CHAPTER IX

CONCLUSION

Resource-rich clusters are an emerging form of cluster architecture where both host CPUs and peripheral devices are utilized by distributed applications. While these clusters can be physically constructed from commercially available hardware, the enabling technology for these systems is specially designed message layer software. Current generation message layers are ill equipped to handle the communication needs of these clusters because they are by design CPU centric. Therefore this thesis has addressed the design of new message layers that are able to support efficient interactions between applications and a cluster's host CPUs and peripheral devices.

The work presented in this thesis advocates migrating communication functionality in the message layer from the communication endpoints into the NI when possible. This migration reduces the workload of the endpoint and simplifies the task of adding new peripheral devices to the cluster architecture. Three primary design characteristics for message layers have been discussed as a means of accomplishing this task. First, end-to-end flow control in the message layer is managed on a per-hop basis in order to simplify communication protocols for endpoints and provide better dynamic buffer management. Second, logical channels are employed in the NI to allow multiple endpoints in a host to efficiently share a singe NI. Finally, two programming interfaces are defined for the message layer to support a rich set of communication functions. An active message interface provides a powerful means of controlling peripheral devices while a remote memory interface allows users to perform low-level transfers of memory between cluster resources.

A critical characteristic of a message layer for resource-rich clusters is extensibility. Users of resource-rich cluster computers frequently need to perform custom operations and therefore need to be able to layer new functionality on top of existing message layer software. More precisely a message layer must be extensible in at least two dimensions. In a horizontal dimension the message layer must provide means by which new peripheral devices can easily be incorporated into the cluster environment. In a vertical dimension, a message layer must be designed to allow users to add new application-specific functionality. These additions can be made at the endpoint level (e.g., the sockets emulation) or at the NI level (e.g., NI support for multicast).

GRIM is a message layer that has been constructed with the preceding design principles. GRIM has been utilized to integrate four different peripheral devices into a cluster architecture. The fact that these devices have a diverse range of capabilities illustrates that GRIM's communication mechanisms are flexible and sufficient for the needs of resource-rich cluster computers. Multiple application-specific extensions have also been constructed for GRIM. These extensions include methods for performing streaming computations in the cluster, NI-supported multicast, fragmentation and reassembly, and an emulation of the sockets API. These extensions demonstrate that new functionality can easily be layered on top of GRIM's core communication operations.

GRIM's performance has been evaluated and compared to existing message layers. For host-to-host transmissions GRIM obtains a maximum bandwidth of 146 MB/s (1.168 Gb/s) and exhibits latencies as small as 8 μs. This performance is comparable to existing message layers, indicating that it is possible to add resource-rich cluster functionality to the message layer without severely impacting the performance of traditional communication operations. Attempting to implement GRIM's functionality in other message layers is impractical and at the very least highly inefficient due to the manner in which these message layers are designed.

9.1 Implementation Challenges

A significant portion of the work presented in this thesis deals with the challenge of overcoming the limitations of commodity hardware. Some of the more challenging aspects of working with commodity hardware, peripheral devices, and cluster computers in general include the following.

- **Low-level Operation**: Peripheral devices operate as low-level hardware appended to the host system. Programming at this level can be challenging for a number of reasons. Errors at this level often have catastrophic effects on the host. For example, programming a DMA engine with bad pointers can result in the entire operating system being relocated in physical memory. These errors can be difficult to locate as it is more likely that a bad DMA will simply corrupt a random location of host memory that may not cause an immediate system crash. A key to working in this environment is to construct protective debugging mechanisms around functions that pose a risk to system stability.

- **Blind Debugging**: Another programming difficulty in dealing with peripherals is that is often difficult to observe the behavior of firmware. While some high-end cards such as the I_2O adaptor are equipped with a serial debugging connection, many cards have no other monitoring equipment other than LEDs and memory. A significant amount of the work in dealing with the peripheral devices used in GRIM involved constructing debugging frameworks, such as a journaling systems to record card operations. These facilities are essential to observing low-level card behavior.

- **Device Limitations**: One of the most significant problems encountered in this work is simply dealing with the fact that peripheral devices are utilized in ways they were not originally intended for. Peripherals devices are typically built on the assumption that only the host CPU will communicate with the card. This assumption is often used to justify minimizing peripheral device functionality when a host driver can perform the same functions. Therefore work in resource-rich clusters often requires defining new mechanisms by which existing devices can be adapted.

- **Byte Endian and Alignment Issues**: It is common for the processor of different peripheral devices to use a different byte endian order than the host processor. For example, network cards are often big endian (to match network byte order) while x86 processors are little endian. All communication between the host and the NI must be translated to match the destination's endian order. Alignment is a similar issue in that some peripheral devices require data to be aligned on specific byte boundaries. For example, the LANai 9 NI's DMA units require memory address to be align on 64-bit boundaries. Therefore, communication software must be designed to place the right data in the right locations.

- **Limited Data Transfer Mechanisms**: Each peripheral device generally has a card-specific set of hardware for performing operations such as DMA transfers. To complicate matters, some peripheral devices do not provide all of the desired mechanisms for performing data transfers. For example, while the I_2O card features DMA engines, the engines can only be initiated by the card. This adds to the complexity of transferring data to the device from entities such as the host CPU that do not have a built-in DMA engine. Therefore it is beneficial to use a library such as TPIL to accelerate I/O operations.

- **An Evolving OS Kernel**: Over the last five years, GRIM has had to be adjusted to operate with three different versions of the Linux kernel (2.0, 2.2, and 2.4). Each of these transitions required a number of modifications to the device drivers built for GRIM. While it is natural and desirable for an OS to evolve with improvements, maintaining both a working knowledge of the kernel and a functional custom device driver can require a significant amount of effort. One method of dealing with a changing kernel is to move application functionality from the kernel-level device driver to user-space software.

- **Poor Documentation**: A universal problem with working with peripheral devices is that usually there is a lack of decent documentation. Vendors often do not release low-level details for a peripheral device to prevent competitors from leveraging their work. Therefore the only options for developers are reverse engineering and methods based on trial and error. Discussing driver issues with other Linux developers can greatly help in this work.

- **Deadlock**: Deadlock is an important issue that needed to be addressed at all levels of GRIM's development. Any time new functionality is added to a message layer the designer should check to observe whether the operation holds one resource while waiting for another. Deadlock prevention techniques do not have to be complicated and can often be implemented with sufficient buffering.

9.2 Future Directions

The work presented in this thesis provides the first steps in constructing extensible message layers for resource-rich cluster computers. This work can be continued in multiple directions.

9.2.1 GRIM Enhancements

The current version of GRIM provides a basic, flexible substrate for allowing cluster resources to communicate efficiently. However, there are a number of improvements that can be made to the implementation. First, since GRIM is designed to operate with both the old and new versions of the LANai NI processor, the NI firmware does not take advantage of hardware features found in the new card. GRIM's performance could therefore be enhanced by making use of card-specific functions such as the PCI doorbells. Second, like many Myrinet message layers, GRIM only allows one host-level application in a host to be connected to the network at a time. GRIM could be modified to support multiple applications at a time by allocating each application a separate logical channel in the NI. Finally, GRIM can be enhanced by adding new and different peripheral devices to the communication model. Given the flexibility of GRIM and the four existing peripheral device examples, this work can be performed in a relatively straightforward manner.

9.2.2 Gigabit Ethernet Substrates

It is useful to consider how the Myrinet SAN currently used in GRIM could be replaced with commodity Gigabit Ethernet LAN equipment. Gigabit Ethernet is in general more affordable than Myrinet hardware and is widely utilized for clusters. Adapting GRIM to support Gigabit Ethernet would therefore provide an opportunity for a large number of existing clusters to function as resource-rich clusters. The first task in this effort is selecting a Gigabit Ethernet NI card that can be programmed with GRIM's low-level NI functionality. Multiple Gigabit Ethernet cards can be utilized in this effort, including the Alteon AceNIC [9] card, the Intel Pro/1000 series [50] cards, and network cards based on the Intel IXP processor [48]. Emerging IXP cards provide the most promising environment for this work as the cards are very powerful and are well supported by Intel. The IXP cards feature multiple Gigabit Ethernet ports, up to 256 MB of memory, and multiple threaded microengine processors.

Adapting GRIM's NI software to a Gigabit Ethernet NI platform would require changes to some of the basic functions of the NI. At a fundamental level, message data structures would have to be modified to meet the formatting requirements of the new network. A more challenging aspect however is dealing with the communication differences between the Myrinet SAN and a Gigabit Ethernet LAN. While Myrinet provides highly-reliable data transmissions, messages may be dropped or reordered in Gigabit Ethernet LANs. Therefore it is necessary to modify GRIM's reliable transmission mechanisms to account for these factors. While GRIM's in-order delivery mechanisms can be utilized to sort out-of-order messages, extensions are necessary to protect against dropped messages. These modifications involve adding timeouts mechanisms for data message transmissions so that dropped messages are automatically retransmitted. Once a reliable network transmission protocol is established for Gigabit Ethernet NIs, adapting the remaining portion of GRIM's functionality should be a relatively straightforward process.

9.2.3 Active SANs

Another direction for future research related to this thesis is in the field of active SANs. As the streaming extensions of this thesis have demonstrated, it is possible to utilize FPGAs as processing elements in a cluster's network substrate. The next step in this effort is to reduce the distance between the FPGAs and the NI. One such approach is to include an FPGA on a NI card. The advantage of this architecture is that the FPGA can process network messages without costly traversals of the PCI bus. Since the FPGA can process messages at a finer granularity, it is possible for the FPGA to play a more pivotal role in streaming computations.

Emerging FPGA architectures provide another opportunity for research exploration related to active SANs. Recently announced FPGA chips such as the Xilinx Virtex-II Pro [100] include large amounts of reconfigurable logic, network transceivers, as well as dedicated processor cores. These chips will be capable of directly interacting with the physical links of networks such as InfiniBand. Therefore these FPGAs can be visualized as the next generation of high-performance NI chips. In these chips a portion of reconfigurable logic and CPU processing time will be utilized to implement network interactions. The remaining resources of the chip can be utilized to implement custom computations for streaming operations. Since the NI and FPGA are implemented in a single chip, it is expected that constructing a streaming computational system will be much more efficient and straightforward than the effort required to incorporate the RC-1000 FPGA card as a coprocessor. However, the system for performing streaming computations on network messages presented in this thesis is applicable to this architecture and provides a starting point for future research in this field.