HIGH PERFORMANCE SCALABLE COMPUTING PERFORMANCE MODELING USING PTOLEMY

Eric K. Pauer
Sanders, a Lockheed Martin Company
Signal Processing Center
Nashua, NH 03061-0868
(603) 885-8358, Fax (603) 885-0631
pauer@sanders.com

1. INTRODUCTION

Many of today's signal processing applications are becoming more and more computationally intensive, requiring an increasing number of processors to satisfy their needs. Along with the larger number of processors, the bandwidth needed to pass data among the processors has also increased dramatically. In many classes of applications, designing an architecture to satisfy the data passing requirements while maintaining cost and complexity at reasonable levels is very challenging. An answer to these classes of problems is the High Performance Scalable Computing (HPSC) architecture. It consists of clusters of processing nodes interconnected over a high performance network, implemented with the Myrinet protocol. Application algorithms are partitioned and mapped onto the various processing nodes in the architecture for distributed execution. The goal of this simulation work is to provide a performance modeling capability to automate the evaluation of the relative performance of various architectural candidates with respect to network design choices. In conceptualizing and designing such systems, it will be important to have various simulation capabilities to help in assessing tradeoffs and making system design decisions. A performance modeling capability allows the development team to quickly assess the performance and impact of various design decisions. The results of the performance simulation help in tailoring an architecture and identifying a network topology and routing scheme that best satisfies the system requirements of the application.

2. HPSC AND MYRINET

The purpose of the HPSC architecture is to enable the implementation of computationally intensive algorithms with high data bandwidth requirements on a distributed multiprocessor system. The philosophy behind HPSC is to decouple the application software executing on the processing nodes from the system software which passes data and messages among the nodes. Data is passed between the nodes over the Myrinet network [1], as depicted in a conceptual diagram of the HPSC architecture (figure 1).

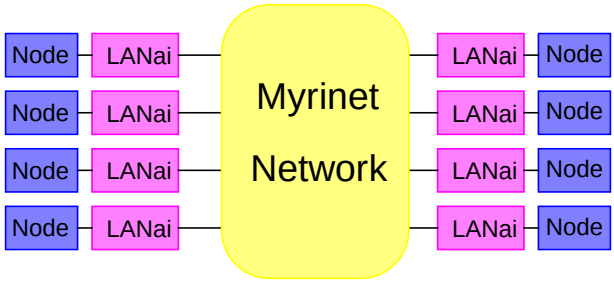

Figure 1: HPSC Architecture and the Myrinet Network

A processing node may contain programmable logic, like an FPGA or ASIC, but often contains one or more programmable processors such as DSP or RISC processors. The HPSC architecture specifies how communication occurs between nodes, but does not specify what happens within a node. All processors within the same node share a common interface to the network, via a Myrinet device called a LANai.

The LANai is a programmable device, and takes responsibility for coordinating the transmission and reception of data from the node to all other nodes over the Myrinet network using the Myrinet protocol. The Myrinet network consists of a topology of multi-port Myrinet switches (4, 8, or 16-port switches are most common), interconnected together, with a LANai as an interface at each node, as shown in figure 2. The LANai sends data to other nodes in the system using routing defined in data synchronization tables (DST) residing in the LANai memory. The DST contains a list of entries, each one specifying information about a block of data to be sent. This information includes the start address of the data within the node, the size of the data block, a list of port numbers for routing the data, and an index for the data at the destination node. When given the command to transmit, the LANai assembles data packets using the information in the DST, and begins sending the packets one after the other. Each data packet is given a header, which starts with the list of port numbers followed by the size and index. The LANai transmits the header and then body of the data packet containing the data block. The first Myrinet switch receives the data, and examines the first port number in the header. It uses this number to determine

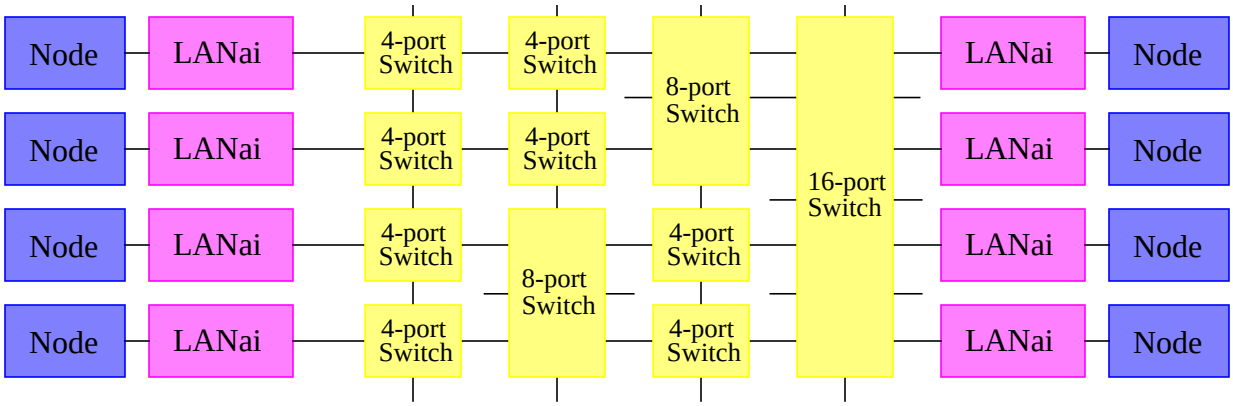

Figure 2: HPSC Architecture Example

which of its ports will send the data. This port number is stripped from the packet, and the switch sends the rest of the packet out the specified port, assuming that port is not currently busy. This continues through a series of switches until the data packet finally arrives at the LANai of the destination node, at which time all port numbers in the header have been stripped and used. On the receiving side, the LANai also has a receive DST to specify the address and size of each data block that it receives. The index sent along with each data packet tells the receiving LANai which entry in this DST to use. Once all data packets are received, the LANai then transfers the data to the appropriate memory locations. Myrinet is a high performance interconnect capability, with a network data rate of 160 Megabytes/second.

As alluded to in the previous discussion, there may be contention within a switch where a port, that is currently busy transmitting, receives a second data packet also needing to use the same port. The switch responds by queuing the second request, and sending a Stop control packet back through the switched network to the originating LANai. The LANai and one port on each of the switches, used to route the packet up to the switch where the contention occurred, are now idle but remain allocated. Once the port becomes free, the switch sends a Go control packet to the originating LANai and the transmission of the packet continues. Multiple requests for the same port are queued and served in the order received. Control packets are small and are not blocked like data packets, as they can be interleaved if necessary over busy or blocked ports. Because of the propogating effect of port contention in a switch, these conditions should be minimized as they will degrade the effective bandwidth of the network.

3. GOAL OF PERFORMANCE MODELING

The first benefit of a performance modeling capability is that it provides a means of evaluating candidate network topologies without having to build them. As the number of nodes and switches grow, it becomes more and more difficult to estimate or predict performance. Questions like how many switches, what type of switches, how they should be interconnected, and what the DST should look like become more difficult to answer, and an automated means of evaluating options becomes necessary. A second benefit is that the DST tables developed in the simulation can often be used directly to program the LANais to implement the desired connectivity when the target hardware is available.

The type of problems that are targeted by this type of simulation are applications that are synchronous in nature. There is a predictable flow of data, and the flow of the data does not depend on the content of the data. Typical applications of HPSC include synthetic aperture radar (SAR), infrared search and track (IRST) systems, and space-time adaptive processing (STAP) based problems. In addition, performance modeling is useful when the customer cannot afford the hardware to provide excess bandwidth and processing capability due to cost limitation or environmental constraints, such as power consumption, weight, and/or size. Thus, performance modeling can be used to predict whether system requirements will be met, and it is very useful in optimizing the size of the architecture required to provide sufficient capability without an excess of hardware.

4. PERFORMANCE MODELING WITH PTOLEMY

Ptolemy [2] is a software environment developed at the University of California at Berkeley that supports heterogeneous system simulation and design using several different models of computation, each implemented in a separate domain. The class of application problems addressed by HPSC falls into Ptolemy's synchronous dataflow (SDF) domain, where the flow of data is predictable and does not change. However, the performance modeling capability uses the Discrete Event (DE) domain as its engine for performance simulation. The DE domain is a discrete-event simulator, which uses a model of computation in which tokens with time stamps, called particles, representing events are passed among the simulation building blocks, called stars. Each star has one or more input and/or output

ports that are used to pass the particles to other stars. In this event-driven model of computation, a chronologically sorted list of events is maintained, oldest first. The simulator's schedule examines the oldest event in the list, and in general, executes the star where that particle resides (special cases may exist for multiple-input stars). All of Ptolemy has been developed in C++ using an object-oriented software architecture to facilitate modularity and extensibility. In addition, all of the source code for Ptolemy is freely available, which facilitates adding extensions to the tool. These attributes of Ptolemy enabled the development of the extensions to support the HPSC performance modeling needs.

5. EXTENDING PTOLEMY'S DE DOMAIN

Extensions to the DE domain, in the form of new stars and particles, were created to support our specific Myrinet performance modeling needs. As part of the HPSC effort, the new stars and particles were developed to model the HPSC architecture and Myrinet protocol. This effort leveraged off of similar performance modeling work started here at Sanders under the RASSP program [3], in addition to the work done by the Ptolemy project.

The key components in the HPSC architecture are shown in figure 3. These new stars include data sources (*SourceNode*), processing nodes (*Node*), LANai interfaces (*LANai*), and Myrinet switches (*4/8/16-port Switches*). The other stars shown are used in hierarchial modeling, which is discussed in section 7. The stars can be considered virtual prototypes of the components, as they implement behavioral models at the appropriate level of abstraction. Each type of star has a group of settable states, which allow the behavior of the model to be adjusted or fine-tuned, as appropriate. The models closely model the actual behavior of the HPSC architecture and Myrinet protocol. However, approximations to simplify the behavior were made when the impact of the simplification was minimal when compared to the savings in terms of simulation time and complexity.

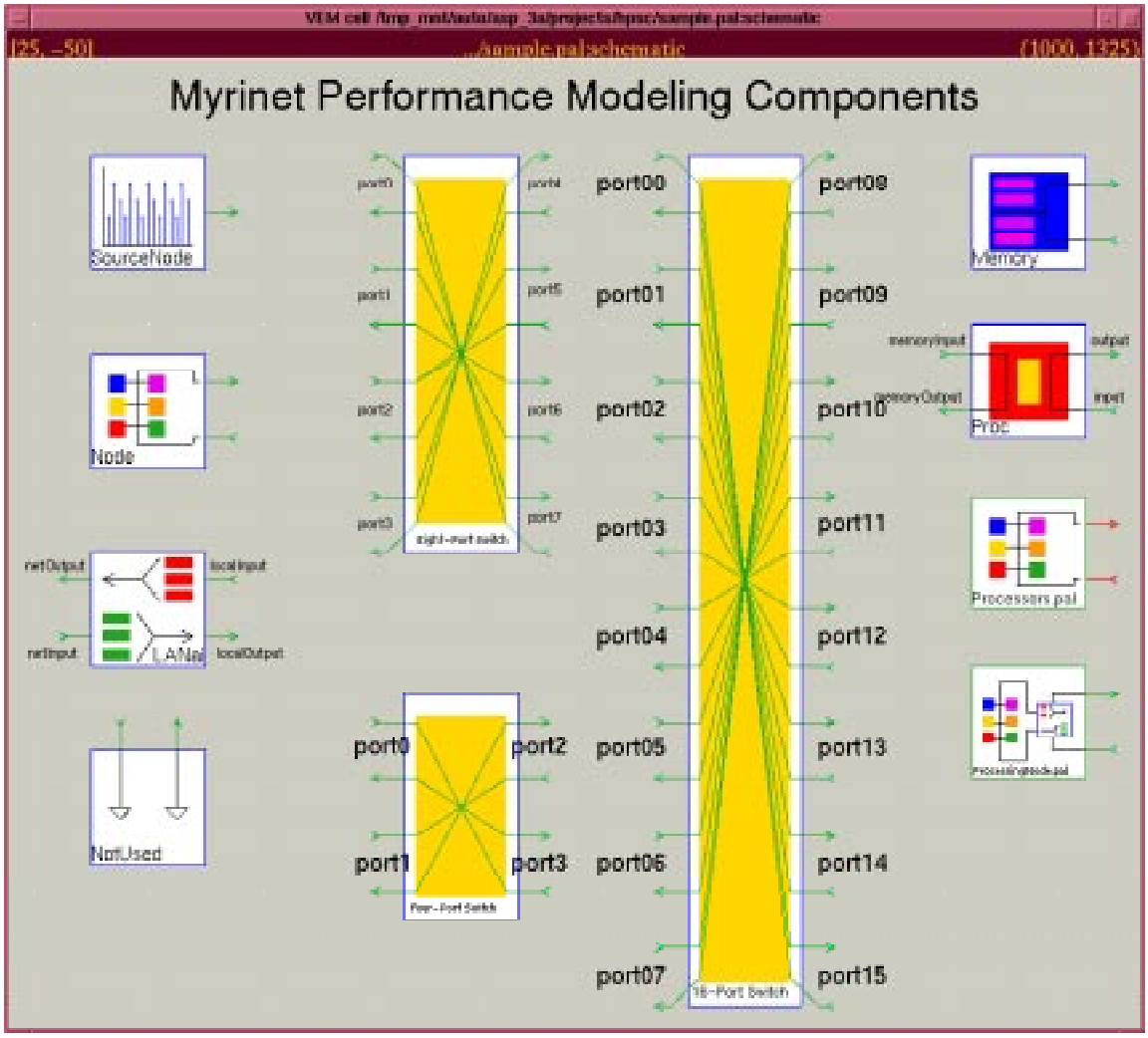

Figure 3: Myrinet Performance Modeling Stars

5.1. SOURCENODE AND NODE STARS

The SourceNode creates blocks of data at a periodic rate. The SourceNode star can represent a sensor or a source of data from another subsystem which is outside the scope of the simulation. The size of the data blocks and their frequency are settable using their state variables. Typically, the SourceNode represents a separate node in the architecture, and hence is connected to its own LANai. The Node star is used to represent a processing node in the HPSC architecture. It models the processing on the node at a fairly high level of abstraction, treating the processing taking place on the node as a single measurable task. Its state variables include input and output data block sizes, a clock rate, and a number of clock cycles needed to complete the processing on that node. It is possible to have a more detailed behavior model on the nodes, using hierarchical modeling which is discussed later. Just as with the SourceNode star, the Node star connects to a LANai star which acts as the interface between the Node and the Myrinet network.

5.2. LANAI STAR

The LANai and Switch stars are responsible for modeling the behavior of the Myrinet network. The LANai has states such as: its clock rate, various latencies (initial transmit, subsequent transmit, receive), local/node side data rate, network data rate, packet header sizes, and transmit and receive DST information. The DST information includes the input packet sizes, output packet sizes, port numbers used for the routing the packets, and destination packet indices. The LANai can simultaneously transmit and receive data from the Myrinet network. A state diagram of the behavior of the LANai is given in figure 4. States are shown with rounded rectangles, particles causing transitions are shown in bold face, and decisions are shown in diamonds. Two state variables (i and ignore) are modified as a result of the transitions, and are shown where appropriate. FB denotes a feedback particle whose purpose will be clear in the next paragraph.

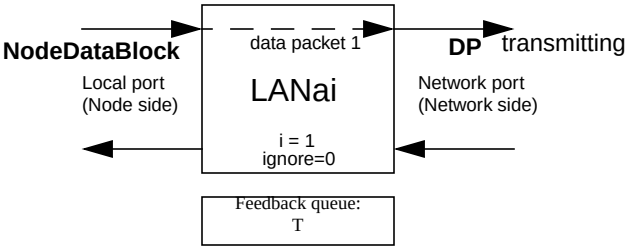

Figure 5: LANai transmitting

The transmit portion of the LANai remains free until it receives a signal from its processing node that data is ready to transmit, which in this model is represented by receiving a NodeDataBlock particle. At this point the LANai moves into the transmit state where it begins transmitting the first (i = 1) of N data packets (DP) listed in its DST, shown in figure 5. Upon moving into the transmitting state, the LANai star calculates a prediction of when the transmission of this packet will finish (time T), assuming the ideal case where no switch contention occurs. The LANai creates a special feedback particle (FB) on an internal feedback event queue time stamped with time T. The particle will cause the Ptolemy kernel to revisit the star at time T. Once this has been accomplished, the kernel returns back to the main simulation to process the next event in the system model. When simulation progresses to time T, the kernel executes the star, determines that the execution is due to this feedback particle, and then causes transmission to commence for the next packet, if any (when i < N), assuming no contention problems (ignore remains zero). This process continues until there are no more packets to send (i = N), at which point the LANai star returns to the free state.

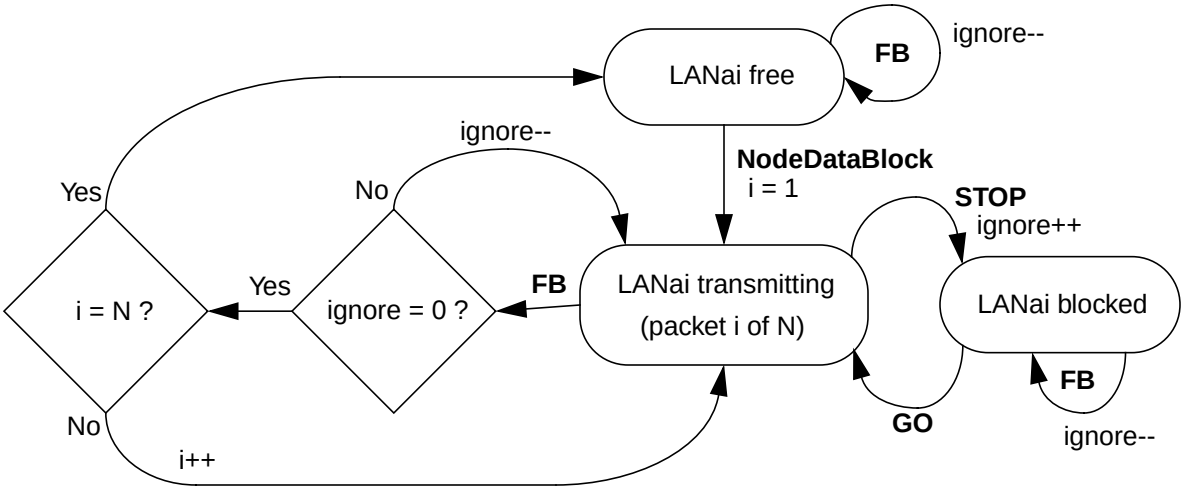

Figure 4: State Diagram of Myrinet LANai Behavior

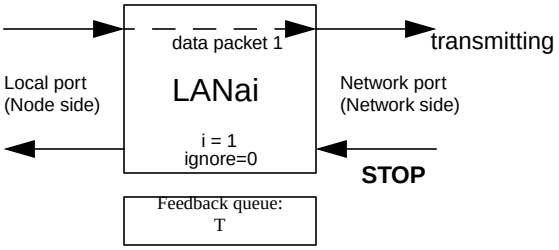

Figure 6a: LANai in transmitting state and receives Stop control packet, moving LANai into blocked state

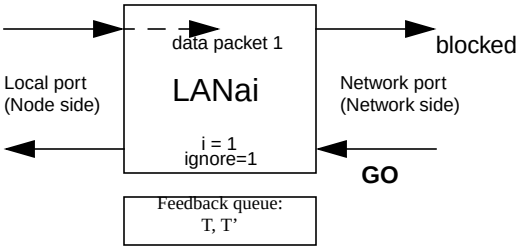

Figure 6b: LANai in blocked state and receives Go control packet, moving LANai back into transmitting state

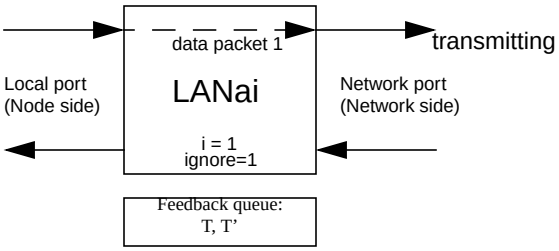

Figure 6c: LANai back in transmitting state, with invalid-feedback particle T and valid feedback particle T'

Assume now that the packet encounters port contention in a switch somewhere in its route. The LANai will be notified of switch contention during transmission by receiving a Stop control packet, as shown in figure 6a. The Myrinet protocol dictates that once a network path has been established, it will not be relinquished until the packet transmission is complete. Thus, this contention will always occur early in the packet transmission. When the LANai receives the Stop control packet, it increments an ignore counter and moves into the blocked state. It remains there until a Go control packet arrives, indicating that transmission may resume, as shown in figure 6b. The LANai then transitions back into the transmitting state to restart the transmission of the packet. It recalculates the time needed to transmit the packet, and creates another feedback particle, time stamped with the new prediction T', as shown in figure 6c. The original feedback particle generated to indicate the completion of transmission at time T is incorrect, and must be accounted for. When the kernel revisits the LANai star due to the original (now invalid) feedback particle, the LANai star ignores it based on its ignore counter. This is possible because it is always true that the second attempt at transmission will complete at a later time than first predicted. Note from figure 4 that this feedback particle may be processed while the LANai is in the blocked or transmitting state.

The receive portion of the LANai operates independently of the transmit portion. The receive side of the LANai is much simpler, and does not have any states. The control packets it receives affect the transmit side of the LANai, as described. As far as incoming data packets, it records their arrival by using the destination index contained in the packet. Once all packets have been received, it notifies the processing node, and prepares to receive the next set of packets. No contention is experienced here, as the switch connected to the LANai arbitrates the data packets and only allows one through at time.

The LANai uses a Ptolemy input and output port for the local side and a Ptolemy input and output port for the network side. Additionally, there is an internal feedback queue as previously mentioned to aid in moving among states. Upon being executed, the LANai must determine which input port has a particle causing the execution, and handle it appropriately. Simultaneous particles (those which share the same time stamp) may occur on two or more input ports, and this case must be handled appropriately.

5.3. SWITCH STAR

The Myrinet switch star also captures and simulates much of the Myrinet behavior. Switches come in a variety of sizes, most commonly four, eight, or sixteen ports. As with the LANai, each port in the switch has a transmit and receive port, with the behavior of the transmit side being much more complex. The state diagram for an individual transmit port on the Switch star is given in figure 7, using the same notation as for the LANai state diagram. For this discussion, this port will be called transmit port A; it also has a companion receive port A. The transmit port A starts out in the free state, as shown in figure 8a. When a data packet arrives at the switch on receive port B to be sent out transmit port A, transmit port A moves into the transmitting state. In this case, the DP B particle causes the transition in the state diagram (where N = B). A feedback particle, with a time stamp corresponding with the expected time the packet transmission will be complete (time TA), is placed in the internal feedback event queue. These actions are shown in figure 8b. If no Stop control packets or other data packets are received in the interim (i.e. ignore and queued counters remain at zero), the kernel will return to this star at time TA as a result of the feedback particle and transition the transmit port A back to the free state. The feedback particles are labeled so that the switch can identify the particles in the common feedback queue for the switch with a specific port.

In the switch, another data packet may arrive on receive port C, which also needs to be sent out transmit port A, while transmit port A is still busy transmitting data for receive port B. If this occurs, transmit port A stays in the transmitting state but queues the request of receive port C and increments its queued counter, as shown in figure 9. This action is repre-

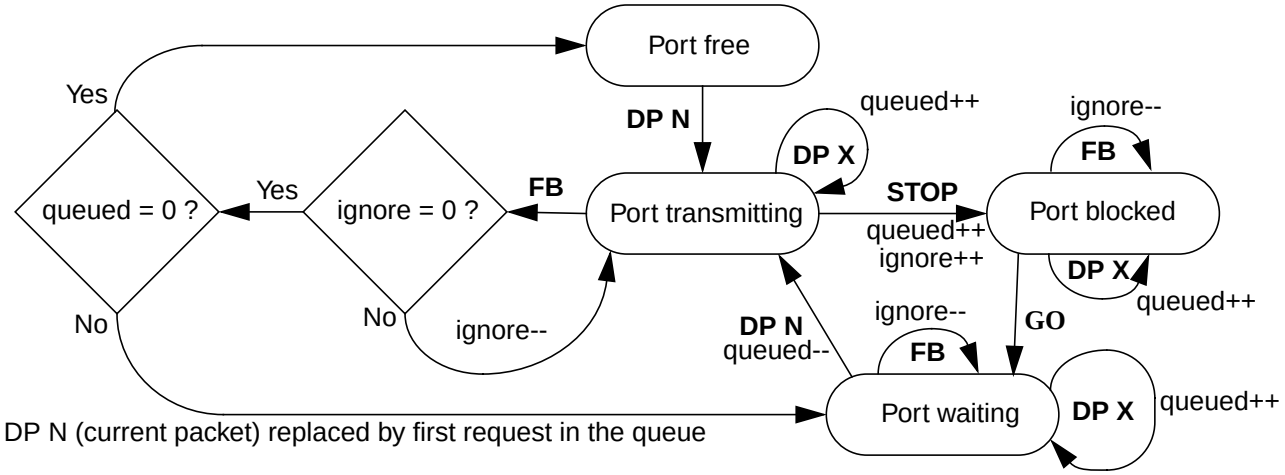

Figure 7: State diagram of Myrinet Switch Port Behavior

sented by the DP X (data packet particle) transition which loops back into the transmitting state in the state diagram. DP X indicates that a data packet is arriving from other receive ports on the switch to be sent out this transmit port. In completing this action, the switch sends a Stop control packet back to the originating LANai through transmit port B to tell the LANai to suspend transmission. This Stop control packet is given appropriate routing so that it makes its way back to the proper LANai. Control packets follow different rules with respect to switch contention; since the Myrinet protocol does not allow them to be blocked, they are interleaved with data packets if necessary in order to reach their destination. Additional data packets arriving on other ports (DP X), also needing to be sent out transmit port A, are also queued in the same fashion and eventually serviced in the order received.

At time TA, the feedback particle, indicating that transmit port A has completed the transmission of the data packet for receive port B, will cause the Ptolemy kernel to revisit the Switch star. Assume for the moment that no Stop control packets were received for this port, implying that the ignore counter remained at zero. The data packet received on receive port C, that was queued, is waiting for transmit port A (queued = 1). Transmit port A is now available, since the transmission is complete. The first request in queue of transmit port A is now serviced, which in this case is receive port C. The switch sends a Go control packet back to the originating LANai through transmit port C to inform the LANai that the port contention in this switch, which blocked it, is now gone and that it can begin transmission. Meanwhile, transmit port A moves into the waiting state, which is used to indicate that it is waiting to service the data packet arriving on receive port C, as shown in figure 10a. In the state diagram, DP N now represents a data packet particle arriving on receive port C (N = C). This state is necessary because it takes some time for the Go control packet to reach the originating LANai through the network, for the LANai to respond by restarting transmission, and for the data packet to propagate back to the switch. When the data packet finally arrives on receive port C, the queued counter is decremented and the transmit port is moved into the transmitting state, shown in figure 10b. Data

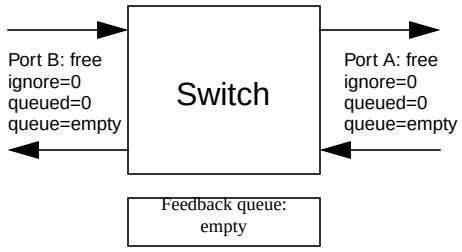

Figure 8a: Transmit Port A in the free state

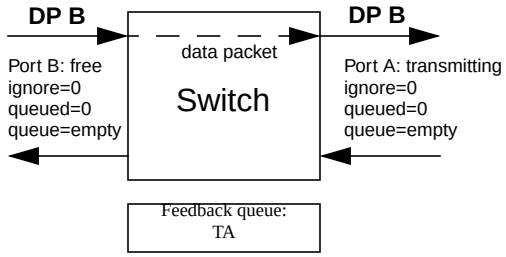

Figure 8b: Transmit port A in the transmitting state, transmitting data packet from receive port B

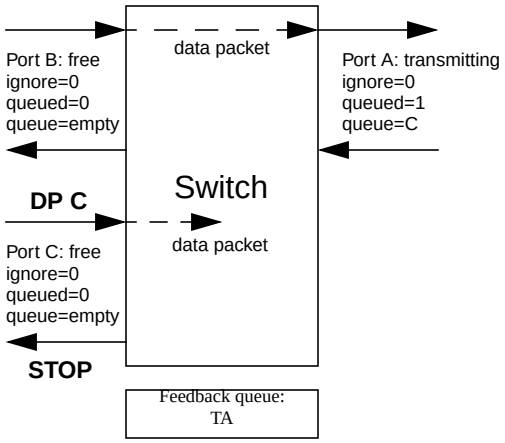

Figure 9: Data packet from receive port C is queued since transmit port A is already busy transmitting

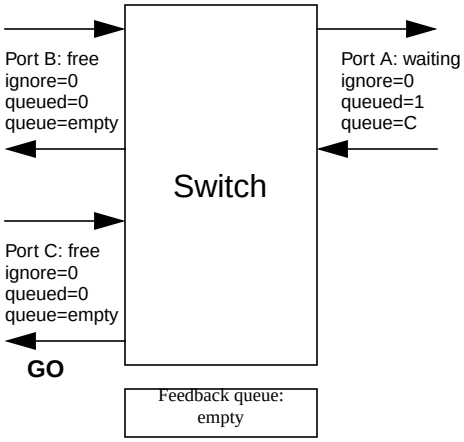

Figure 10a: Transmit port A is in waiting state as Go control packet sent out transmit port C

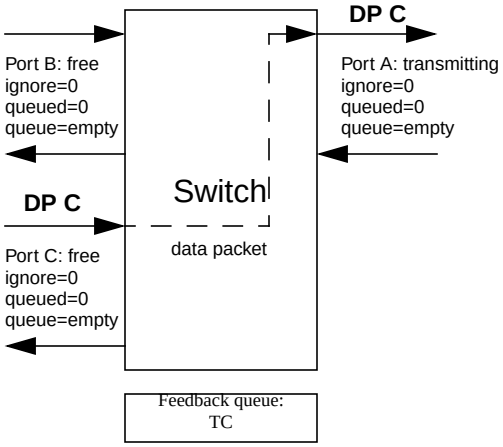

Figure 10b: Transmit port A is in transmitting state, transmitting data packet from receive port C

packets arriving from other than the receive port C are queued as previously described.

The behavior is different when the transmission is interrupted, which occurs when a Stop control packet comes back on the on a receive port, shown in figure 11a. The Stop control packet causes the port to transition from the transmitting to the blocked state, as shown in the state diagram and figure 11a. The Stop control packet is also passed along and sent out transmit port B. The ignore counter is incremented so that the feedback particle TA corresponding to the original end of transmission prediction will be ignored. The data packet transmission is suspended, and is queued on transmit port A, as shown in figure 11b. In this case, the request is placed at the head of the queue instead of being placed at the end. Eventually, a Go control packet will be received on receive port A indicating that transmission can resume (figure 11c), at which time the port will move into the waiting state, and pass along the Go control packet to transmit port B. When the data packet arrives on the receive port B, the queued counter is decremented and transmit port A moves into the transmitting state (figure 11d). It is possible for addi-

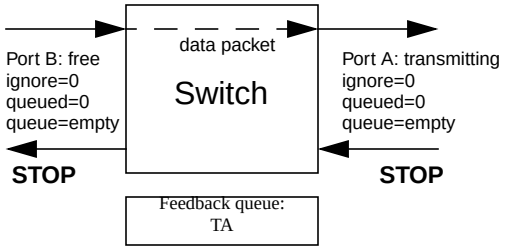

Figure 11a: Receive port A in the transmitting state and receives Stop control packet

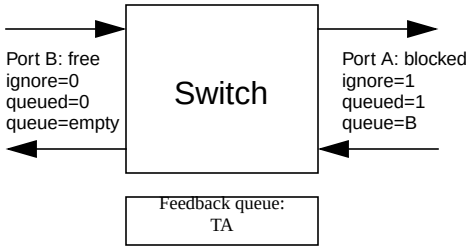

Figure 11b: Transmit port A moves into the blocked state

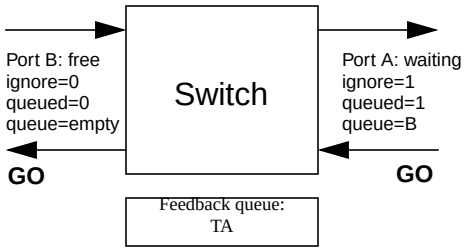

Figure 11c: Receive port A receives Go control packet, moving transmit port A into waiting state

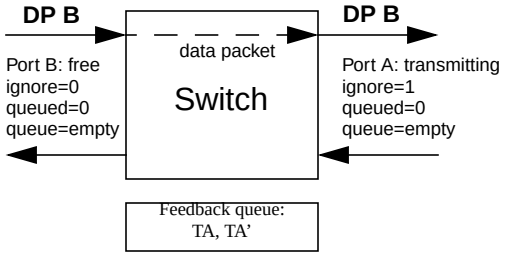

Figure 11d: Receive port B receives data packet, moving transmit port A into transmitting state

tional data packets, that need also need to be sent out transmit port A, to arrive from other receive ports in the switch while it is in the blocked or waiting state. These requests are queued in the same manner as if they were received in the transmitting state.

Feedback particles for the port may cause the Switch star to be revisited. If they are processed while the port is in the blocked or waiting state, the ignore counter is decremented, and they otherwise are ignored because they represent invalid end-of-transmission estimates. If the feedback particles are processed while in the transmitting state, an ignore count of zero indicates that the feedback particle represents a valid end of transmission and results in the port moving back

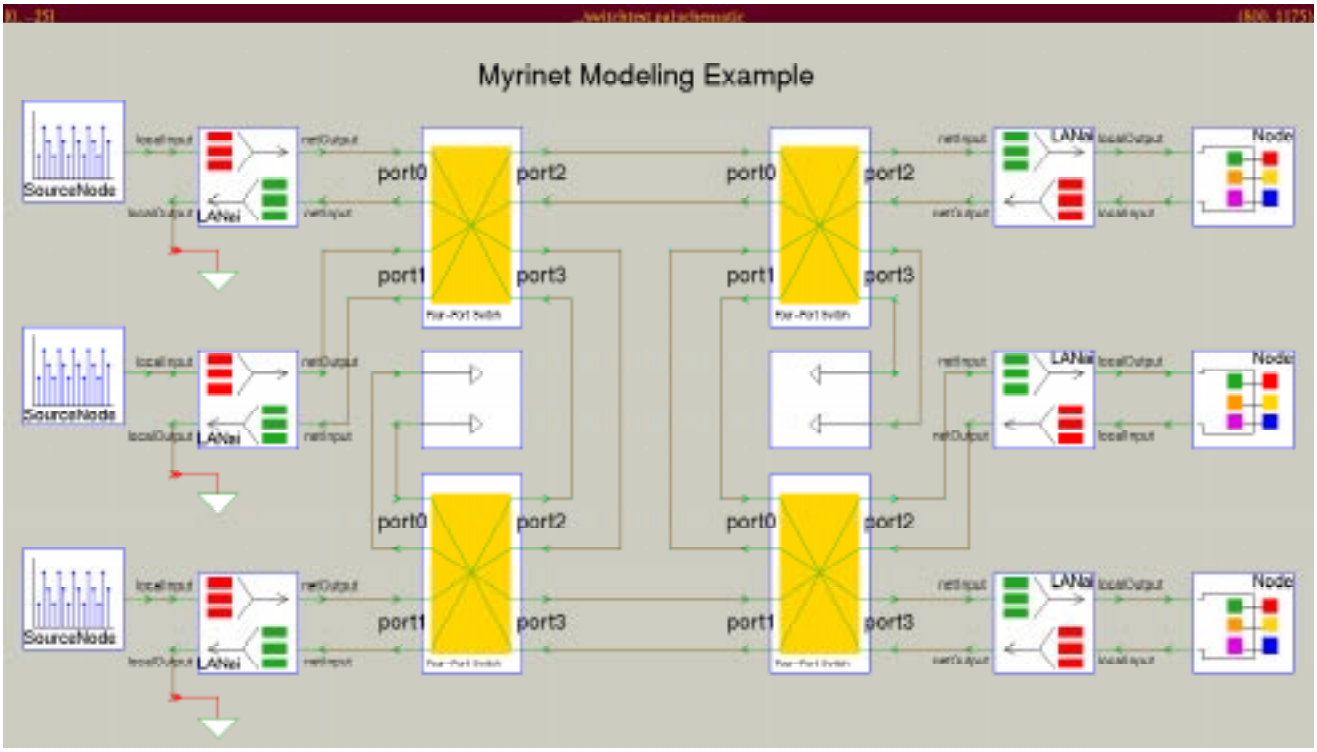

Figure 12: Simple Myrinet Modeling Example

to the free state (no queued requests) or to the waiting state when queued requests exist (queued > 0). If the ignore counter is greater than zero, the particle is ignored, and the ignore counter is decremented as it represents an invalid estimate.

As with the LANai star, when the Switch star is executed, it must determine which of its ports is causing the execution (including the internal feedback queue) in order to properly model the behavior. Each port on the Switch star has been implemented with a Ptolemy input port and output port. To help manage the complexity of the behavior of the Switch star and its various ports, two C++ classes were developed. The PortQueueEntry class is used to represent a request to transmit out a particular port and saves information such as where the packet came from and routing information needed to send control packets. The PortInformation class maintains the state of the port (free, transmitting, blocked, or waiting), various counters (ignore and queued), and a queue of requests for the port (PortQueueEntry objects). It helps in determining the response of the port to various events (Go or Stop control packets, new data packets, or feedback particles), and it also provides a convenient event logging interface for the gantt tool, described in section 6. An instance of the PortInformation class is created for each port in the Switch star so that the behavior of the switch can be modeled effectively. The use of C++ classes in this manner is consistent with the object-oriented software architecture of Ptolemy.

5.4. NEW PARTICLES

As mentioned in section 5.0, several new particles were created for this performance modeling capability. A generic Packet particle, derived from Ptolemy's general purpose Message particle class, was created. It provides a representation of the basic structure of a Myrinet packet and serves as a pure virtual base class for the DataPacket and ControlPacket particles classes. The DataPacket particle provides a representation of a typical Myrinet data packet, and allows a variable body size. The ControlPacket particle is currently used to represent both Stop and Go control packets. A NodeDataBlock particle class was also created, using the Message class as its base as well. The NodeDataBlock particle has been used to represent the passing of blocks of data between the LANai and the processing node. Lastly, an extension to the manner in which Ptolemy handles feedback particles was made to allow them to take on assigned integer values. This was especially useful in the Switch star where the particles could be given values corresponding to the port to which they were associated.

6. MYRINET MODELING EXAMPLES

An example model of a simple HPSC architecture is shown in figure 12. There are three SourceNode stars, four 4-port Switch stars, three Node stars, and six LANai stars. The LANais connected to the top and bottom SourceNodes are each sending two data packets to the top Node and two data packets to the bottom Node. Each of these four data packets are taking different routes. The middle SourceNode is having

Figure 13: Gantt Tool Display of Simple Myrinet Modeling Example

a single data packet sent by its LANai to the middle Node. The states of the LANai stars have been set to reflect this routing. In addition, two NotUsed stars have been used to terminate the unused ports on the Switch stars.

This model was simulated with the performance modeling extensions using the Ptolemy DE domain, with the results being logged into a file. In order to more easily view and interpret the results of the simulation, a gantt tool was developed. Figure 13 shows the display of the gantt tool for this simulation. The gantt tool displays the activity on each resource in rows over time (time is along the x-axis). There is a row for each SourceNode generation of data, each LANai transmit activity, each LANai receive activity, the transmit activity for each port in every switch, the transmit queue for each port in each switch, and the processing on the Node. Thus, most of the stars need several rows to display their behavior and performance. Rows are not displayed when there is no activity, and also, the displaying of rows may be disabled, as was done in this example for the SourceNodes. The various activities have been color-coded to facilitate viewing, but have been shown in gray scale here. Yellow denotes a start up latency, blue indicates normal transmission or reception of data, and green indicates processing of data by the node. Problems are shown in orange and red: orange indicates that one or more blocks have currently originated in the switch port and have caused queuing of requests; red is used where switch ports or LANais are idle due to blocks that occurred somewhere in the current route path. There are also labels containing two integers, on most activities. The first number indicates the data packet's relative position within the transmit DST in the LANai where it was transmitted. This first number of the packet label is different when it is displayed by the receiving LANai, in which case it indicates the relative index of the data packet in the receive DST. The second integer is a unique global identification number assigned to the packet. These numbers are assigned sequentially as packets are created in a given simulation; no two data packets will have the same number. This identification number facilitates the tracing of a given packet through the gantt display from the transmitting LANai, through the switches, to the receiving LANai.

In this example, the gantt tool shows that port contention occurred three times on port 2 of switch 1, and twice on port 2 of switch 2, as a result of multiple data packets vying for the same transmit port. The gantt tool provides a graphical view of which packets are causing the contention, and helps in determining alternatives. The dependency between the reception of data packets and start of processing on the nodes can also be seen. LANais 4 and 6, attached to Nodes 1 and 3 respectively, needed to receive four data packets before their nodes could begin processing. In contrast, LANai 5 needed to receive only one data packet for its Node 2 to begin processing, so it was able to start processing much earlier.

A more involved example of an architecture is shown in figure 14. There are eight SourceNodes and eight Nodes, each paired with a LANai, connected by grid-like topology of sixteen 4-port switches. This application requires each SourceNode to send a data packet to each Node, which means each SourceNode will send eight data packets for a total of 64 packets across the Myrinet network per iteration.

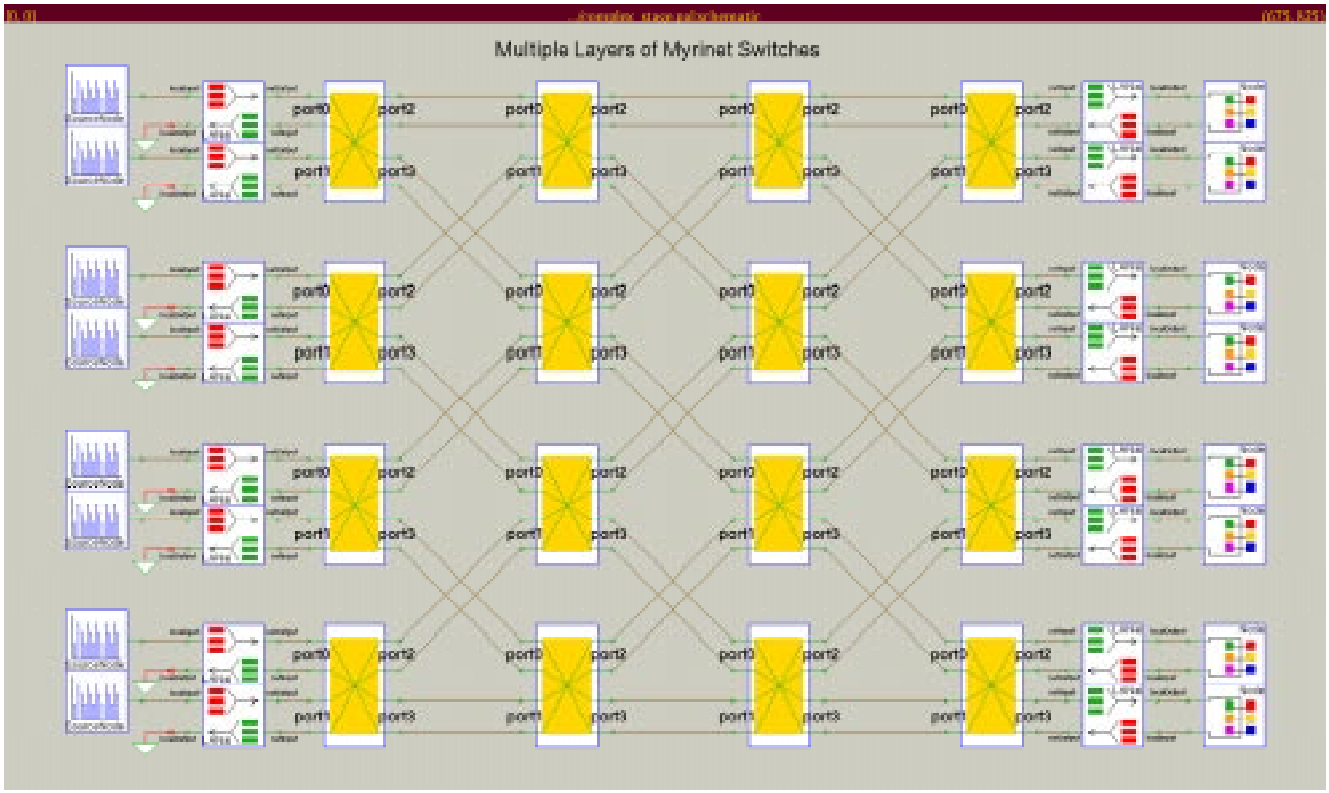

Figure 14: HPSC Architecture with Multiple Layers of Switches

This example is representative of a typical subsystem in a distributed space-time adaptive application.

Figure 15 shows the results of this simulation for each SourceNode to generate a block of data, which took approximately twenty seconds to complete on a Sun SPARCstation 10. A simulation of this size requires more activity rows than can be shown at once; thus, only a subset of the rows are shown. The switches were conveniently named so that their names include their relative XY position within the topology. In this example, there are a number of switches which are experiencing serious contention problems. The performance simulation is useful for determining where potential contention problems exist, and for examining potential solutions. Other options to look at here would be having connectivity from the bottom row of switches to the top row so that packets have more routing possibilities, or considering the use of 8-port switches. In most cases, such as this one, the designer wants to make sure that the latency introduced by the network is short enough so that it does not adversely affect the system throughput requirements, without having a system with an over-abundance of hardware.

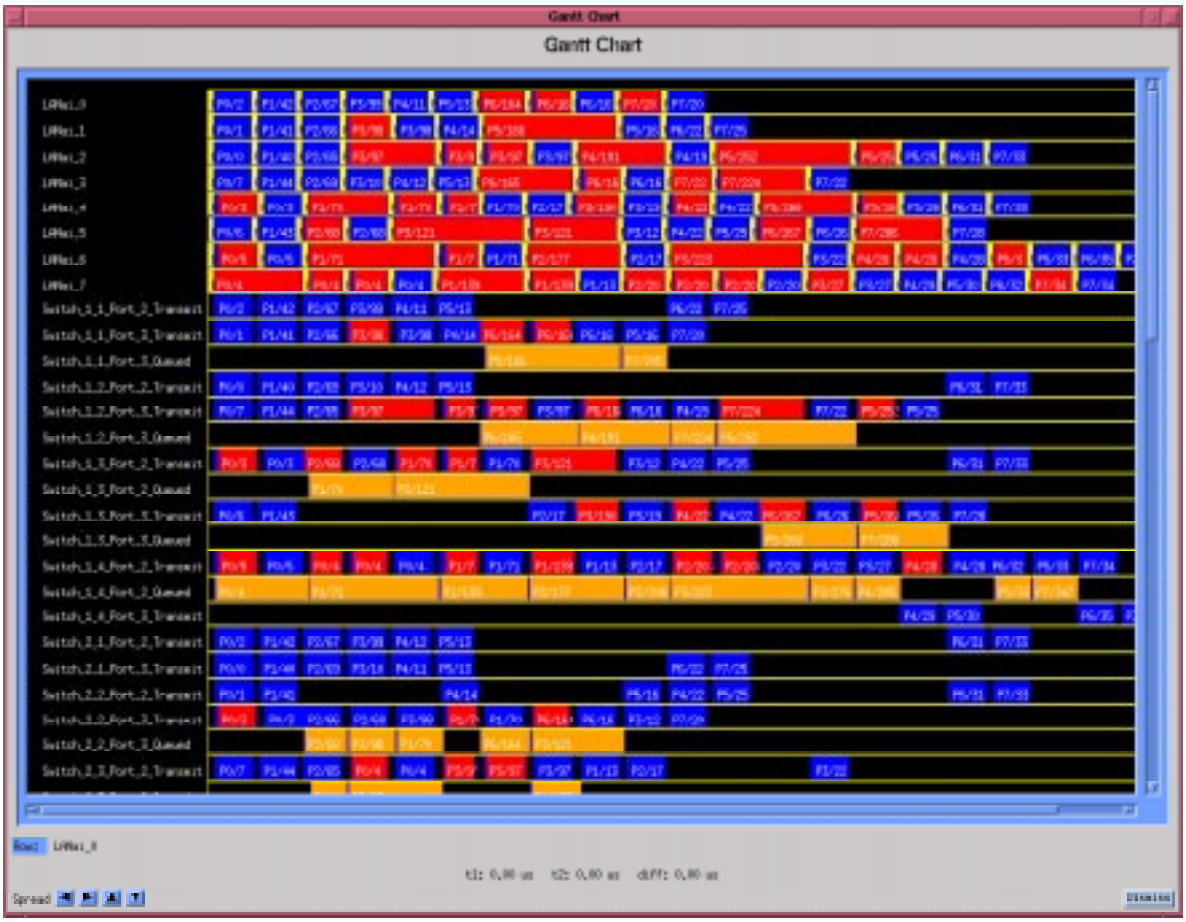

Figure 15: Gantt Display for HPSC Architecture with Multiple Layers of Switches

7. ROLE OF HIERARCHY IN PERFORMANCE MODELING WITH PTOLEMY

To help in managing the simulation of large systems, Ptolemy has a built-in hierarchial capability. Groups of stars can be captured into a single entity called a galaxy, and can be treated as a reusable component. This is useful in defining systems where there are regular patterns in the topology. The hierarchy is also useful for modeling the logical capability of boards that are developed and built, and then used as fundamental building blocks in designing and developing a system solution for an application. As previously mentioned, hierarchy can be useful in refining the performance simulation, especially with respect to the processing nodes. More detailed models of the processor nodes in terms of processors, buses, and memories can be developed, captured, and integrated into this same modeling capability. A conceptual example of using Ptolemy's hierarchy capabilities is shown in figure 16. The Arithmetic Processing Unit (APU) board shown in the upper left window is a galaxy and consists of an 8-port switch with four processing nodes. Each processing node (lower left) is also a galaxy that is in turn composed of a node and a LANai. The lowest level representation of the node has also been captured as a galaxy; the right window shows its composition which includes four SHARC processors with private memories as well as two banks of shared memory. Thus, this hierarchial modeling capability allows different levels of abstraction, to be used where appropriate.

8. SUMMARY

A performance modeling capability to model HPSC architectures and Myrinet using Ptolemy has been developed. These extensions take the form of new stars and particles which implement the behavior of the Myrinet protocol in the Ptolemy Discrete Event domain. By using these extensions, the designer may explore many implementation alternatives in terms of network topology and routing. A gantt tool has been developed to facilitate the viewing and interpretation of the performance results. The hierarchical capabilities of Ptolemy may be utilized to aid in building larger system models or in refining the behavior of the processing nodes. In addition, this performance modeling approach with Ptolemy is quite extensible to other types of architectures, and network protocols and strategies, and can support a variety of system modeling needs. In summary, this capability enhances the ability of the designer to explore many options in order to find the HPSC architecture that best satisfies their system requirements.

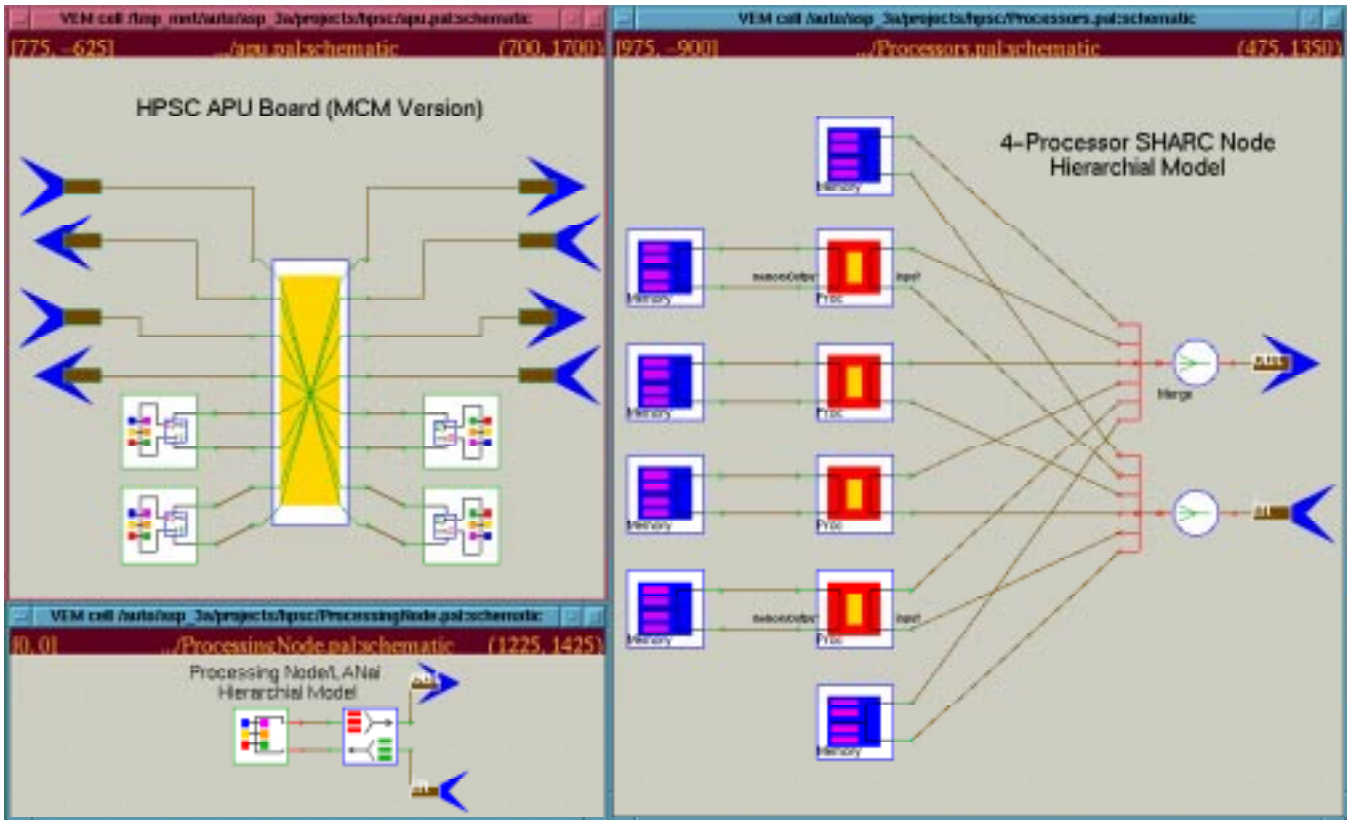

Figure 16: Examples of Hierarchical Performance Modeling within Ptolemy

REFERENCES

1. Myricom, Inc., “Myrinet Link Specification,” Arcadia, California, 1995.
2. E. A. Lee, et al., University of California at Berkeley, The Almagest, Volumes 1-4, Regents of the University of California, 1996.
3. E. K. Pauer and J. B. Prime, “An Architectural Trade Capability Using the Ptolemy Kernel,” IEEE International Conference on Acoustics, Speech, and Signal Processing, 1996.