Modeling the SeaStar Interconnect

SAND2006-4907C

Jason Wertz
Sandia National Laboratories
Albuquerque, NM 87185-0806
E-mail: jswertz@sandia.gov

Abstract
Cray's SeaStar interconnect performance was modeled using Opnet Modeler. This is the interconnect used in Sandia National Laboratories' Red Storm supercomputer. Two versions of the model have been created so far. The first uses a very detailed node model with a small number of nodes. Simple bandwidth, latency, and routing tests were performed. The model results compared favorably with data measured from Red Storm. The second model, which is currently under development, uses a variety of simplified node models derived from the original detailed model. Complex traffic patterns are currently being developed to test the second model.

1.0 Introduction
Sandia National Laboratories is home to the recently built Red Storm Supercomputer. Red Storm is an important resource used to verify and validate the integrity and safety of the US nuclear stockpile. One of its critical components is the Cray proprietary SeaStar interconnect, which allows nodes to communicate with each other in an efficient manner. In order to better understand the SeaStar interconnect chip without using up precious cycles on the live system, a model of it was developed using Opnet Modeler.

There are several broad categories of tests that the model can help with. It can be used to find hidden protocol effects, architecture effects, and congestions effects. It can show where problems might be expected in the real system. It can also demonstrate how certain traffic patterns might affect the system.

Since a model of the system was desired, one of the first questions to be answered was "What would be an appropriate tool to create the SeaStar model with?" The answer was "Opnet Modeler". The main reason behind this was that we didn't want to start from scratch. Modeler provided a framework which simplified the work. It has prebuilt components such as links, queues, and interrupt. It has a well laid out framework for creating custom process models, which can be aggregated into node models and network models. It has built in functions for collecting and displaying statistics, it has debugging capabilities, a memory checker, and it could import/export XML topology files. This last ability to import XML files was very important to this project as it allowed a variety of topologies to be created with a simple C++ program and then imported into Opnet.

2.0 The Model basics
The basic model can be broken down into two main parts. The first part is the node model, which contains many custom built process models detailing the critical functions of the SeaStar chip. The second is the network which connects the SeaStars together. The network portion is modeled after Red Storm and uses a 3-D torroidal mesh.

2.1 Packet Models
There are three basic types of packets needed for the SeaStar model. The first is a *header* packet, which contains information regarding the source and destination of a flow. The second is a *data* packet. The third type of packet is a *network* packet, which consists of one *header* packet and eight *data* packets.

2.2 SeaStar Node Model
The node model of the SeaStar contains eight process models. The first seven, which comprise the SeaStar NIC, are a router, the LCB (Link control block), a PowerPC^{TM}, the Rx_DMA, the Tx_DMA, the RAM, and the bus controller. The eighth process model is for the AMD Opteron processor which is attached to the SeaStar NIC.

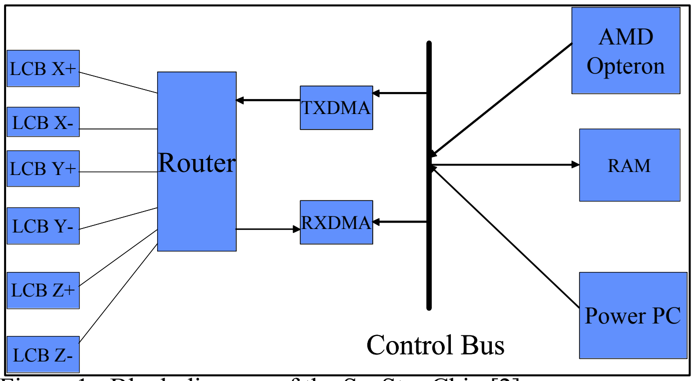

Figure 1: Block diagram of the SeaStar Chip [2]

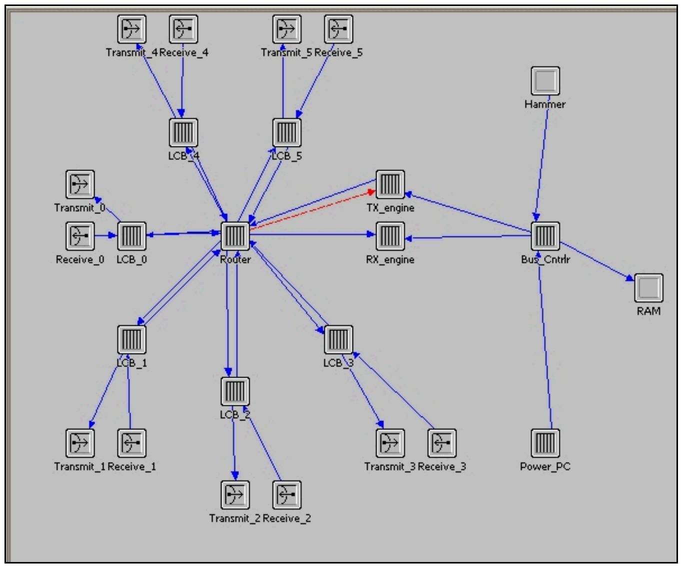

Figure 2: Opnet node model of SeaStar Chip

The router model has six external routing ports and one internal port. *Header* packets and *data* packets are sent out the external ports. *Network* packets are sent between the router and the DMA devices via the internal port. Routing decisions are based on a packet's destination, using a basic dimension-order routing algorithm. Besides just routing the packets, the router model also converts *network* packets into a *header* packet and eight *data* packets and vice versa.

The LCB block sits between the router block and the transmitter/receiver. It provides point-to-point link control. It checks to make sure the next hop has queue space for a packet and it provides a 16-bit CRC check and an ACK to make sure data has been passed intact. If the CRC or ACK check fails, it can retransmit the failed packet or packets.

The PowerPC is the brain of the SeaStar NIC. It controls most of the other pieces of the NIC. It polls the other devices and determines which ones need servicing. The normal events that it deals with are:

1. Does the Opteron have any new messages to be sent?
2. Has the Tx_DMA finished transmitting a message?
3. Is there a new message coming in the Rx_DMA?
4. Does the Rx_DMA have a completed message for the Opteron?

The Tx_DMA block takes messages, as directed by the PowerPC, from the Opteron processor block and breaks them into *network* packets and creates a *header* packet for the message. It forwards the *network* packet to the router on a token bucket scheme (the router must have credits available before the Tx_DMA sends anything to it). The Rx_DMA reassembles *network* packets into full messages. It holds reassembled message until the PowerPC directs it to pass it on. The Rx_DMA also has a mechanism to keep track of incoming messages from up to 256 sources. If more than 256 sources are trying to send to one source, the Rx_DMA will start dropping the excess messages until one or more of the earlier messages finishes.

The RAM block emulates the 384K bytes of RAM the SeaStar NIC has. This is used to hold state information and to handle interactions with the Opteron processor.

The final block is the bus controller. The bus connects many of the pieces of the SeaStar together. It acts as a series of queues between all of the other devices in order to control the flow of data. It can sense when each block is ready to send data and when each block is ready to receive data. The bus also is used as a control channel. The PowerPC and the Opteron are considered master devices, which send signals to other devices through the bus. The Tx_DMA, Rx_DMA, and the RAM are all slave devices to the bus. They receive orders via the bus.

The last process model is for the AMD Opteron processor. This is not technically part of the SeaStar NIC, but it is the tie-in between the host machine and the SeaStar. This block generates messages to be sent out and is where completed messages are sent for processing. The actual system uses HyperTransport^{TM} to talk to the bus. For this model, a simple link is used.

For more details on the SeaStar NIC, see Reference 1.

2.1 Network Model

Once the node model was finished, the nodes needed to be placed into a torroidal mesh. The basic shape of this is a cubic 3D mesh with formed by X, Y, and Z planes with the Z-plane wrapped around to itself (i.e., Zmax links to Zmin). Each non-edge node has 6 neighbors (+-X,+-Y, and +-Z).

For the full sized model of Red Storm, the XxYxZ dimensions of the compute nodes are 27x16x24 for a total of 10,368 total compute nodes. There is also a bank of special I/O nodes on two ends of the machine, which to date haven't been included in the model, that are 2x8x16 meshes for a total of 512 I/O nodes.

3.0 Modeling Methodology

3.1 Tools

The SeaStar model was created using Opnet Modeler 11.0 for Windows. All the process models were built as finite state machines. The router and the LCB process models were based on Opnet's acb_fifo model. The acb_fifo model was chosen as a basis for these two models because it had the basic functionality that was required: a FIFO scheme and a way to control service time. The acb_fifo model was also laid out in such a way that coding for these blocks was straight forward and easy for others to follow. The arrival section was used for starting service and receiving packets, the departure section was used for sending packets and ending service and the service time section was used to create proper timing for the whole block. All of the other process models were built from scratch. The three basic packet models were created using the packet formatter.

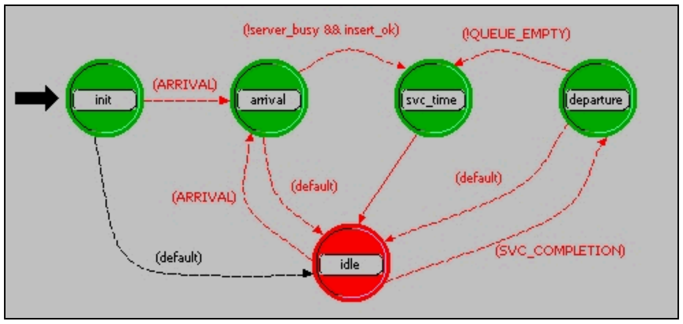

Figure 3: Router process model
-based on acb_fifo process model.

One non-Opnet tool was used for this model. Since it would be difficult to hand create large topologies and because there isn't an Opnet cubic mesh generator, XML generators were written in C++ to generate 2D and 3D topologies. The 3D generator would take in X, Y, and Z dimensions and create a topology. The X-Y plane was created as a subnet on a 100x100 grid (See Figure 4). The Z-planes were handled as a series of connected subnets (See Figure 5). Names were created for each node, which described where the node was in the subnet. (Ex. 'Node_2,3,4' where 2,3, and 4 are the X,Y, and Z coordinates). The links were also generated by the program. The XML file from the generator could be imported directly into Opnet Modeler for testing. The simplified 2D version of the generator set Z as '1' and created a single X-Y subnet.

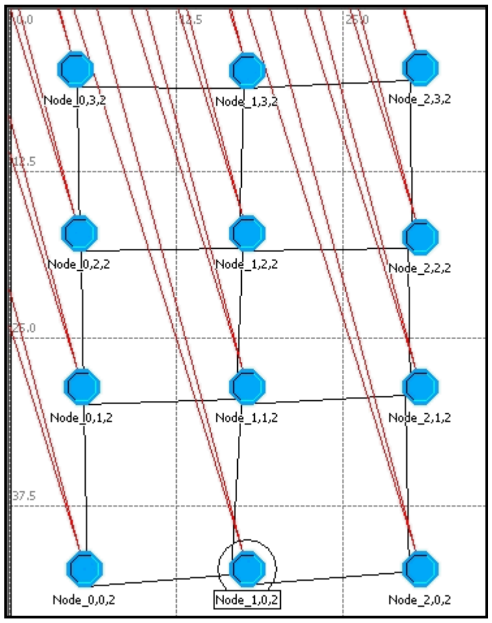

Figure 4: The X-Y Plane for subnet Z=2. [2]
-The red lines connect the nodes to the adjacent Z-planes

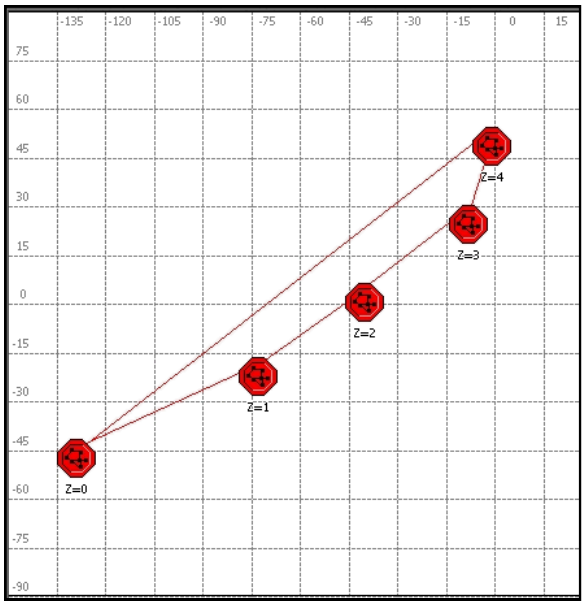

Figure 5: Z-plane, as connected subnets [2]
-Each subnet is an X-Y plane

3.2 Model Tuning

Message timing is the primary parameter that had to be tuned for this model. The basic equation for the message delay is:

Msg Delay ~= Msg. Preprocessing Time
+
Transmission Time
+
Msg. Post processing [2]

The message preprocessing time includes the conversion of the message into packets and the time to add a header. The transmission time includes the time to traverse all the devices in the message path, such as the Tx_DMA, router, LCB, etc. The message post processing time is primarily the time it takes to convert a string of packets back into a message. For small messages, the two processing steps are the dominate time factors. For large messages, the transmission time is the dominate factor. Since small messages and large messages had different timing parameters affecting them, Red Storm data for very small packets and very large packets was used to tune the model. Mid-sized packets were essentially self-tuned in the model once the extreme cases were taken into account.

3.3 Model Validation

Early on in this project there were a lot of questions raised about how accurate a model could be. To answer this, it was important for this model to be validated against real data generated on Red Storm. To do this, simple tests were devised to test latency and throughput against a variety of message sizes. These were tests that could be run on Red Storm and would give a clear answer as to how well the model compared to the actual machine.

3.4 Modeling Compromises

There were a variety of compromises made to smooth out the modeling process. Compromises such as using small meshes, simple traffic patterns, simplified nodes, and time scaling were all used at various points.

The original model used a very detailed set of process models for the SeaStar, but limited the number of nodes used. Early on, it was found that just to import the full 10,368 node mesh took about 30 minutes on a 2GHz machine. Simpler meshes (ex. 3x3x1, 8x16x1, and 4x8x16) were much easier to import and run basic functionality tests on. Simpler meshes are much to simulate.

Simple traffic patterns were also used. In order to test the system, a number of simple tests were created which used many lines of debugging code. Some of these tests were an all-to-one test, each node talking to a specific neighbor test, all-to-one-plane test, tree based traffic patterns, and one-way communication tests (ACKS returned, but no response data returned) were used. Simple traffic patterns make it easier to follow individual flows and simpler to debug problems.

Once the basic testing was completed, it was desired to increase the mesh size. Before doing this, several types of simplified nodes were created. Unnecessary pieces were removed from some nodes. For example, if a node was only passing data through, not generating or receiving, it only needed a router block and dummy sinks for the rest of the NIC. For all the nodes, the LCB was removed. The LCB was deemed unnecessary since Opnet doesn't accidentally corrupt packets. The vital next-hop queue checking function of the LCB was simply added to the Router process model for these nodes. Less complicated nodes made for a slightly faster and smaller run-time model. See Figure 6 for an example of a simplified node model that is used for generating and receiving messages, without all the overhead that the removed pieces would have created. A simple delay was used to simulate the time the other devices would have added.

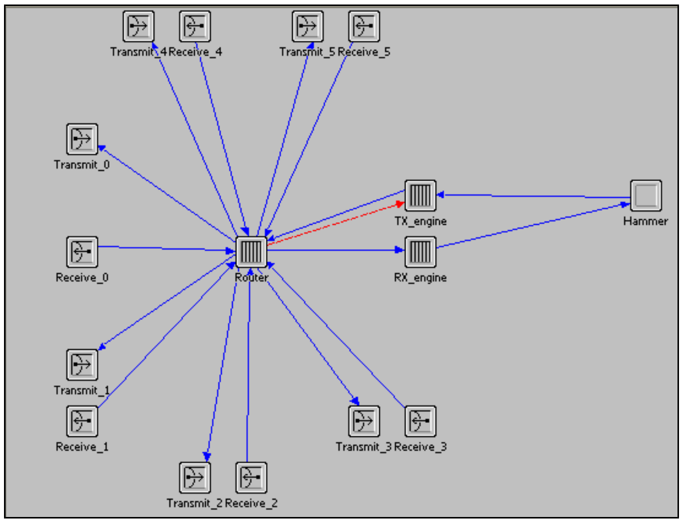

Figure 6: Example of a simplified node model
-no Bus, no PowerPC, no LCB

Another compromise made was that the model was time-scaled by a factor of 1,000,000. One nanosecond of real time became 1 millisecond of model time. This was done to simplify the modeling process and to make the output graphs easier to read.

4.0 Results

4.1 Early models

The earliest model created used a 3x3x1 mesh with fully functional nodes.

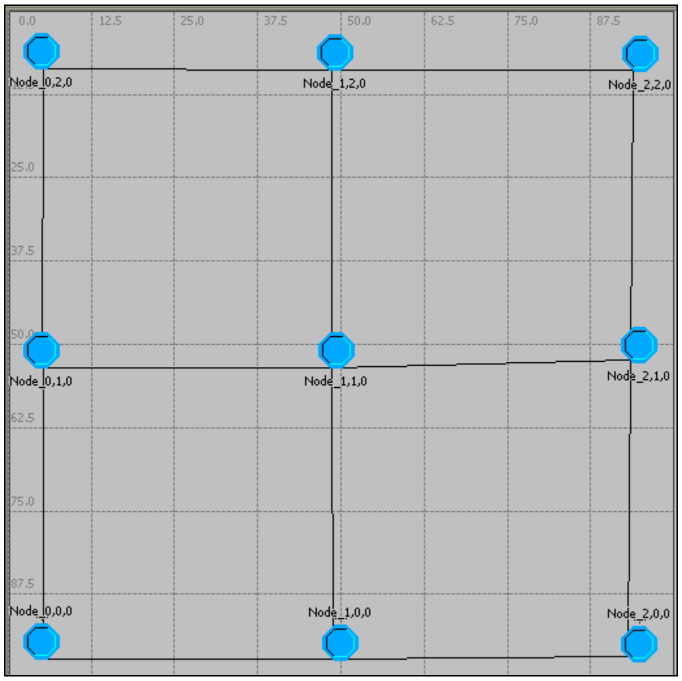

Figure 7: 3x3x1 basic mesh used for early tests

After basic functionality tests were completed on the model, two types of simple tests were run: latency tests and throughput tests. Data from Red Storm was used to validate the model. The Red Storm tests ran transfers between contiguous nodes using a wide range of message sizes from 8 bytes up to 4 million bytes, by powers of two.

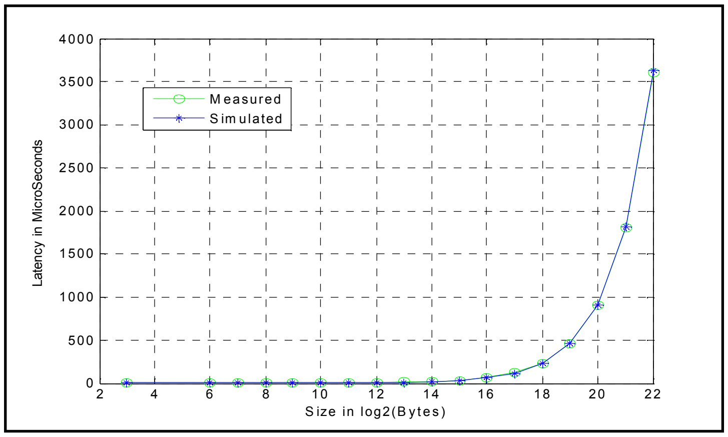

Figure 8: Message Size vs. Delay on Contiguous Nodes [2]

As can be seen from Figure 8, the latency number from the model provided a good match (0.5 percent difference or less).

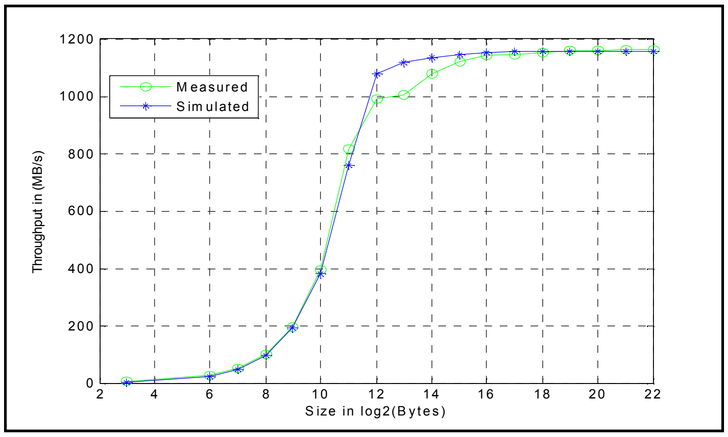

Figure 9: Message Size vs. Throughput on Contiguous Nodes [2]

Examining Figure 9, it can be seen that the throughput results weren't as close as the latency results. The model predicts a smooth curve for the range of message sizes, whereas the Red Storm data showed some anomalies between 2k bytes and 32k bytes.

4.2 Recent Models

The most recent models that have been developed used simplified node models. Currently, newer more complicated traffic flows are being designed to test these models, but so far no reportable results have been generated.

4.3 Issues

One major issue with this model has caused some headaches. When switching between node models, all the links are seemingly randomly reassigned. The expected behavior for changing between nodes which have the same set of external links (same number and same names), would either be that all the links would be reset to having no endpoints or that the endpoints would remain the same as before the change. Having the endpoints reassigned randomly is a pain to fix for small meshes and prohibitively time consuming for larger meshes.

5.0 Future Work

There are several areas for future work. The first is to create more realistic traffic patterns based on actual Red Storm traffic. The second would be to try and import real traffic traces from Red Storm. This may prove difficult if not impossible since it is desired to not only have traces of the traffic passing between nodes, but also to have low level internal traces from the SeaStar chip as well. A third area to look at will be a move from a pure Discrete Event Simulation to a Hybrid simulation. By incorporating realistic background flows and loads, more complicated scenarios can be examined, while only minimally affecting run times. One final piece that may need to be examined is the interface between the compute nodes and the I/O nodes. The I/O nodes have both internal facing interfaces and external facing interfaces, which will require a new node model to simulate. They also use specialized algorithms in order to translate traffic coming from internal compute nodes into traffic that can be passed to external devices.

6.0 Conclusions

There have been many successes with this project. The most important one is that this model proved to a number of people the feasibility of using Opnet to model the internal workings of a hardware chip and its external interactions with other chips. Going into this project there were some doubts about the ability of a 'network modeling tool' to simulate the internal functions of a NIC.

One question arises out of this project: Was Opnet the right tool for this job? The answer is 'Yes.' Looking back on the project, choosing to use Opnet has saved a lot of time. The wide array of tools and functions that are built into made the process of building a new model fairly painless. Time was spent creating the required state machines for the model, not in reinventing the wheel.

7.0 Acknowledgements

The author would like to acknowledge Anand Ganti for helping code the SeaStar model, helping with the Opnetwork 2006 presentation, and for running many of the model tests. The author would also like to acknowledge Keith Underwood for providing briefings on the SeaStar chip, data from Red Storm, and answering many questions.

8.0 Reference

[1] Ron Brightwell, Trammel Hudson, Kevin Pedretti, and Keith Underwood. "SeaStar Interconnect: Balanced Bandwidth for Scalable Performance." IEEE Micro, vol 26, num. 3, May/June 2006.

[2] Anand Ganti and Jason Wertz. "Supercomputing Interconnects." Winter Simulation Conference, 2005.

9.0 Disclaimer

Sandia is a multiprogram laboratory operated by Sandia Corporation, a Lockheed Martin Company, for the United States Department of Energy's National Nuclear Security Administration under contract DE-AC04-94AL8500.