, TDFFT, SEQ, and HIST, are kernels that exhibit the communication patterns discussed in section 2. These kernels are part of the Fx compiler test suite. AIRSHED [16, 14], an air quality modeling application, represents a “real” scientific application.

3.1 Fx kernels

Five of the Fx programs, SOR, 2DFFT, T2DFFT, SEQ, and HIST, were chosen to exhibit communication patterns common to SPMD parallel programs discussed in section 2. These kernels are summarized in figure 2. For each program, we discuss the distribution of its data (an $N \times N$ matrix) over its P processors, the local computation on each processor, and the global communication it exhibits.

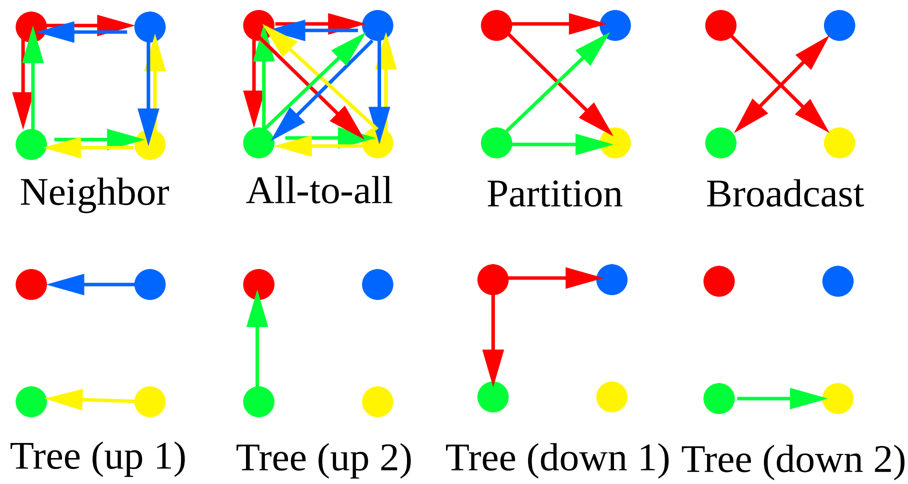


Figure 1: Fx Communication patterns

Pattern	Kernel	Description
Neighbor	SOR	2D Successive overrelaxation
All-to-all	2DFFT	2D Data parallel FFT
Partition	T2DFFT	2D Task parallel FFT
Broadcast	SEQ	Sequential I/O
Tree	HIST	2D Image histogram

Figure 2: Fx kernels

SOR

SOR is a successive overrelaxation kernel. In each step, each element of an $N \times N$ matrix computes its next value as a function of its neighboring elements. In the Fx implementation, the rows of the matrix are distributed across P processors by blocks: processor 0 owns the first $\frac{N}{P}$ rows, processor 1 the next $\frac{N}{P}$ rows, etc. Because of this distribution, at each step, every processor p except for processors 0 and $P-1$ must exchange a row of data with processor $p-1$ and processor $p+1$ before computing the next value of each of the elements it owns. In every step, each processor performs $O(\frac{N^2}{P})$ local work and sends an $O(N)$ size message to processors $p-1$ and $p+1$. SOR is our example of such a *neighbor* communication pattern.

2DFFT

2DFFT is a two-dimensional Fast Fourier Transform. Like in SOR, the $N \times N$ input matrix has its rows block-distributed over the processors. In the first step, local one-dimensional FFTs are run over each row a processor owns. Next, the matrix is redistributed so that its columns are block-distributed over the processors. Finally, local one-dimensional FFTs are run over each column a processor owns. Each processor performs $O(\frac{N^2 \log N}{P})$ work and generates a $O(\left(\frac{N}{P}\right)^2)$ size message for every other processor. 2DFFT is our example of a *all-to-all* communication pattern.

T2DFFT

T2DFFT is a pipelined, task parallel 2DFFT. Half of the processors perform the local row FFTs and send the resulting matrix to the other half, which perform the local column FFTs. A side effect of the communication is the distribution transpose, so each sending processor sends an $O(\left(\frac{N}{P}\right)^2)$ size message to each of the receiving processors. Notice that each message is twice as large as for 2DFFT for the same number of processors. Each processor performs $O(\frac{N^2 logN}{P})$ work. This is our example of a *partition* communication pattern.

SEQ

SEQ is an example of the kind of *broadcast* communication pattern that results from sequential I/O in Fx programs. An $N \times N$ matrix distributed over the processors is initialized element-wise by data produced on processor 0. This is implemented by having processor 0 broadcast each element to each of the other processors, which collect the elements they need. This program performs no computation, but processor 0 sends N^2 $O(1)$ size messages to every other processor. This is our example of a *broadcast* communication pattern.

HIST

HIST computes the histogram of elements of a $N \times N$ input matrix. The input matrix has its rows distributed over the processors. Each processor computes a local histogram vector for the rows it owns. After this, there are $\log P$ steps, where at step i, processors whose numbers are odd multiples of 2^i send their histogram vector to the processors that are even multiples of 2^i. These processors merge the incoming histogram vector with their local histogram vector. Ultimately, processor 0 has the complete histogram, which it broadcasts to all the other processors. This is an example of a *tree* communication pattern.

3.2 AIRSHED Simulation

The multiscale AIRSHED model captures the formation, reaction, and transport of atmospheric pollutants and related chemical species [15]. The goal of a related research project is to convert this massive application into a portable and scalable parallel program [14]. As a part of this work, AIRSHED is being ported to Fx. However, at the time of our research, this port had not been completed. Instead, we measured an Fx skeleton of the application which was prepared by the group performing the actual port. The skeleton application models both the computation and communication of the actual application.

AIRSHED simulates the movement and reaction of s chemical species, distributed over domains containing p grid points in each of l atmospheric layers [16]. In our simulation, $s = 35$ species, $p = 1024$ grid points, and $l = 4$ atmospheric layers. The program computes in two principle phases: (1) horizontal transport (using a finite element method with repeated application of a direct solver), followed by (2) chemistry/vertical transport (using an iterative, predictor-corrector method). Input is an $l \times s \times p$ concentration array C. Initial conditions are input from disk, and in a preprocessing phase for the horizontal transport phases to follow, the finite element stiffness matrix for each layer is assembled and factored. The atmospheric conditions captured by the stiffness matrix are assumed to be constant during the simulation hour, so this step is performed just once per hour. This is followed by a sequence of k simulation steps ($k = 5$ in the simulation), where each step consists of a horizontal transport phase, followed by a chemistry/vertical transport phase, followed by another horizontal transport phase. Each horizontal transport phase performs $l \times s$ backsolves, one for each layer and species. All may be computed independently. However, for each layer l, all backsolves use the same factored matrix A_l. The chemistry/vertical transport phase performs an independent computation for each of the p grid points. Output for the hour is an updated concentration array C', which is the input to the next hour.

In the implementation, the array is distributed across P processors by layer: processor 0 owns the first $\frac{l}{P}$ layers, processor 1 owns the next $\frac{l}{P}$ layers, and so on. In the first stage, the preprocessing and horizontal transport operates on the *layer* dimension, so the computation is local and no communication is involved. In the second stage, however, the chemistry/vertical transport operates on the *grid* dimension, and so a transpose on the concentration array C is performed to distribute the data across the processors by grid point: processor 0 owns the first $\frac{p}{P}$ grid points, processor 1 owns the next $\frac{p}{P}$ grid points, and so on. Such transpose requires that each processor sends a message of size $O(\frac{p \times s \times l}{P^2})$ to every other processors. Once the chemistry/vertical transport computation is finished, a reversed transpose is performed in a similar fashion – each processor sends a message of size $O(\frac{p \times s \times l}{P^2})$ to each of the other processors. This is followed by another horizontal transport phase. In summary, each step is characterized by a period of computation phase of duration t_i (preprocessing), followed by k back-to-back pairs of *all-to-all* traffic attributed to the distribution transpose, interleaved with horizontal transport (of duration t_h) and vertical/chemical transport computation (of duration t_v).

4 Communication mechanisms

All of our test applications use the PVM system for communication. PVM [21, 12] is a message-passing and utility package which provides a presentation layer interface which has the syntax and semantics of message passing interfaces on distributed memory parallel supercomputers. In addition to message passing, PVM also provides mechanisms for managing a dynamic, heterogeneous pool of machines as a single "parallel virtual machine." This support is implemented in a user-level daemon process which is run on each machine. The daemons talk to each other via UDP in order to maintain information about the global state of the virtual machine, as well as to handle user requests such as sending signals to remote user processes. Each machine may run multiple user processes. A user process can communicate with another user process on the same machine or on a different machine using the same interface. Intramachine communication is done via a local IPC mechanism. Intermachine communication can be done in two distinct (user selectable) ways. By default, the message is copied via IPC to the daemon, which sends it to the daemon on the destination machine via a protocol built on top of UDP. The receiving daemon then delivers the message to the destination process via IPC. This mechanism has the advantage of better scalability, but tends to be somewhat slow. In the alternative mechanism, the messages are sent directly from the sender process to receiver process via TCP. All of the Fx kernels and AIRSHED use this mechanism.

PVM messages can contain arbitrary data collected from arbitrary memory locations. Data is “packed” into a message using a variety of API calls. However, the data is not necessarily appended into a contiguous memory buffer. Instead, it is stored as a list of fragments which are sent independently. This distinction is important to understand the behavior of one of the Fx kernels, T2DFFT. All the other kernels (and AIRSHED) assemble their messages in a copy loop *before* using PVM. The result is that each message is sent as a single, large fragment by PVM. The copy loop is an artifact of other (older) Fx implementations for message passing systems which only support sending contiguous buffers. T2DFFT, however, tries to avoid the intermediate copy step by performing multiple packs per message. The result is that each message is passed to the socket layer as a series of fragments.

5 Methodology

Our approach is to directly measure the network traffic of each of the programs on a LAN of Ethernet [17] connected DEC Alpha [6] workstations. A machine running in promiscuous mode is used to record each packet. This data is then analyzed using a variety of simple, custom programs.

5.1 Environment

Nine DEC 3000/400 Alpha (21064 [5] at 133 MHz with 64 MB RAM) workstations [6] running OSF/1 2.0 were used as our testbed. The built-in Ethernet [17] adaptors were married to a multi-segment bridged Ethernet LAN, so all machines shared a common collision domain and an aggregate 1.25 MB/s of bandwidth. Since these machines are office workstations and other machines share the LAN, all measurements were performed in the early morning hours (4-5 am) to avoid other traffic, and were repeated several times.

5.2 Compilation

Each of the six Fx programs can be compiled for an arbitrary number of processors. Due to the stress these programs place on machines and networks, it was decided to compile them for four processors. The programs were compiled with version 2.2 of Fx compiler and version 3.3 of the DEC Fortran compiler. The basic level of optimization (-O) was used with the latter compiler. The object files were linked with version 3.3.3 of PVM and with version 2.2 of the Fx/PVM run-time system.

5.3 Measurement

To measure the network traffic, one of the workstations was configured with the DEC packet filter software, which allows priveledged users to use the network adaptor in promiscuous mode. The measurement workstation was not used to run any Fx program. Instead, it ran the TCPDUMP program included with OSF/1 and collected a trace of all the packets on the LAN generated by each test program. For the Fx programs, including AIRSHED, each outer loop as iterated 100 times, except for SEQ, which was iterated five times.

Each of our traces captured all the packets on the network, providing a time stamp, size, protocol, source and destination for each packet. We considered the size of the packet to include the data portion, TCP or UDP header, IP header, and Ethernet header and trailer. Where sensible, we produced a trace for a single connection by extracting all packets sent from one host to another.

6 Results

In this section, we describe the traffic characteristics for each of the six Fx programs.

6.1 Fx kernels

For each of the kernels, we examined its aggregate traffic and the traffic of a representative connection, if there was one. We define a connection to be a kernel-specific simplex channel between a source machine in a destination machine. Thus for $P = 4$, each of the kernels exhibits 12 connections. Notice that by considering a connection between *machines* as opposed to between machine-port pairs, we capture all kernel-specific traffic between a source and destination machine. This includes TCP traffic for message passing, UDP traffic between the PVM daemons, and TCP ACKs for the symmetric channel. The communication pattern of HIST and SEQ are not symmetric, so we only examine the aggregate traffic of these kernels. T2DFFT's pattern is symmetric about the partition, so we consider a connection from a machine in the sending half to a machine in the receiving half. The other kernels have symmetric communication patterns, so we choose the connection between an two arbitrary machines.

The traffic of each of the kernels is characterized by its packet sizes, interarrival times for packets, and bandwidth, both for the aggregate traffic and the traffic over the representative connection. We concentrate on characterizing the bandwidth, since this appears the most interesting.

We note here that the graphs presented are not all to the same scale. The intention is to better highlight the features of each graph. However, this does make quick comparisons between graphs more difficult.

Packet size statistics

Figure 3 shows the minimum, maximum, average and standard deviation of packet sizes for each of the five applications. The first table covers all the connections while the second includes only packets in a single representative connection. Although we do not present histograms here, it is important to remark that for several of the kernels (2DFFT, HIST, SOR), the distribution of packet sizes is *trimodal*. This is because these programs send messages large messages which are split over several maximal size packets and a single smaller packet for the remainder. Further, because TCP is used for the data transfer, there are a significant number of ACK packets. One would expect T2DFFT to also send large messages and therefore exhibit a trimodal distribution of packet sizes. However, a different PVM mechanism is used to assemble messages in T2DFFT. As described in section 4, PVM internally stores messages as a fragment list and generates packets for each fragment separately. Because of the way messages are assembled in T2DFFT, many fragments result, explaining the variety of packet sizes.

Program	Packet Size (Bytes)			
	Min	Max	Avg	SD
SOR	58	1518	473	568
2DFFT	58	1518	969	678
T2DFFT	58	1518	912	663
SEQ	58	90	75	14
HIST	58	1518	499	575

(aggregate)

Program	Packet Size (Bytes)			
	Min	Max	Avg	SD
SOR	58	1518	577	591
2DFFT	58	1518	977	667
T2DFFT	134	1518	1442	158
SEQ	-	-	-	-
HIST	-	-	-	-

(connection)

Figure 3: Packet size statistics for Fx kernels

Program	Interarrival Time (ms)			
	Min	Max	Avg	SD
SOR	0.0	1728.7	82.1	234.9
2DFFT	0.0	1395.8	1.3	10.8
T2DFFT	0.0	1301.6	1.5	14.3
SEQ	0.0	218.6	1.3	8.6
HIST	0.0	449.9	16.5	45.5

(aggregate)

Program	Interarrival Time (ms)			
	Min	Max	Avg	SD
SOR	0.0	1797.0	614.2	590.8
2DFFT	0.0	2732.6	15.1	120.5
T2DFFT	0.0	4216.7	9.5	127.3
SEQ	-	-	-	-
HIST	-	-	-	-

(connection)

Figure 4: Packet interarrival time statistics for Fx kernels

Interarrival time statistics

Figure 4 shows the minimum, maximum, average, and standard deviation of the packet interarrival times for each of the five programs. The first table shows the statistics for all the connections, while the second concentrates on a single representative connection. Notice that ratio of maximum to average interarrival time for each program is quite high. This is due to the aggregate bursty nature of the traffic, as we discuss below.

Bandwidth

Figure 5 shows the aggregate and per-connection average bandwidth used over the lifetime of each of the five programs. It is somewhat counter-intuitive (and quite promising!) that even the most communication intensive Fx programs such as 2DFFT do not consume all the available bandwidth. However, recall that Fx programs synchronize via their global communication phases, so there are stretches of time where every processor is computing. Each of these periods is followed by an intense burst of traffic, as every processor tries to communicate.

It is important to note that this synchronization is inherent in the Fx model and is not merely a result of serialization due to the Ethernet MAC protocol. In fact, in several new communication strategies optimized for compiler-generated SPMD programs the global synchronization is *enforced* by a separate barrier synchronization before each communication phase [18, 20].

The effect of this inherent synchronization is made clear by examining figure 6, which plots the instantaneous bandwidth averaged over a 10 ms window for the each of the kernel. This was

Program	KB/s
SOR	5.6
2DFFT	754.8
T2DFFT	607.1
SEQ	58.3
HIST	29.6

(aggregate)

Program	KB/s
SOR	0.9
2DFFT	63.2
T2DFFT	148.6
SEQ	-
HIST	-

(connection)

Figure 5: Average bandwidth for Fx kernels

computed using a sliding 10 ms averaging window which moves a single packet at a time. For the SOR, 2DFFT, and T2DFFT kernels, the aggregate bandwidth is plotted on the left and the bandwidth of the representative connection on the right. Since HIST and SEQ have no representative connection, only their aggregate bandwidths are plotted. In each case, we show a ten second span of time, enough to include several iterations of the kernel. The complete traces are between 50 and several hundred seconds long.

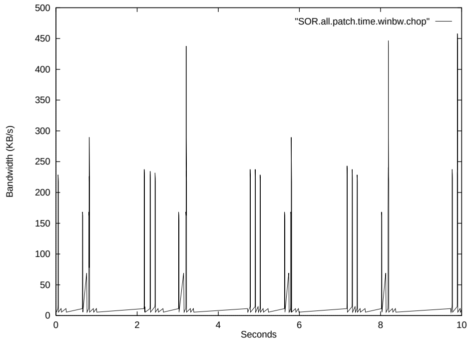
The most remarkable attribute of each of the kernels is that the bandwidth demand is highly periodic. Consider the 2DFFT. Both plots show about five iterations of the kernel. Notice that there are substantial portions of time where virtually no bandwidth is used (all the processors are in a compute phase). The reason the third and fourth burst are short is because they are, in fact, a single communication phase where some processor descheduled the program. Because the all-to-all communication schedule is fixed and synchronous, the communication phase stalled until that processor was able to send again.

Figure 7 shows the corresponding power spectra (periodograms) of the instantaneous average bandwidth. The power spectra show the frequency-domain behavior of the bandwidth, and are very useful for characterizing it, as we will explore in Section 7.2. It is important to note that the power spectra capture the periodicity of the bandwidth demands these applications place on the network.

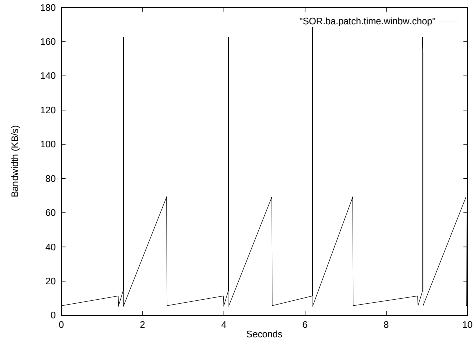
For these calculations, the entire trace of each kernel was used, not just the first 10 seconds displayed in figure 6. Because a power spectrum computation requires evenly spaced input data, the input bandwidth was a computed along static 10 ms intervals by including all packets that arrived during the interval. This is a close approximation to the sliding window bandwidth, and more feasible than correctly sampling the sliding window bandwidth data, which would require a curve fit over a massive amount of data.

Not surprisingly, SEQ, in which processor 0 repeatedly broadcasts a single word, is extremely periodic, with the four Hz harmonic being the most important. HIST has a 5 Hz fundamental with linearly declining harmonics at 10, 15, etc. Hz.

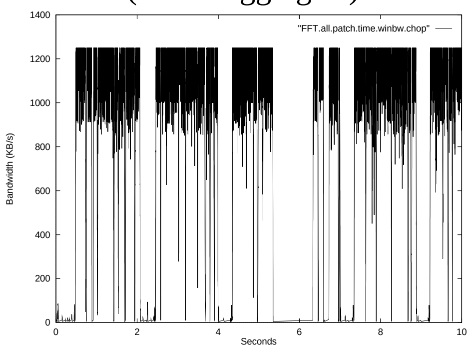
SOR and 2DFFT display opposite relationships between the connection and aggregate power spectra. For SOR, the connection power spectrum shows great periodicity, with a fundamental of about 5 Hz and interestingly modulated harmonics, but the aggregate power spectrum shows far less clear periodicity. For 2DFFT, the relationship is the reverse, although less strong, with a clear fundamental of $1/2$ Hz and exponentially declining harmonics. There are two explanations for this. First, 2DFFT transfers more data per message than SOR ($O(\left(\frac{N}{P}\right)^2)$ versus $O(N)$, $N = 512$,



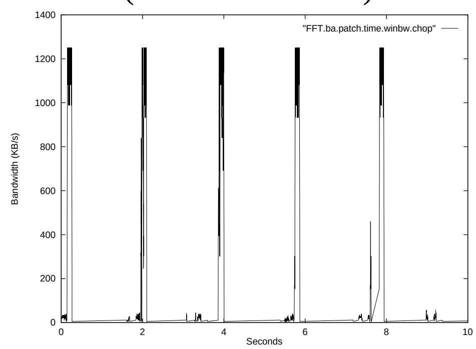
(SOR - aggregate)



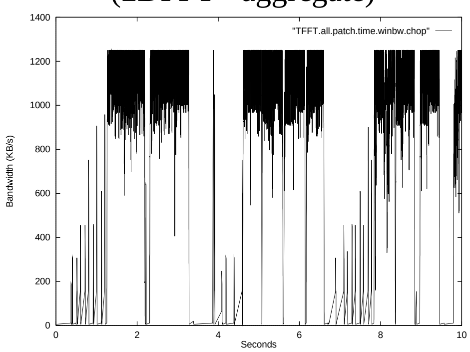
(SOR - connection)



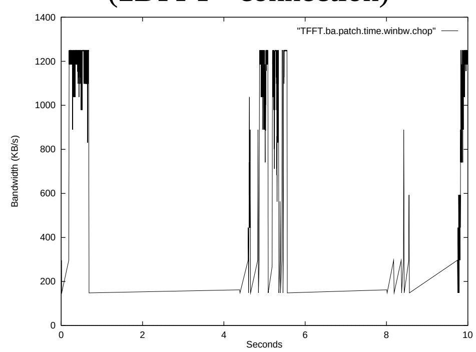
(2DFFT - aggregate)



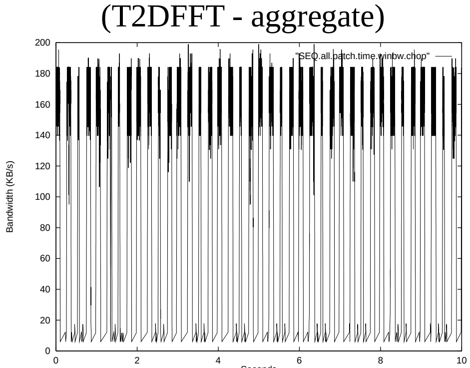
(2DFFT - connection)



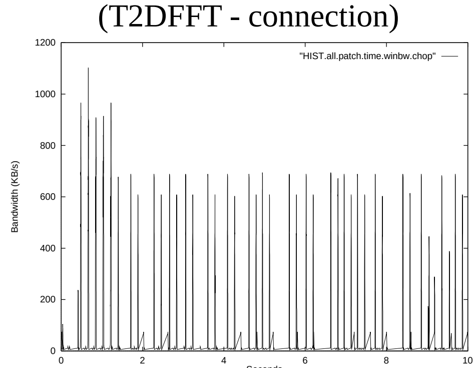
(T2DFFT - aggregate)



(T2DFFT - connection)

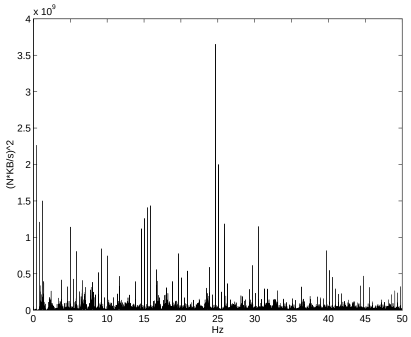


(SEQ - aggregate)

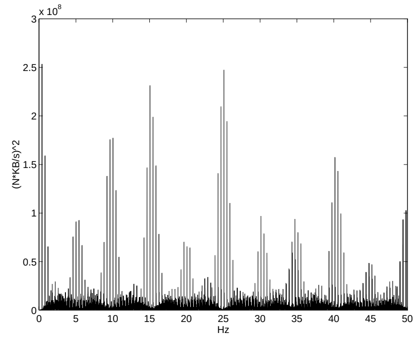


(HIST - aggregate)

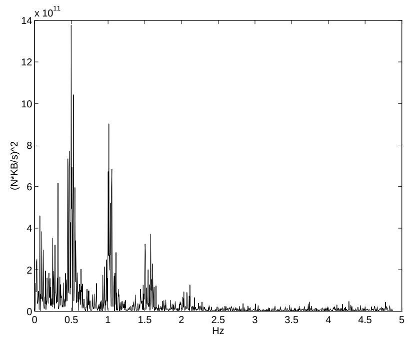
Figure 6: Instantaneous bandwidth of Fx kernels (10ms averaging interval)



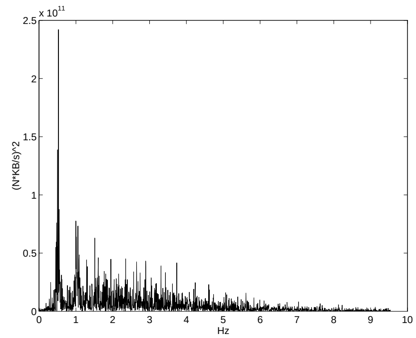
(SOR - aggregate)



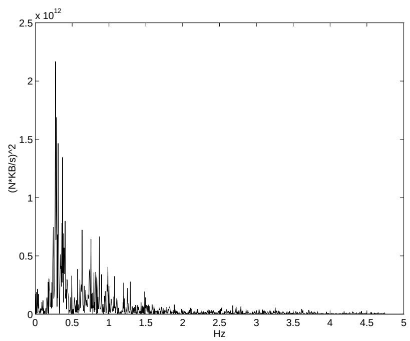
(SOR - connection)



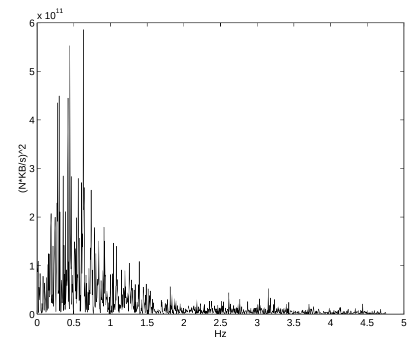
(2DFFT - aggregate)



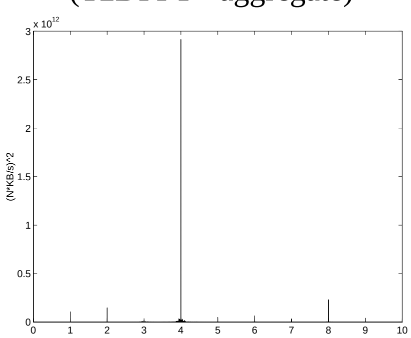
(2DFFT - connection)



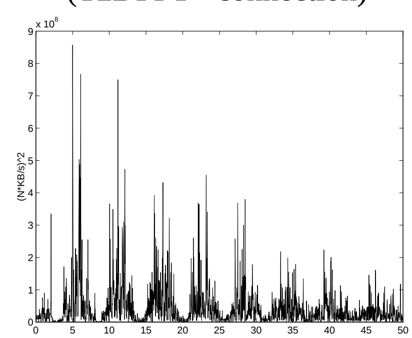
(T2DFFT - aggregate)



(T2DFFT - connection)



(SEQ - aggregate)



(HIST - aggregate)

Figure 7: Power spectrum of bandwidth of Fx kernels (10ms averaging interval)

Program	Packet Size (Bytes)			
	Min	Max	Avg	SD
AIRSHED	58	1518	899	693

(aggregate)

Program	Packet Size (Bytes)			
	Min	Max	Avg	SD
AIRSHED	58	1518	889	688

(connection)

Figure 8: Packet size statistics for AIRSHED

$P = 4$), so has a better chance of being descheduled (as discussed above). Second, 2DFFT's communication pattern more closely synchronizes all the processors than SOR's. Thus a single SOR connection has a better chance of being periodic because the sending processor is less likely to be descheduled. On the other hand, SOR's aggregate traffic will be less periodic because the processors are less tightly synchronized. Notice, however, that in both cases, the representative connection's power spectrum *does* show considerable periodicity.

T2DFFT's power spectra have the least clear periodicity of all the Fx kernels. However, the aggregate spectrum seems slightly cleaner than the spectrum of the representative connection. The fact that neither spectrum is very clean is surprising given the synchronizing nature of this pattern, the balanced message sizes, and the communication schedule (shift) used for it. We believe the problem arises from PVM's handling of the message as a cluster of fragments.

6.2 AIRSHED Simulation

For AIRSHED, we examined both the aggregate traffic as well as the traffic of one connection. The format of the data we present mirrors that of the previous section.

Packet size statistics

Figure 8 shows the minimum, maximum, average, and standard deviation of packet sizes for the AIRSHED application (for all connections and for the representative connection). We observe that the packet size distribution for the single connection is very similar to the aggregate packet distribution, which supports the argument that the traffic from the single connection is representative of the aggregate traffic.

Interarrival time statistics

Figure 9 shows the minimum, maximum, average, and the standard deviation of packet interarrival times. Note that both the maximum and average interarrival times are of an order of magnitude greater than that of the kernel applications. As in the case of the kernel applications, the ratio of maximum to average interarrival time is quite high, which is characteristic of bursty traffic.

Bandwidth

The average aggregate and per-connection bandwidths for the AIRSHED application are 32.7 KB/s and 2.7 KB/s, respectively. Figure 10 shows the instantaneous bandwidth averaged over a 10 ms window (over a 500 sec interval, and a 60 sec interval). It is clear that the bandwidth demand is

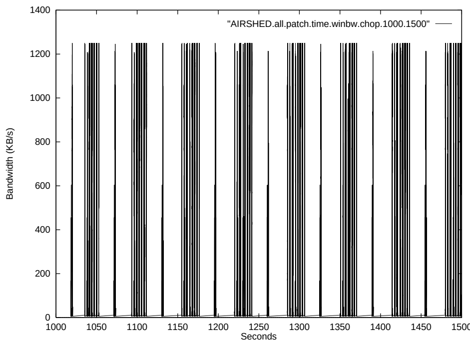
Program	Interarrival Time (ms)			
	Min	Max	Avg	SD
AIRSHED	0.0	23448.6	26.8	513.3

(aggregate)

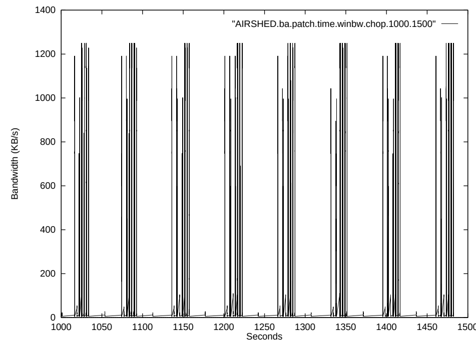
Program	Interarrival Time (ms)			
	Min	Max	Avg	SD
AIRSHED	0.0	37018.5	317.4	2353.6

(connection)

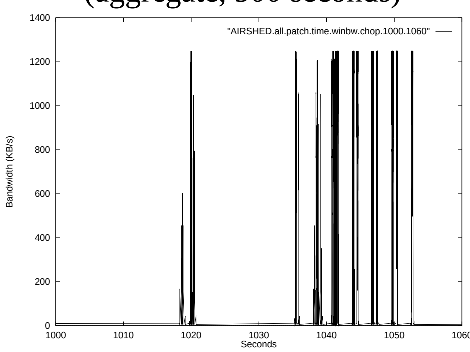
Figure 9: Packet interarrival time statistics for AIRSHED



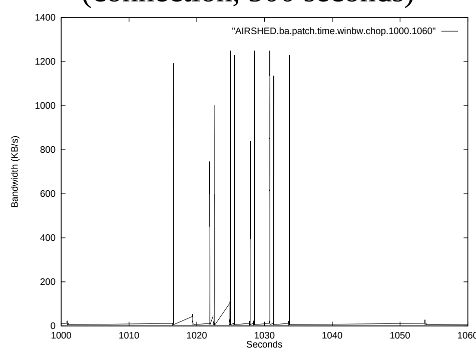
(aggregate, 500 seconds)



(connection, 500 seconds)



(aggregate, 60 seconds)

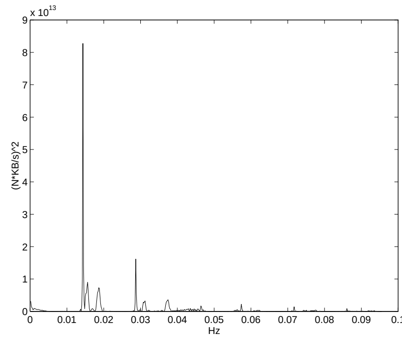


(connection, 60 seconds)

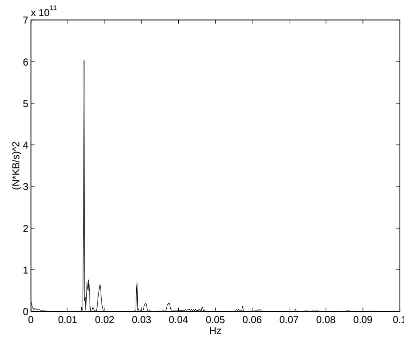
Figure 10: Instantaneous bandwidth of AIRSHED (10ms averaging interval)

highly periodic, and is periodic over *three* different time scales. The simulation is divided into a sequence of h simulation-hours ($h = 100$ in the simulation), each of which involves a sequence of k simulations steps ($k = 5$). Each simulation hour starts with a preprocessing stage, where the stiffness matrix is computed. Once the stiffness matrix is computed, the program moves on to the simulation stages. Such simulation is characterized by (1) a local horizontal transport computation phase, (2) a subsequent global *all-to-all* transpose traffic, (3) a local chemical/vertical transport computation phase, and finally (4) a global *all-to-all* transpose traffic in the reversed direction.

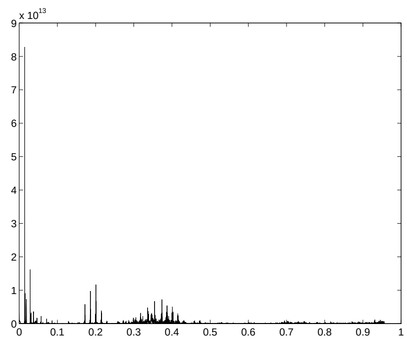
A total of 100 bursty periods are observed, corresponding to the 100 simulation hours. The bandwidth utilization between each bursty period is very low because no communication is involved during the preprocessing stage at the beginning of each simulation hour. Each bursty period can be further divided into 5 pairs of peaks, with each pair of peaks corresponding to one simulation step. The time between each pair of peaks reflects the time spent in the chemical/vertical transport computation stage, whereas the time interval between adjacent pairs – which is slightly



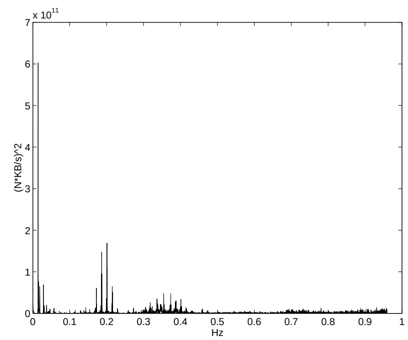
(aggregate, 0 – 0.1 Hz)



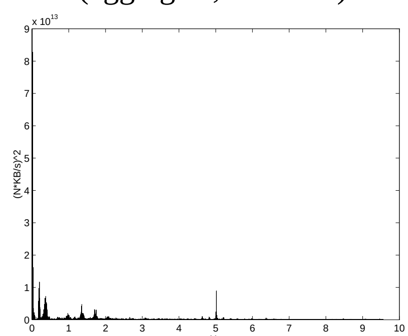
(connection, 0 – 0.1 Hz)



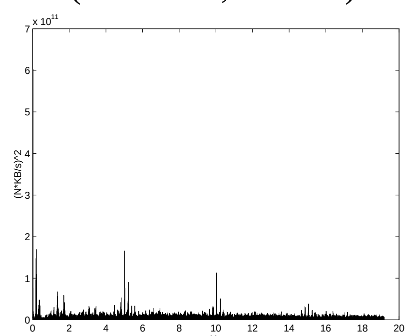
(aggregate, 0 – 1 Hz)



(connection, 0 – 1 Hz)



(aggregate, 0 – 20 Hz)



(connection, 0 – 20 Hz)

Figure 11: Power spectrum of bandwidth of AIRSHED (10ms averaging interval)

shorter – corresponds to the time spent in the horizontal transport computation. Such periodicity becomes very clear when we observe the power spectra for the AIRSHED simulation (figure 11). There are three peaks (plus their harmonics) in the power spectrum at approximately 0.015 Hz (66 sec, corresponding to a simulation hour), 0.2 Hz (5 sec, corresponding to the length of the chemical/vertical transport phase), and 5 Hz (200 ms, corresponding to that of the horizontal transport phase), respectively. Section 7.2 discusses the use of power spectra for characterizing the network traffic of these programs.

7 Discussion

The measurement and analysis of the Fx kernels and the AIRSHED program point to several important characteristics of the network traffic of Fx parallel programs. The most important of these is that their periodicity is well characterized by their power spectra, and can be emulated by simplifying the Fourier series implied by the spectra. Finally, we suggest a negotiation model for QoS which would allow both the network and the program to co-optimize for performance.

7.1 Elementary characteristics

Fx programs exhibit some global, collective communication patterns which may not necessarily be characterized by the behavior of single connection. For example, the SEQ (broadcast pattern) and HIST (tree pattern) kernels are not symmetric — in SEQ only the connection from processor 0 to the every other processor (and the symmetric connections back to processor 0) see traffic. Further, characterizing the symmetric patterns such as neighbor, all-to-all, and partition by a single connection ignores the fact that these patterns are very different in the number of connections that are used. For example, each of the patterns may communicate the same size message along a connection, but while all-to-all sends such a message along *all* $P(P-1)$ connections, neighbor sends a message along only at most $2P$ connections. The partition pattern is in the middle at $\frac{P^2}{4}$ connections for an equal partition into two halves.

Another important characteristic of Fx programs is that their communication phases are synchronized, either explicitly or implicitly. This means that the traffic along the active connection is *correlated* and any traffic model must capture this. Further, the stronger the synchronization, the more likely it is that the connections are *in phase.*

7.2 Characterizing periodicity

As stated above, the synchronized communication phases of a Fx program imply that its connections act in phase. Thus, the power spectra of Figures 7 and 11 fully characterize the bandwidth demands of the applications discussed in this paper. Furthermore, it should be realized that the power spectrum is the square of the Fourier transform of the time-domain instantaneous average bandwidth. Since this underlying signal is periodic, the transform is a Fourier series:

$$X(\omega) = \sum_{k=-\infty}^{\infty} 2\pi a_k \delta(\omega - \omega_0) \quad (1)$$

where the a_k are the coefficients which can be read off of the power spectrum graphs. The time-domain instantaneous bandwidth can then be reconstructed as:

$$x(t) = \sum_{k=-\infty}^{\infty} a_k e^{jk\omega_0 t} \quad (2)$$

While the summation may appear analytically daunting, note that $x(t)$ can be approximated by choosing some number of the “spike” a_ks from the spectra (those with the greatest magnitude). As the number of spikes chosen increases, the approximation will converge to the actual signal.

7.3 QoS negotiation model

Consider a simple parallel program where each processor generates periodic bursts along one of its connections (a shift pattern.) Unlike variable bit rate video source, where the periodicity is known, but the burst size is variable, the parallel program's burst size is usually known a priori (in the case of Fx, at compile-time), but the period between bursts depends on the number of processors and the bandwidth the network can provide to the application during the burst. Suppose the program performs W work during a compute phase, and each processor send a message of length N. If the network can allocate a burst bandwidth of B for each active connection without congestion, then the burst length is $t_b = \frac{N}{B}$ and the burst interval is $t_{bi} = \frac{W}{P} + \frac{N}{B}$. Notice that the burst interval, which certainly plays into the decision of what B the network can commit to, is a function of B itself (as well as of the other commitments the network has made.)

It must pointed out that the parallel program clearly wants to minimize t_{bi} in order to minimize its total execution time. One way it can do this is to increase the number of processors P it runs on. However, there is a natural tension with the bandwidth B that the network can commit to, and, less obviously, the communication pattern determines how strong that tension is. Thus getting the best performance from a parallel program on a network is essentially an optimization problem, *where the number of processors plays a role.* We suggest that a SPMD parallel program should characterize its traffic with three parameters, $[l(), b(), c]$, where c is the communication pattern, l is a function from the number of processors P to the local computation time t_{local} on each processor, and b is a function from P to the burst size N, along each connection. In order to meet the "guarantee" of minimizing t_{bi}, the network is allowed to return the number of processors P the program should run on.

8 Conclusions and Future Work

We measured the traffic characteristics of six parallel programs on an Ethernet. The conclusion to be drawn from the measurements is that the traffic of parallel programs is fundamentally different from the media traffic that is the current focus of QoS research. Unlike media traffic, there is no intrinsic periodicity due to a frame rate. Instead, the periodicity is determined by application parameters and the network itself. We suggested a traffic characterization and service negotiation model that allows the network to modulate application parameters in an effort to achieve the best performance possible given the current network state. This is clearly an important area for future research.

Bibliography

[1] AGRAWAL, G., SUSSMAN, A., AND SALZ, J. An integrated runtime and compile-time approach for parallelizing structured and block structured applications. *IEEE Transaction on Parallel and Distributed Systems 6*, 7 (July 1995), 747–754.

[2] BIAGIONI, E., COOPER, E., AND SANSOM, R. Designing a practical ATM LAN. *IEEE Network* (March 1993), 32–39.

[3] DEPRYCHER, M., PESCHI, R., AND LANDEGEM, T. V. B-ISDN and the OSI protocol reference model. *IEEE Network* (March 1993), 10–18.

[4] DINDA, P. A., AND O'HALLARON, D. R. Fast message assembly using compact address relations. In *Proc. of SIGMETRICS '96* (Philadelphia, 1996), ACM, pp. 47–56.

[5] DOBBERPUHL, D. I. A. A 200-MHz 64-bit dual-issue CMOS microprocessor. *Digital Technical Journal 4*, 4 (1992), 35–50. ftp://ftp.digital.com/pub/Digital/info/DTJ/axp-cmos.txt.

[6] DUTTON, T. A., EIREF, D., KURTH, H. R., REISERT, J. J., AND STEWART, R. L. The design of the DEC 3000 AXP systems, two high performance workstations. *Digital Technical Journal 4*, 4 (1992), 66–81. ftp://ftp.digital.com/pub/Digital/info/DTJ/axp-dec-3000.txt.

[7] FERNANDEZ, J. R., AND MUTKA, M. W. Model and call admission control for distributed applications with correlated bursty traffic. In *Proceedings of Supercomputing '95* (San Diego, December 1995).

[8] FERRARI, D. Client requirements for real-time communication services. *IEEE Communications Magazine 11*, 11 (November 1990), 65–72.

[9] FERRARI, D., BANERJEA, A., AND ZHANG, H. Network support for multimedia - a discussion of the tenet approach. *Computer Networks and ISDN Systems 26*, 10 (July 1994), 1167–1180.

[10] FORUM, H. P. F. High Performance Fortran language specification version 1.0 draft, Jan. 1993.

[11] GARRETT, M., AND WILLINGER, W. Analysis, modeling and genreation of self-similar VBR video traffic. In *Proceedings of SIGCOMM '94* (London, September 1994).

[12] GEIST, A., BEGUELIN, A., DONGARRA, J., JIANG, W., MANCHECK, R., AND SUNDERAM, V. *PVM: Parallel Virtual Machine*. MIT Press, Cambridge, Massachusetts, 1994.

[13] GROSS, T., O'HALLARON, D., AND SUBHLOK, J. Task parallelism in a High Performance Fortran framework. *IEEE Parallel & Distributed Technology 2*, 3 (1994), 16–26.

[14] KUMAR, N., RUSSEL, A., SEGALL, E., AND STEENKISTE, P. Parallel and distributed application of an urban regional multiscale model. Carnegie Mellon Dept. of Mech. Eng. and Dept. of Computer Science, 1995. working paper.

[15] MCRAE, G., GOODIN, W., AND SEINFELD, J. Development of a second-generation mathematical model for urban air pollution – 1. model formulation. *Atmospheric Environment 16*, 4 (1982).

[16] MCRAE, G., RUSSELL, A., , AND HARLEY, R. *CIT Photochemical Airshed Model – System Manual*. Carnegie Mellon University, Pittsburgh, PA, and California Institute of Technology, Pasadena, CA, Febrary 1992.

[17] METCALFE, R. M., AND BOGGS, D. R. Ethernet: Distributed packet switching for local computer networks. *Communications of the ACM 19*, 7 (July 1976), 365–404.

[18] OSBORNE, R. A hybrid deposit model for low overhead communication in high speed lans. In *Proceedings of 4th IFIP International Workshop on Protocols for High Speed Networks* (August 1994), G. Neufeld and M. Ito, Eds., ???

[19] STICHNOTH, J., O'HALLARON, D., AND GROSS, T. Generating communication for array statements: Design, implementation, and evaluation. *Journal of Parallel and Distributed Computing 21*, 1 (Apr. 1994), 150–159.

[20] STRICKER, T. M. *A Communication Infrastructure for Parallel and Dsitributed Programs*. PhD thesis, Carnegie Mellon University School of Computer Science, November 1996. To Appear.

[21] SUNDERAM, V. S. Pvm : A framework for parallel distributed computing. *Concurrency: Practice and Experience 2*, 4 (December 1990), 315–339.

[22] WALKER, D. The design of a standard message passing interface for distributed memory concurrent computers. Tech. Rep. TR-12512, ONRL, October 1993. To appear in Parallel Computing, 1994.