

# Real-Time Communications Scheduling for Massively Parallel Processors (Position Paper) \*

Richard Games      Arkady Kanevsky      Peter Krupp      Leonard Monk

The Mitre Corp.  
202 Burlington Rd.  
Bedford, MA 01730-1420  
email:{rg,arkady,pck}@mitre.org

## 1 Introduction

Can general purpose commercial massively parallel processors (MPPs) be used for computationally intensive real-time applications that have traditionally required a custom arrangement of special-purpose computers and mainframes? If so, then the enormous life-cycle costs of many systems needed by, for instance, the Government could potentially be reduced. The components would be commercially available and continuing technological advances could more easily be incorporated into existing systems.

Relevant applications have requirements not found in large-scale scientific computing, which has up to now provided most of the motivation for the development of MPPs. Perhaps the most important difference is the need for real-time processing. Depending on the application, multi-level security, fault tolerance and other features may also be necessary.

There have already been hardware advances that may make such a high performance computing solution possible. Also, the packaging problems associated with embedded applications are currently being addressed by a variety of ARPA and industry research and development programs. However, daunting software challenges remain. In this paper we focus on one critical problem, the real-time scheduling of the

communications between processing nodes.

We are initially concentrating on sensor processing applications, which involve a mix of processing types and exhibit a wide range of real-time requirements. What is called “front-end” signal processing is performed first. Front-end signal processing must maintain high throughput while accomplishing a variety of functions, such as the removal of interference from digital data extracted from a sensor. Subsequent “object processing” involves time-critical functions such as automatic target recognition and tracking, with strict latency requirements. A final “mission processing” phase, involving, for instance, the allocation of resources that may affect the entire processing chain, must be responsive to the commands of a human decision maker. If necessary, combining the results of multiple sensors or multiple mode operation with a single sensor increases the complexity of the real-time requirements.

The overall goal of MITRE’s work in this area is to determine to what degree MPPs can meet this range of processing requirements. To this end researchers at MITRE have been assessing the performance of current MPPs on applications of interest, identifying potential problem areas relative to real-time processing, and developing solutions that provide the desired predictability without undue sacrifice of performance.

To understand the role and importance of the specific topic of this paper, consider some of the necessary ingredients for the successful use of MPPs for the targeted applications. Individual processing nodes must first be able to sustain high average processing rates for the computational kernels of the relevant numerical algorithms. Then allowance must be made for alternation of tasks at nodes, and strict real-time deadlines must be guaranteed. These added constraints moti-

---

\*This work was supported in part by: the U.S. Air Force Electronic Systems Center and performed under MITRE MOIE Project 74110 of contract F19628-94-C-0001, managed by Rome Laboratory/C3CB; the Advanced Research Projects Agency and performed under MITRE Project 820W of contract DAAB07-94-C-H601, managed by the Naval Command, Control and Ocean Surveillance Center, RDT&E Division (NRaD); and a grant of HPC time from the DoD HPC Major Shared Resource Center, Wright Patterson Air Force Base. We wish to thank the Honeywell Corporation, Space Systems, Clearwater Florida for use of their Paragon and the referees for their comments.

vate the application of real-time process scheduling technology ([13]). The underlying raw capability of the hardware to transmit messages from point to point is analogous to the advertized processing rate for the CPU. It must first be shown that the theoretical rates can practically be approached with available operating systems and communications software, realistic message lengths, etc. Then the possibility that different message streams might interfere with each other must be taken into account, and real-time deadlines must obviously be met. Our goal here is to address these last two difficulties (see [5] for more details).

In addition to summarizing MITRE's related and supporting benchmarking work through the summer of 1994, the next section gives some relevant background on machine architecture and motivating applications. Section 3 discusses special considerations for this scheduling problem and compares general approaches. Section 4 gives more detail on an analytical framework called Packet Stream Analysis, and the final section summarizes the paper.

## 2 Real-Time MPP Application Benchmarks

A distributed memory MPP consists of a set of processing nodes, each with its own local memory, interconnected by a communications network. We have focused on the case where processing nodes communicate through explicit message passing. There are a variety of computation models (e.g., SIMD, MIMD); see [7] for background.

As an example, consider the Intel Paragon MPP, each of whose processing nodes currently employs two i860XP RISC microprocessors (one for processing and one for communication) and is very roughly equivalent in complexity and power to a modest standalone workstation. The i860XP has a peak processing rate of 100 Million Floating point Operations Per Second (MFLOPS) (single precision). These nodes are arranged in a rectangular array. The data communications network is a rectangular backplane mesh, with one special-purpose routing chip for each processing node. Each router has connecting links to its corresponding processing node and to each neighboring router. Counting one-way directional links, which have a capacity of 200 million bytes per second, the routers for interior nodes each have ten connecting links. A detailed picture would also show other devices at the nodes and between the node buses and the backplane.

Messages from one node to another are transferred entirely on the backplane, with no intermediate storage at other nodes and very little buffering on the

backplane. A fixed deadlock-free "wormhole" routing algorithm is used. It sets switches in the routers as the header of a message is processed and fixes settings until the entire message has passed. Header contention at the router is resolved in a round-robin manner. A separate communications network is used for certain system purposes and makes possible efficient clock synchronization. Any node can be configured for I/O with a variety of interfaces available. This connection to the outside does not use the backplane, but the operating system will generate traffic on the backplane when an I/O system call is made by a non-I/O node. For more details on the Paragon see [4].

The Paragon processing nodes can run the OSF/1 operating system along with the Intel NX message passing primitives. In this case, user-designated messages are broken into pieces called packets for actual transmission. Packet size is fixed per program and is at most 1792 bytes. It is these packets which appear as messages to the hardware (as in the above description), and we will maintain the useful conceptual distinction between packets and the possibly much larger user level units to be called messages. Another option is the Sandia-University of New Mexico operating system (SUNMOS) [15, 8], which has been streamlined for high performance. In this case, no breaking up of messages is performed, and packets are just messages with headers. There are also a variety of optimized library calls available for common processing and communication kernels. The ARPA-Honeywell Embedded Touchstone [2] program is producing an embeddable version of the Paragon that will run the same system and application software. Real-time system software for the Embedded Touchstone is currently under development by Honeywell and the Open Software Foundation-Research Institute (OSF-RI).

In sensor processing applications, a sequence of problem instances are presented with a fixed period, and the result for a given instance must be computed within a fixed amount of time, the latency. The problem latency may be less than, equal to, or greater than the period. Front-end signal processing algorithms are dominated by data-independent numerical manipulation, and it is customary to express their periodic requirements in terms of FLOPS.

An example is the Lincoln Laboratory Advanced Detection Technology Sensor (ADTS), a synthetic aperture radar (SAR) whose real-time requirements are specified (as a benchmark for a rapid prototyping ARPA program) in [16]. Pulses are received at a maximum rate of 556 per second. A variety of processing functions are applied in sequence to form the SAR

image, yielding a total processing rate requirement of 1085 MFLOPS. The latency requirement is stated in terms of a processing frame consisting of 512 consecutive pulses. The time between the completion of the input of a frame to the SAR processor and the production of the corresponding image must not exceed three seconds.

Before a real-time application can be mapped to a parallel processor, benchmarks must be conducted to assess the capabilities of the MPP on the essential processing and communication kernels. A rough knowledge of the demonstrable capabilities of targeted machines in conjunction with available support software is also necessary to point our communications scheduling work in a reasonable direction. A generic benchmarking methodology designed to assess a system's real-time performance was introduced by Weideman and Kamenoff in 1992, often called the Hartstone benchmarks, see [14]. MITRE applied various benchmarks to the Intel Paragon in anticipation of the availability of the Embedded Touchstone, including adaptations of the Hartstone benchmarks to our context (see [3] for details). For our purposes it is reasonable to assume that the entire machine is devoted to the application system while it is running. A source node injected blocks of data at a pre-specified rate into the processing chain being tested. A sink node collected the results for off-line verification of correct functionality. The benchmarking metric was the minimum period that could be sustained for a selected processing and/or communication kernel for a pre-specified period of time (usually from a few seconds to 30 seconds).

The functions involved in SAR signal processing were tested on the Paragon. In the important case of the fast Fourier transform (FFT), the existing library routines were most efficient at transform lengths commensurate with the size of on-chip cache. It was shown that processing efficiencies could be maintained for pipeline processing in which double buffering was used to hide most of the overhead of interprocessor communication. For example, a two-dimensional FFT processing pipeline consisting of a source, an FFT node, a corner-turn node that implements a matrix transpose, a second FFT node, and a sink was configured. For a processing load consisting of 512 instances of a 512-point FFT, the minimum period sustained was 223 ms. This corresponds to the FFT processing nodes in this pipeline sustaining 53 MFLOPS.

Operating system "drop outs" at crucial points in time affect the size of the minimum period that can be sustained. The conclusion for the OSF/1 oper-

ating system is that periods must be on the order of 100s of milliseconds if the current drop-out effects are to be ignored, i.e., have only a 1% -2% impact. This forces coarse-grain processing in which a relatively large amount of processing is accomplished during each period.

The message-passing rates we were able to sustain improved by a factor of four over the span of the year (from roughly 20 to 80 MB/s). The message-passing rates under SUNMOS were higher still (150 MB/s). At these high rates, however, problems with messages interfering with each other become more significant. The time required for a communication step begins to depend significantly on the message passing activity of other processing nodes.

### 3 Approaches to a Solution

In order to assess general approaches to a solution and to contrast with previous work on related problems, it is useful to consider several characteristic features of the problem of real-time communications scheduling for the targeted applications on MPPs:

1. Tightly coupled MPP architectures. Although the memory is distributed, the nodes are co-located, raw communication latencies are low, and bandwidths are high. In the case of the Paragon, an auxiliary network allows for extremely good clock synchronization.
2. Tightly coupled application programs on the nodes. The programs running at the various nodes can be assumed to be aware of each other and cooperating.
3. A highly distributed communications resource. It may be necessary to take advantage of the potential for many simultaneous communications channels. This feature makes it difficult to use the theory based on ordinary notions of utilization, but does not rule out reducing the larger problem to smaller cases where ordinary utilization is more relevant.
4. Rigid routers. One may have to use routers that cannot take into account the origin of a packet, priorities, or what message a packet is part of.
5. Decentralized control. Central control must be effected "by hand" at a processing node. Even knowledge of communications requests as they may develop in real time is not automatically centrally available.
6. Highly periodic requirements. This is more true of some systems than others, but (as mentioned

above) we initially are targeting cases where most of the message traffic and computation is periodic, and the periods of different tasks are either equal or simply related (although deadlines and release times will not be).

7. Fine time granularity. The set of times that must be considered by the communications scheduler (including release times and deadlines) may be closely spaced. Again, this is more true of some applications than others, but a significant feature of many important cases (including SAR).

To this list might well be added various other problems, such as the possible need for distributed input and/or output. One must also be careful about what requirements must be imposed on the operating system and applications processes (see [11]). Simply guaranteeing that certain messages will be ready before a given time may not be sufficient, because having them ready too early could slow down the transmission of other messages. Thus the importance of lack of "jitter" is amplified in this context. Calculations of required buffer size may also be difficult.

In recent years real-time communications scheduling has been extensively studied by many different research communities for both distributed and parallel environments (see surveys [1] and [9]). Three different "degrees of freedom" are available to reduce the problem to problems where existing theory may be more applicable:

- Temporal — an individual's access to all or part of the communications resource varies with time according to various possible schemes.
- Spatial — divide a network into non-interfering sets of channels.
- Bandwidth — allow a part of a network to be used by more than one entity "simultaneously," as long as the total bandwidth allocated does not exceed some capacity bound. Here "simultaneously" means at the level of detail considered by the scheduling theory.

The multimedia community uses all three degrees of freedom, but incorporates them into a reservation/price/quality-of-service block that has very different assumptions about communications characteristics and guarantees than does the real-time community (see [1]). The coarse time granularity assumed by the multimedia community makes bandwidth subdivision easier.

Other real-time communities do consider a very fine time granularity. But they concentrate on relatively simple networks like buses and rings, and the theories developed depend primarily on temporal subdivision. They generally do not make use of spatial subdivision, even sometimes in cases of rings where it might be physically possible. See [10, 9, 6, 12]. We have concluded that we will probably have to make use of all three kinds of subdivision, at least to some degree, and a level of granularity finer than that of (sometimes quite long) messages must be directly addressed by the analysis and implementing mechanisms.

### The Potential Failure of Fairness

To emphasize the inadequacy of separately providing high performance message passing, high performance processing at the nodes, and a high quality real-time operating system at the nodes, without addressing the communications scheduling component in the integrated way we advocate, consider the following example for what might happen on a machine like the Paragon. Suppose that sixteen nodes arranged in a horizontal row are paired into eight overlapping channels as shown in figure 1. We assume that the message passing software is good enough so that each node can generate messages sufficiently fast to use up the capacity of a horizontal link.

The backplane does not take into account message origin or priority when merging packet streams, but uses a fair merge. This means that when exactly two streams are being merged, and a given packet  $p$  of one stream arrives while a packet  $q$  from the other stream is passing, or if  $p$  arrives simultaneously with  $q$  but loses out, then  $p$  will get its turn next after  $q$ .

Assume data flows from  $n_7$  to  $n_8$  at a rate of, say,  $C$  bytes per second in a steady state. The fair merge policy implies that the data flows along the channel from  $n_0$  to  $n_{15}$  at a rate of  $C/128$  bytes per second. We call this rate disparity the failure of fairness.

It seems likely that for certain pipelined cases this effect will be transient and not a serious problem. With less than full link utilization the disparity will be less, but may still be considerable and has been experimentally observed (see below).

A related problem arises if typical messages comprise only a few packets. A particular packet  $X$  traveling from  $n_0$  to  $n_{15}$  will be delayed at an intermediate node if that node just happens to be putting one of its packets into the stream slightly before the arrival of the head of  $X$ , or if it puts on a packet at essentially the same time as the arrival of the head of  $X$  and  $X$  loses a (fair or unfair) arbitration. With the Paragon

system,  $X$  can get no credit down the line for having been delayed earlier. Nor can  $X$  be given consideration for being part of a large or important message. Even with fair arbitration the possible variation in the arrival time of  $X$  may be troublesome.

Interestingly, the theoretical possibility of the failure of fairness was actually observed in an experiment called the Symmetric Many Pairs benchmark, designed to determine the practical carrying capacity of a link (see [3]). The assumption above that the message passing software allows one sender to fill up the capacity of a link does not apply to our set up. But multiple senders, paired with multiple receivers (who also need time to take packets off the backplane) can be used. This benchmark allocates an even number of processors in a single line on the mesh so that all message passing occurs in one dimension. The layout for 16 processors is also as shown in Figure 1. Sender nodes repeatedly send messages to their mates, which repeatedly receive messages.

Table 1 shows the results for 16 processors of the symmetric many pairs benchmark on a Paragon running OSF/1, revision 1.2.3, with the message coprocessor turned on. Message were of size 1/4 MB and packets 1792 bytes.

#### 4 Packet Stream Analysis

This is an analytical framework designed to be able to provide performance guarantees for message passing schedules that use the parallel capacity of communications networks like that of the Paragon relatively efficiently. It is based on performance parameters of the underlying hardware and software for which we hope that trustworthy bounds can realistically be established. It allows for arbitrary message passing patterns, but does assume the the major requirements are known at compile time.

Hard-real time constraints typically require consideration of deadlines and availability for individual messages. Conceivably, one might choose to schedule the sending and arrival of each packet of each message as a device to ensure the message level requirements. Packet Stream Analysis depends on a cumulative analysis at a level between packets and messages. Individual packets are not scheduled. Their precise fate is left to the vagaries of intricate or obscure lower-level software, hardware, and run-time events. Attention is focused rather on the rates of transmission in packets per second and the times at which those rates are attained.

We are motivated by situations where most messages are relatively long and are sent broken up into many packets of size  $\Delta$ . There may be different packet

sizes for different messages, but we start by assuming that  $\Delta$  is fixed. On the other hand, the optimal value for  $\Delta$  is one of the parameters that one may try to determine using this kind of analysis. Even if only one channel is considered, different packet sizes result in different data transmission rates in bytes per second, because of the overhead of packetization and possibly other factors. In the case of several channels in operation at one time, the packet size has other effects too. For instance, smaller packets result in the stream model being more valid and the smoother mixing of streams, but they require regulated sending at higher frequencies.

If transmission rates are reasonably constant, and individual packet latencies and other overhead costs are relatively small, then one is justified in basing calculations of arrival times primarily on time of initiation of transmission, size of message, size of packets, and transmission rate in packets per second. Other costs can then be treated as second-order corrections.

#### Stream Mixing

An attractive way of viewing the overall carrying capacity of the communications network is that, with certain allowances for overhead and as a function of  $\Delta$ , it can simultaneously support any combination of packet streams moving at various rates as long as an upper bound on the total carrying capacity of each network element is observed. This might allow for an enormously simplified analysis, as long as adequate (and not too drastic) allowances can be made for all relevant categories of overhead.

Initial latency, the transmission time for the first packet of a message, is one kind of overhead. The exact nature of the network interface software and message passing software must be considered at this point, because various other costs, like memory copies, might be relevant. In the case of the Paragon architecture, one kind of overhead would be an allowance for the extra difficulty that a router has when it needs to combine two incoming streams, as opposed to just passing one stream through. It should be optimistically noted that two streams moving in opposite directions through a backplane node do not interfere with each other, at least according to the best information we have at this point.

This view of capacity suggests the intriguing possibility that we may be able to use the Paragon backplane for real-time applications more or less as it was intended to be used in general. The Paragon backplane design is sophisticated, based on a long process of interacting theory and practice, and its designers

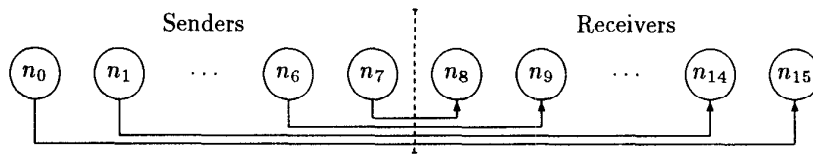


Figure 1: Arrangement of Eight Nested Channels

<i>From - To</i>	0-15	1-14	2-13	3-12	4-11	5-10	6-9	7-8	<i>Sum</i>
<i>Rate (MB/s)</i>	1.5	1.5	3.0	5.8	11.5	22.3	41.7	69.7	157.0

Table 1: Communication Rates for Eight Nested Channels

clearly did not have in mind that applications systems would need to take care that message streams never interfere with each other. Rather, the underlying hardware and software should take care of stream mixing efficiently. All we need to do is get hard bounds on how well this is done and make sure we never ask too much of the system.

Implicit in the above discussion is the idea that under certain circumstances we might ask a node to send a stream at a regulated rate less than the maximum possible. It is unclear how good a mechanism for doing this is available. Even assuming a good real-time operating system at the nodes, this requirement has a “negative” aspect to it that is not of the same character as the requirements usually envisioned for operating systems or message passing software. That is, for most requirements, faster is better. In as much as it seems that stream sending regulation would be much more efficient if done by software at a lower level than the application, this capability may eventually be a serious suggestion for a new requirement on real-time operating systems or message passing software that is motivated by the global communications scheduling problem.

On the other hand, clumsy, user level stream sending regulation is possibly adequate. Note that very low rates can undoubtedly be implemented by the applications process itself. But we would also like to use intelligently timed “blasting”, i.e., sending packets at as high a rate as possible, partly because this is, again, the natural way to use the machine. It may be possible to provide useful guarantees for blasting when combined with regulated sending at very low rates. This highly asymmetric paradigm would be reasonable when most of the message passing traffic is high rate data transmission that has been carefully planned beforehand, but some small amount of miscellaneous message traffic is necessary.

### Feasible Message Transmission Modes

The technical notion of a feasible transmission mode is introduced because of several difficulties with the simplistic rate analysis. How is one to isolate and definitively measure or credibly predict the capacities of each of the network elements involved in the transmission of a message? How are we to combine the overheads associated with the elements of even a single channel into one overall channel rate? Most compellingly, when two streams are combined, how does the associated overhead affect them individually?

Especially when blasting, it may be an unacceptable oversimplification to use exactly prescribed rates in our description of achievable, stable message transmission modes. The rates may vary with time considerably, and the order of starting may be important. There may be hidden aspects of the situation which determine the actual rates in a given trial, so one must be very careful about assertions of what can reliably be achieved. For this reason, we use rate intervals rather than particular values for the rates in the formalization below.

First define a *channel* to be an ordered pair of nodes, a sender and a receiver. Because of the fixed routing algorithm of the Paragon, a channel corresponds to a particular path from the sender to the receiver, at least on the backplane. If one considers in detail all steps involved in message transmission, then it may be possible for two messages from the same sender to the same receiver to take slightly different routes. One way this could happen is that there might be a buffer which is used only in certain circumstances and bypassed in others. It is thus possible that a channel may not be quite the same as a path. For our purposes we can just talk about channels and avoid worrying about paths as such. We do not differentiate among different processes at the same node in the

notion of a channel, although this is a possibility for the future.

A (*message transmission*) *mode* is a vector of distinct channels, together with associated (positive) upper and (non-negative) lower bounds on the packet rates for the channels of the pattern. The channels do not have to be restricted to one horizontal or vertical line.

A message transmission mode  $M$  is *feasible* for a particular packet size  $\Delta$  if simultaneous packet streams for all channels of  $M$  with packets of size  $\Delta$  and packet rates consistent with the bounds of  $M$  can actually be established and sustained. It is not required in this definition that the senders be able to maintain constant rates, only rates within the given bounds. One case where this is particularly relevant occurs when we can guarantee certain of the lower bounds only by blasting. Hidden parameters in this definition include the operating system and message passing software being used.

In order to make guarantees of schedule adequacy as discussed below, it is not necessary to know exactly which modes are feasible. It is enough to know that certain modes are feasible, and we do suppose that this is possible by a combination of experiment and analysis, at least for some relatively simple, but useful modes. The more information of this kind one has, the better analysis of communications schedulability Packet Stream Analysis provides.

### Packet Stream Mode Schedules

A *packet stream mode schedule* (or *PSM* schedule)  $S$  is a sequence of (say  $k$ ) message transmission modes say together with a list of times  $t_1 < \dots < t_k < t_{k+1}$ . Modes may occur more than once in the sequence. The schedule means that the  $i^{\text{th}}$  mode of the sequence is to be used from time  $t_i$  to time  $t_{i+1}$ . Overhead for switching modes must eventually be addressed, as must other sources of inefficiency, of course. A PSM schedule  $S$ , as above, is *feasible* if each of its sending modes is feasible.

We adapt a fairly common notion of message passing requirements. Neither schedules nor requirements are assumed to be periodic, but a more refined version should take into account periodicity. For now, an *individual message requirement* is just given by a sender, a receiver, a number of bytes, a ready time, and a deadline for arrival (which must be greater than its ready time). The *live interval* of an individual message requirement is the interval of time from its ready time to its deadline. A *simple requirement set* is a set of individual message requirements such that no two

individual requirements with the same channel have overlapping live intervals.

Recall that we are assuming relatively small individual packet latencies. Given a PSM schedule and an individual message requirement it is easy to calculate, as a function of time  $t$ , a lower bound on the number of bytes of that message that will have been sent by time  $t$ , assuming that the sending modes indicated by the schedule are followed. Hence one can calculate a worst case arrival time for the entire message. We say that a PSM schedule  $S$  is *adequate* for a simple requirement set  $Req$  if  $S$  is feasible and, for every individual message requirement  $R$  in  $Req$ , the worst case arrival time thus calculated using  $S$  is at most the deadline for  $R$ .

Given a packet stream mode schedule  $S$  which is adequate for a set of requirements  $Req$  and faced with a system whose communications requirements are described by  $Req$ , there are still implementation issues to be faced. At some point one must make allowances for time necessary between modes to coordinate all senders' changes to their next determined rates and receivers. The appropriate mechanism for switching modes is not clear either. If sufficiently good global synchronization is available, then one can simply insert modest pauses to allow for all packets of the last mode to be cleared and for all senders to be assured of having reached their idea of when the switch is to occur. As an optimistic example, the relative difference between the hardware supported local clocks at any two nodes of the Paragon at any given time is guaranteed to be at most one microsecond.

We assume below that artificial padding can be sent (to satisfy the lower bound requirement) in case a schedule calls for a message stream to be sent before the message is really ready or after it has been completely sent. It is not out of the question that such padding might be necessary for the smooth and hence efficient operation of the message passing system.

### The Schedulability Problem

Given a simple requirement set  $Req$ , is there a PSM schedule which is adequate for  $Req$ ? This can be taken to be the fundamental *schedulability problem* for Packet Stream Analysis. There are many, many variants (for instance, one should be able to produce an adequate schedule if there is one, and periodic versions should be considered). The schedulability problem is representative of the difficulties involved with related problems, and we concentrate on it for definiteness.

**Example:** Suppose nodes six nodes  $A, B, C, D, E$ , and  $F$  are arranged in that order from left to right on one horizontal line of the Paragon. There are

three message requirements along the three channels  $A \rightarrow F$ ,  $B \rightarrow C$ , and  $D \rightarrow E$ : one message from  $A$  to  $F$  of length  $L_{AF}$ , one message from  $B$  to  $C$  of length  $L_{BC}$ , and one message from  $D$  to  $E$  of length  $L_{DE}$ . All messages are ready at time 0, and all have deadline  $d$ . Assume we know that three modes ( $M_1$ ,  $M_2$ , and  $M_3$ ) are feasible, where  $M_i$  has lower rate bounds of  $\alpha_i$ ,  $\beta_i$ , and  $\gamma_i$  respectively for channels  $A \rightarrow F$ ,  $B \rightarrow C$ , and  $D \rightarrow E$ , and where the corresponding upper bounds are  $\alpha'_i$ ,  $\beta'_i$ , and  $\gamma'_i$ .

Is there a PSM schedule that is adequate for these requirements and uses only these modes? It should be clear that we may legitimately restrict attention to schedules that start with  $M_1$  from time 0 to time  $t_1$ , then switch to mode  $M_2$  for a duration of  $t_2$ , and end with mode  $M_3$  for a duration of  $t_3$ . Such a schedule is adequate for these requirements if and only if the following seven inequalities are satisfied:

$$(1-3) \quad 0 \leq t_i, \text{ (for } i = 1, 2, 3)$$

$$(4) \quad t_1 + t_2 + t_3 \leq S$$

$$(5) \quad \alpha_1 t_1 + \alpha_2 t_2 + \alpha_3 t_3 \geq L_{AF}$$

$$(6) \quad \beta_1 t_1 + \beta_2 t_2 + \beta_3 t_3 \geq L_{BC}$$

$$(7) \quad \gamma_1 t_1 + \gamma_2 t_2 + \gamma_3 t_3 \geq L_{DE}$$

It is easy to imagine adding various terms to these inequalities to allow for time to switch modes, initial latencies, etc.

Note that the upper bounds are not used in this calculation. They are relevant to implementation requirements, because achieving a given lower rate bound on one channel may depend upon assuming that another, interfering channel is being used at at most some stipulated rate. Also, the upper bounds give a better description of the running system in a way that is quite relevant when questions arise concerning how one might add tasks. A nonlinear problem would arise if we did not take the mode bounds (for instance) to be constants, but variables meeting some extra constraints. This might be desirable if reasonable physical assumptions allow us to conclude that a large class of modes that can be conveniently described parametrically are all feasible.

For example, one might be able to incorporate general knowledge about what modes are feasible by adding another set of inequalities. A simple model would take a mode to be feasible if the sum of its rates (upper bounds) is bounded by a network dependent constant multiplied by a correction factor (between 0 and 1) which depends on the number of message

streams. Note that the other inequalities mentioned so far do not mention upper bounds.

One can also view these sets of inequalities as sources of optimization problems. This is particularly useful because there are a variety of relevant metrics (like total duration or robustness) that might be formulated as objective functions, individually or in combinations.

### A Finitely Constrained Problem

It is informative to consider the following precise mathematical problem, derived by stipulating a finite set of feasible modes and considering only PSM schedules that use only modes from that set. The *finitely constrained* schedulability problem (for Packet Stream Analysis) is then this: given a simple requirement set  $Req$  and a finite set  $\mathbf{M}$  of message transmission modes, is there a PSM schedule that is adequate for  $Req$  and uses only modes from  $\mathbf{M}$ ?

The key idea of the following assertion is the same as in the example above:

**Theorem 1** *There is an algorithm which decides the finitely constrained schedulability problem for Packet Stream Analysis.*

*Proof.* Suppose  $Req = \{R_1, \dots, R_q\}$  is a simple requirement set and  $\mathbf{M} = \{M_1, \dots, M_m\}$  is a finite set of modes. Enumerate in increasing order as  $x_1 < x_2 < \dots < x_p$  all numbers occurring as ready times or deadlines in  $Req$ . If a PSM schedule  $S$  is adequate for  $Req$  and  $1 \leq i < p$ , then one can rearrange and recombine the mode usage by  $S$  between  $x_i$  and  $x_{i+1}$  so that no mode is used more than once on this interval, while preserving the adequacy of the schedule. In fact, this schedulability problem is equivalent to the existence of a solution to the set of linear inequalities constructed using variables  $t_{ij}$  (for  $1 \leq i < p$  and  $1 \leq j \leq m$ ) to stand for the amount of time mode  $M_j$  is used between  $x_i$  and  $x_{i+1}$  and having inequalities of three classes: first,  $t_{ij} \geq 0$ , (for  $1 \leq i < p$  and  $1 \leq j \leq m$ ), second, for  $1 \leq i < p$ ,

$$\sum_{j=1}^m t_{ij} \leq x_{i+1} - x_i,$$

and third, for each  $R$  in  $Req$ ,

$$\sum_{i=a}^b \sum_{j=1}^m \alpha_j t_{ij} \geq L,$$

where  $L$  is the length of  $R$ ,  $x_a$  is the ready time of  $R$ ,  $x_b$  is the deadline of  $R$ , and  $\alpha_j$  is the minimum rate

guaranteed by  $M_j$  for the channel of  $R$ .

Q. E. D.

Of course, concrete algorithms for producing adequate schedules are of more interest than the mere existence of an algorithm. The theorem is stated in this way because no efficient algorithm is possible for a fully general situation. Furthermore, the process of extracting efficient algorithms for contexts in which special assumptions hold is a vast and highly developed theory that we can depend on, but do not want to get into here. This proof is mathematically valid in any case, but the schedules it implicitly produces are reasonable only if one makes use of the assumption mentioned above about the possibility of padding.

An efficient implementation of the above algorithm may be possible and useful for small systems or for restricted parts of larger systems (perhaps looking at one horizontal or vertical line). Larger problems may be mathematically tractable if approximated by linear programming problems using only a reasonable number of modes, but the validity of schedules generated as solutions to large systems of simultaneous constraints would be somewhat suspect. They might be hard to understand, and hence hard to work with or even to trust. Certainly the robustness of solutions in the face of inaccuracy of the input parameters would need to be carefully analyzed. Some structured way of applying the theory (perhaps hierarchically) is desirable if one is to be able to analyze potential system modifications easily. Packet Stream Analysis may also provide theoretical justification to more immediately intelligible scheduling techniques.

#### 4 Conclusion

Commercial MPPs today have a tremendous amount of raw network capacity and the tendency is to assume that such high capacity translates into predictable communications performance. Our experiences and analysis indicate that this is not the case. This paper has framed the real-time communication scheduling problem and presented some of our initial work towards its solution, including an analytical framework called Packet Stream Analysis.

Packet Stream Analysis is not yet a mature analytic framework, but it does seem to present a valid and informative treatment of a significant class of cases. Even if the only practical alternatives for regulated sending are the extremes of blasting and very low rate transmission, combining these as indicated by the inequalities this approach generates may still be useful in establishing guarantees for complex or somewhat dynamic data flow patterns.

This is a relatively new area in terms of relevant

published literature, and theoretical work could be profitably pursued in many directions. But we feel that it is important to let the near-term course of theoretical development be strongly guided by the needs of producing realistic scalable MPP demonstrations. An outstanding problem in this regard is the difficulty of obtaining robust and sufficiently informative measures of the communications capacity of relevant combinations of hardware and underlying software.

Another outstanding problem concerns the interplay between scheduling computational activity and communications activity at the nodes, particularly when it is desirable to overlap computation and communication. We have not explicitly discussed this issue here, partly because it is highly dependent on the particular architecture of the nodes and the operating system, but also because the general problem of lack of direct control that we have discussed above becomes a major factor. We are just beginning to attack this problem. We are currently investigating mechanisms for "controlling" (or influencing) node scheduling from the application level.

Beyond demonstrations of realistic applications, the longer term problem of providing a genuine communications scheduling service for scalable MPP's that enables complex systems to be designed, maintained and updated relatively easily is much more difficult. In fact, the tempting analogy of scheduling tasks on a single processor may be misleading, because of the intrinsic complexity of massively parallel programs and machines.

It may be that any solution to the real-time communications scheduling problem for MPPs more general than specially designed, hand tuned applications will require a good real-time operating system that allows fine-grained control of scheduling from the applications level. More sophisticated routing chips would make the implementation of communications scheduling algorithms easier. Also, operating systems which can guarantee not only that deadlines are met, but also that certain events do not occur too soon, and that certain sequences of events are regularly spaced at prescribed rates would be helpful. Work on real-time MPP communications scheduling will hopefully contribute to the stating of clearer and more complete requirements for real-time operating systems, other software, and MPP hardware. It may even have a positive effect on their future design.

#### References

- [1] C. M. Aras, J. F. Kurose, D. S. Reeves, and H. Schulzrinne. Real-time communication in

- packet-switched networks. *IEEE Proceedings*, 82:122–139, January 1994.
- [2] F. Blitzer. Militarized touchstone program. In *1993 IEEE National Aerospace and Electronics Conference*, volume 1, pages 137–143, 1993. Dayton, OH.
- [3] Curtis P. Brown, Mark I. Flanzbaum, Richard A. Games, and John D. Ramsdell. Real-time embedded high performance computing: Application benchmarks. MTR94B145, The MITRE Corporation, Bedford, Massachusetts, 1994.
- [4] Intel Corporation. Paragon XP/S product overview, 1991.
- [5] Richard A. Games, Arkady Kanevsky, Peter C. Krupp, and Leonard G. Monk. Real-time embedded high performance computing: Communication scheduling. MTR94B146, The MITRE Corporation, Bedford, Massachusetts, 1994.
- [6] M H. Klein, J. P. Lehoczky, and R. Rajkumar. Rate-monotonic analysis for real-time industrial computing. *IEEE Computer*, 27(1):24–33, January 1994.
- [7] V. Kumar, A. Grama, A. Gupta, and G. Karypis. *Introduction to Parallel Computing: design and analysis of algorithms*. Benjamin Cummings, 1993.
- [8] Arthur B. Maccabe, Kevin S. McCurley, Rolf Riesen, and Stephen R. Wheat. SUNMOS for the Intel Paragon: A brief user's guide. In *Proceedings of the Intel Supercomputer Users' Group. 1994 Annual North America Users' Conference.*, pages 245–251, June 1994.
- [9] Nicholas Malcolm and Wei Zhao. Advances in hard real-time communication with local area networks. In *17th IEEE Conf. on Local Computer Networks*, 1990.
- [10] Nicholas Malcolm and Wei Zhao. The timed-token protocol for real-time communications. *IEEE Computer*, pages 35–41, January 1994.
- [11] K. Ramamritham and J. A. Stankovic. Scheduling algorithms and operating systems support for real-time systems. *IEEE Proceedings*, 82:55–67, January 1994.
- [12] L. Sha, R. Rajkumar, and S. S. Sathaye. Generalized rate-monotonic scheduling theory: A framework for developing real-time systems. *Proceedings of the IEEE*, 82(1):68–82, January 1994.
- [13] A. van Tilborg and G. Koob. *Foundations of Real-Time Computing: Scheduling and Resource Management*. Kluwer Academic Press, 1991.
- [14] N. H. Weiderman and N. I. Kamenoff. Hartstone uniprocessor benchmark: Definitions and experiments for real-time systems. *Journal of Real-Time Systems*, 4:353–382, 1992.
- [15] Stephen R. Wheat, Arthur B. Maccabe, Rolf Riesen, David W. van Dresser, and T. Mack Stallcup. PUMA: An operating system for massively parallel systems. To appear in a special issue of *Scientific Programming*. A preliminary version appeared in *Proceedings of the IEEE Twenty-Seventh Annual Hawaii International Conference on System Sciences*, 1994, pp. 56-65.
- [16] B. Zuerndorfer and G. A. Shaw. SAR processing for RASSP application. In *First Annual RASSP Conference*, pages 253–268. ARPA, 1994.