EFFICIENT IMPLEMENTATION OF PARTIAL INTEGRATED NETWORK

BARRIERS

BY

BRYAN BARNEY REAGAN
B.S., University of Illinois at Chicago, May 1991
M.S., University of Illinois at Chicago, June 1995

THESIS

Submitted as partial fulfillment of the requirements
for the degree of Doctor of Philosophy in Computer Science
in the Graduate College of the
University of Illinois at Chicago, 2004

Chicago, Illinois
ACKNOWLEDGEMENTS

I would like to thank my advisor Dr. Jon A. Solworth, not only for academic leadership, but also for his extreme patience during the last few years. I would also like to thank the other members of my committees, Dr. Wolfgang Martin Boerner, Dr. Shantanu Dutt, Dr. Joe Hummel, Dr. Ashfaq Ahmad Khokhar, Dr. John Lillis, and Dr. Robert Sloan. Also particularly deserving of thanks are the emergency proof-readers, Manigandan Radhakrishnan and the Very Reverend Frank M. Levi. I would also like to thank my parents Barney and Joan Reagan, and other members of my family and friends who encouraged me in this endeavor.

BBR
# TABLE OF CONTENTS

<table>
<thead>
<tr>
<th>CHAPTER</th>
<th>INTRODUCTION</th>
<th>PAGE</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.</td>
<td></td>
<td>1</td>
</tr>
<tr>
<td>2.</td>
<td>BACKGROUND</td>
<td>4</td>
</tr>
<tr>
<td>2.1</td>
<td>Early Barriers and Flush Primitives</td>
<td>4</td>
</tr>
<tr>
<td>2.1.1</td>
<td>Early Synchronization Techniques</td>
<td>4</td>
</tr>
<tr>
<td>2.2</td>
<td>Barrier Synchronization</td>
<td>4</td>
</tr>
<tr>
<td>2.2.1</td>
<td>Origin of the Term: Jordan’s Barrier Synchronization</td>
<td>4</td>
</tr>
<tr>
<td>2.2.2</td>
<td>Brook’s Butterfly Barrier</td>
<td>5</td>
</tr>
<tr>
<td>2.2.3</td>
<td>Tournament or Fan-In Tree Barrier Synchronizations</td>
<td>8</td>
</tr>
<tr>
<td>2.2.4</td>
<td>NYU Ultra-Computer’s Fetch-And-Add</td>
<td>10</td>
</tr>
<tr>
<td>2.2.5</td>
<td>Connection Machines Two and Five</td>
<td>11</td>
</tr>
<tr>
<td>2.3</td>
<td>Packet Switching in the Abstract</td>
<td>12</td>
</tr>
<tr>
<td>2.3.1</td>
<td>Flits, Packets, and Queues</td>
<td>12</td>
</tr>
<tr>
<td>2.4</td>
<td>Memory Operation Completion</td>
<td>16</td>
</tr>
<tr>
<td>2.4.1</td>
<td>Mohan Ahuja’s Flush Primitives</td>
<td>17</td>
</tr>
<tr>
<td>2.4.2</td>
<td>Renade’s Network Merging Algorithm</td>
<td>18</td>
</tr>
<tr>
<td>2.4.3</td>
<td>Mahapatra and Dutt’s Termination Detection Algorithm</td>
<td>19</td>
</tr>
<tr>
<td>2.5</td>
<td>The Route Step Function and Routes</td>
<td>21</td>
</tr>
<tr>
<td>2.6</td>
<td>Queue Dependence and Queue Dependence Graphs</td>
<td>26</td>
</tr>
<tr>
<td>2.7</td>
<td>Plies and Virtual Interconnection Networks</td>
<td>27</td>
</tr>
<tr>
<td>2.8</td>
<td>Interconnection Networks</td>
<td>29</td>
</tr>
<tr>
<td>2.9</td>
<td>Integrated Network Barriers</td>
<td>32</td>
</tr>
<tr>
<td>2.9.1</td>
<td>The Integrated Network Barriers of Birk, Gibbons, Sanz, and Soroker</td>
<td>33</td>
</tr>
<tr>
<td>2.9.2</td>
<td>Stamatopolous and Solworth’s Edge Wave Integrated Network Barriers</td>
<td>37</td>
</tr>
<tr>
<td>2.9.3</td>
<td>Barriers and Congestion Control</td>
<td>41</td>
</tr>
<tr>
<td>2.9.4</td>
<td>McShane’s Perfectly Pipelinable Barriers and Corner Waves</td>
<td>42</td>
</tr>
<tr>
<td>2.9.5</td>
<td>An Extended Example: A Linear Network of Three Nodes</td>
<td>45</td>
</tr>
<tr>
<td>3.</td>
<td>PICKETS, CLOSED AND OPEN REGIONS</td>
<td>51</td>
</tr>
<tr>
<td>3.1</td>
<td>Source, Destinations, Routes, and Pickets</td>
<td>52</td>
</tr>
<tr>
<td>3.1.1</td>
<td>Entry Pickets</td>
<td>53</td>
</tr>
<tr>
<td>3.1.2</td>
<td>Exit Pickets</td>
<td>54</td>
</tr>
<tr>
<td>3.1.3</td>
<td>Picket Implementation</td>
<td>55</td>
</tr>
<tr>
<td>3.2</td>
<td>Regions</td>
<td>55</td>
</tr>
<tr>
<td>3.2.1</td>
<td>Closed Regions</td>
<td>56</td>
</tr>
<tr>
<td>3.2.2</td>
<td>Open Regions</td>
<td>57</td>
</tr>
<tr>
<td>3.3</td>
<td>Why Are Open Regions Interesting?</td>
<td>58</td>
</tr>
</tbody>
</table>
### TABLE OF CONTENTS (CONTINUED)

3.4 An Extended Example: Open Region in the Center of a Two Dimensional Mesh ........................................................................ 59
3.5 An Open Region And Four Ply Virtual Interconnection Networks ............................................................................. 62
3.6 Recursive Tiling of a Network ................................................................. 67

#### 4 EXPERIMENTAL METHODS AND RESULTS

4.1 Network Model .......................................................................................... 71
4.2 Performance Metrics .................................................................................. 73
  4.2.1 Packet Latency .................................................................................... 74
    4.2.1.1 Packet Latency on an Empty Network .................................... 74
    4.2.1.2 Non-Empty Queues .................................................................... 76
    4.2.1.3 Contention Between Queues ..................................................... 76
    4.2.1.4 Barrier Latencies ......................................................................... 78
    4.2.1.5 In Saturation ................................................................................ 78
  4.2.2 Bandwidth ............................................................................................ 79
  4.3 Simulations without Regions .................................................................... 84
    4.3.1 Single Ply Virtual Interconnection Networks and Edge Waves .... 84
    4.3.2 Four Ply Virtual Interconnection Networks and Corner Wave Barriers ................................................................. 88
    4.3.3 Summary .......................................................................................... 91
4.4 Tiling with Closed Regions ...................................................................... 92
4.5 Tiling with Open Regions ......................................................................... 93
4.6 Time Stamp Versus Barrier Arbitration .................................................. 99
4.7 Acknowledgements and Sequential Consistency .................................... 100

#### 5 CONCLUSIONS AND FUTURE WORK

5.1 Conclusions .............................................................................................. 109
  5.1.1 Conceptual Contributions .................................................................. 109
  5.1.2 Experimentally Verified Observations ............................................. 110
5.2 Future Work .............................................................................................. 111
5.3 Applications .............................................................................................. 112

### CITED LITERATURE

.......................................................... 113

### VITA

.......................................................... 115
<table>
<thead>
<tr>
<th>TABLE</th>
<th>PAGE</th>
</tr>
</thead>
<tbody>
<tr>
<td>I.</td>
<td>25</td>
</tr>
<tr>
<td>II.</td>
<td>25</td>
</tr>
<tr>
<td>III.</td>
<td>63</td>
</tr>
<tr>
<td>IV.</td>
<td>84</td>
</tr>
<tr>
<td>V.</td>
<td>87</td>
</tr>
<tr>
<td>VI.</td>
<td>88</td>
</tr>
<tr>
<td>VII.</td>
<td>89</td>
</tr>
<tr>
<td>VIII.</td>
<td>91</td>
</tr>
<tr>
<td>IX.</td>
<td>93</td>
</tr>
<tr>
<td>X.</td>
<td>95</td>
</tr>
<tr>
<td>XI.</td>
<td>96</td>
</tr>
<tr>
<td>XII.</td>
<td>97</td>
</tr>
<tr>
<td>XIII.</td>
<td>97</td>
</tr>
<tr>
<td>XIV.</td>
<td>100</td>
</tr>
<tr>
<td>XV.</td>
<td>103</td>
</tr>
<tr>
<td>XVI.</td>
<td>104</td>
</tr>
<tr>
<td>XVII.</td>
<td>105</td>
</tr>
<tr>
<td>XVIII.</td>
<td>106</td>
</tr>
<tr>
<td>XIX.</td>
<td>106</td>
</tr>
</tbody>
</table>
# LIST OF FIGURES

<table>
<thead>
<tr>
<th>FIGURE</th>
<th>PAGE</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. A Butterfly Barrier Synchronization for Eight Nodes</td>
<td>8</td>
</tr>
<tr>
<td>2. Communication Structure in a Tournament</td>
<td>10</td>
</tr>
<tr>
<td>3. A Queue as Drawn in Literature</td>
<td>14</td>
</tr>
<tr>
<td>4. Schematic of Processor, Queue, And Interconnection</td>
<td>16</td>
</tr>
<tr>
<td>5. Sample Network of Queues</td>
<td>24</td>
</tr>
<tr>
<td>6. A Two Dimensional Mesh Network Topology</td>
<td>31</td>
</tr>
<tr>
<td>7. Layout of a Simple Switch</td>
<td>32</td>
</tr>
<tr>
<td>8. An Eight Node Omega Network with Hung Barrier</td>
<td>36</td>
</tr>
<tr>
<td>9. Split with two successors</td>
<td>39</td>
</tr>
<tr>
<td>10. Gantt Chart of Staggered Data and Barriers on a Four VIN network</td>
<td>42</td>
</tr>
<tr>
<td>11. North East (X Positive Y Positive) Corner Wave</td>
<td>44</td>
</tr>
<tr>
<td>12. One Dimensional Array In a 1 Ply VIN</td>
<td>46</td>
</tr>
<tr>
<td>13. One Dimensional Array In a 2 Ply VIN</td>
<td>49</td>
</tr>
<tr>
<td>14. An Entry Picket</td>
<td>54</td>
</tr>
<tr>
<td>15. An Exit Picket</td>
<td>55</td>
</tr>
<tr>
<td>16. Closed Region on an Omega Network</td>
<td>58</td>
</tr>
<tr>
<td>17. Open Region and Enumerated Areas</td>
<td>62</td>
</tr>
<tr>
<td>18. Four Ply VIN Packet Directions by Wave and Ply</td>
<td>64</td>
</tr>
<tr>
<td>19. Area Swept by Wave on Ply 0</td>
<td>66</td>
</tr>
<tr>
<td>20. Sample Recursive Division Regions</td>
<td>67</td>
</tr>
<tr>
<td>21. Closed Region Tiling of a Single Virtual Interconnection Network</td>
<td>69</td>
</tr>
<tr>
<td>22. Open Region Tiling across Four Virtual Interconnection Networks</td>
<td>70</td>
</tr>
<tr>
<td>23. A Join with Three Predecessors</td>
<td>77</td>
</tr>
<tr>
<td>24. Lines Crossing Links on Bisections</td>
<td>81</td>
</tr>
<tr>
<td>25. Simple Bisection Example</td>
<td>83</td>
</tr>
<tr>
<td>26. Graph of Mean Bisection Usage</td>
<td>92</td>
</tr>
<tr>
<td>27. Graph of Mean Bisection Usage With Open Region Tiles</td>
<td>99</td>
</tr>
<tr>
<td>28. Plot of Achieved Load Ratio for Barriered No Acknowledgements</td>
<td>107</td>
</tr>
<tr>
<td>29. Plot of Achieved Load Ratio for Barriered and Read Acknowledgements</td>
<td>108</td>
</tr>
</tbody>
</table>
## LIST OF ABBREVIATIONS

<table>
<thead>
<tr>
<th>Abbreviation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>BMC</td>
<td>Barrier Marker Counter</td>
</tr>
<tr>
<td>INB</td>
<td>Integrated Network Barrier</td>
</tr>
<tr>
<td>QDG</td>
<td>Queue Dependency Graph</td>
</tr>
<tr>
<td>VIN</td>
<td>Virtual Interconnection Network</td>
</tr>
<tr>
<td>X+</td>
<td>X Positive Direction (Increasing Values of X)</td>
</tr>
<tr>
<td>X-</td>
<td>X Negative Direction (Decreasing Values of X)</td>
</tr>
<tr>
<td>Y+</td>
<td>Y Positive Direction (Increasing Values of Y)</td>
</tr>
<tr>
<td>Y+X+</td>
<td>Y Positive X Positive Direction</td>
</tr>
<tr>
<td>Y+X-</td>
<td>Y Positive X Negative Direction</td>
</tr>
<tr>
<td>Y-</td>
<td>Y Negative Direction (Decreasing Values of Y)</td>
</tr>
<tr>
<td>Y-X+</td>
<td>Y Negative X Positive Direction</td>
</tr>
<tr>
<td>Y-X-</td>
<td>Y Negative X Negative Direction</td>
</tr>
</tbody>
</table>
SUMMARY

This thesis develops techniques for implementing partial integrated network barriers (partial INBs) on packet switched interconnection networks.

An Integrated Network Barrier (INB) is a barrier in which operation completion is integrated with synchronization. A partial barrier is a barrier in which only a subset of the processors participate. In this case, Proc(S’) denotes the set of source processors, which participate in the barrier and initiate operations, and Proc(D’) denotes another set of processors, which are be effected by the operations initiated by the processors in Proc(S’).

In the language of Graph Theory, a route is a walk on the queue dependency graph, and a deadlock free walk is a path in the queue dependency graph (Chartrand, 1985). Continuing in this vein, a ply is the union of an arbitrary set of non-disjoint routes. A virtual interconnection network (VIN) is a collection of plies which contains at least one route between every valid source – destination pair. A network can be viewed as a collection of one or more VINs.

The Total Embedding Property stipulates that a route must traverse queues from exclusively one ply. This guarantees that component waves of a barrier propagate independently on each ply.
A physical ply is ply which is the closure of all queues which are either predecessors or successors of some queue. When partial barriers are in use, the set of routes between the queues associated with \( \text{Proc}(S') \) and \( \text{Proc}(D') \), denoted \( R' \), forms a ply which may be embedded in a larger physical ply. Pickets are used to limit how the set of queues traversed by the routes in \( R' \), denoted \( Q' \), interacts with those queues in the physical ply but not in \( Q' \), denoted \( Q-Q' \). Entry pickets mark queues in \( Q-Q' \) which are predecessors in the physical ply to queues in \( Q' \), that should be ignored when checking for barrier markers. Similarly, Exit pickets mark queues in \( Q-Q' \) which are successors in the physical ply to queues in \( Q' \), to which queues in \( Q' \) may not route packets or barrier markers.

Pickets can be used to create two types of regions: closed regions where \( \text{Proc}(S') = \text{Proc}(D') \), and open regions where \( \text{Proc}(S') \subseteq \text{Proc}(D') \). Closed regions act in isolation, and behave like smaller networks of the same size as the region. A network can be tiled with open regions, each region on its own VIN, which use partial barriers to emit packets to other processors, which maintain barrier order even when they reach processors not in Proc\((S')\). Such tilings can use partial barriers not only to drive network bandwidth usage toward the theoretical maximum (100% at the bisection,) but also to enforce barrier semantics while doing so. Most significantly, open regions with barriers can be used to implement sequential consistency with much higher performance than the traditional technique, in which processors block pending acknowledgement of each operation they initiate.
The purpose of this thesis is to develop techniques to efficiently implement Partial Integrated Network Barriers on packet switched interconnection networks.

While it will be more properly defined in Chapter 2, a barrier is a means of coordinating the activities between multiple processes or processors. Furthermore, an Integrated Network Barrier is a barrier which incorporates operation completion with synchronization (Solworth and Stamatopoulos, 1993). A barrier is partial when only some of the processors or processes participate (Birk et al, 1989). Informally, a packet switched interconnection network is a communications network in which information to be transferred is encapsulated in finite size packets, which may be stored en route.

Chapter 2 presents the background material upon which the work of this thesis depends. First it highlights the development of barriers for synchronization. The focus of this section is on developments relevant to this work, rather than on presenting a comprehensive survey. Then a brief overview of packets, queues and packet switching is presented, so that the terminology is available for later sections. Then operation completion is introduced, including Mohan Ahuja’s Flush Primitives, Renade’s Network Merging Algorithm, and a brief look at Mahapatra and Dutt’s solution to the related Termination Detection Problem. Next some general information about routing is presented, including some terminology, notation, and functions, some of which were created for this thesis, but are presented here for use in later sections of Chapter 2. Next queue dependency and queue dependency graphs are described. Then plies and virtual
interconnection networks are explained. Then simple interconnection networks for multi-processors are described. Finally, Integrated Network Barriers are explained, including those developed by Birk et al, Solworth & Stamatopoulos, and those proposed by McShane. Finally Chapter 2 ends with an extended example to demonstrate these concepts.

Chapter 3 contains most of the conceptual work of this thesis, although as noted earlier, some new concepts and terminology were introduced in sections of Chapter 2, in order to allow their use in describing and examining earlier work. First Chapter 3 addresses the issues of routes between sets of sources and destinations in the network. Plies are re-examined as sets of routes, which can be embedded within physical plies in a network. Pickets are the introduced as a mechanism to separate a ply embedded inside a physical ply from the rest of the physical ply. Two types of pickets are defined, entry pickets and exit pickets. Regions are defined as sets of processors, which can be considered open or closed based on whether all destinations are also sources of packets. Chapter 3 then closes with two examples, one describing pickets for an open region on a two-dimensional mesh, and another describing how multi-ply virtual interconnection network, like those required to support McShane’s type barrier waves, would embed routes into plies.

Chapter 4 contains summaries of experimental work to test the concepts in this thesis. Specifically, software simulating networks was implemented in C++, and large numbers of test cases were simulated and data collected. The simulator is described, as are the performance metrics used. The following experimental results are given.
1. Profiles were created to calibrate the simulator, in order to confirm that its behavior is similar to the simulator used in Stamatopoulos and Solworth’s work.

2. Test were performed to determine the characteristics of multi-ply virtual interconnection networks, and to test the barriers proposed by McShane.

3. Closed regions were tested and their performance was verified to be similar to small networks of the same size.

4. Open regions were tested in order to verify that for a network with single ply virtual interconnection networks, which is tiled with open regions, partial barriers can improve performance as well as provide barrier semantics.

5. Barrier time stamp arbitration was tested and demonstrated to partially offset the performance degradation associated with the use of multiple virtual interconnection networks of the multi-ply type.

6. Most significantly, it was confirmed that partial barriers over an open region can produce significantly better network performance while used to maintain sequential consistency, compared to the single outstanding operation technique in common usage.

Chapter 5 presents the conclusions drawn, and also presents some possible applications to network designs beyond those internal to a computer.
2: Background

2.1 Early Barriers and Flush Primitives

2.1.1 Early Synchronization Techniques

Even before the term barrier synchronization was coined, techniques were developed to synchronize concurrently executing processes or processors. Most of these early techniques relied on specific hardware or operating system features. Many of these techniques used counters of one form or another, which were either incremented or decremented when a process (or processor) reached the synchronization point. These techniques required each individual process to gain access to the counter, often using a lock, semaphore, or similar mutual exclusion device to ensure exclusive access. This meant that the time to execute was linear with the number of processes synchronizing, and also created contention as the different processes or processors competed for access. In addition, while some schemes used notification to inform the waiting processes that all had reached the synchronization point, many of these implementation used “busy waiting”, which further aggravated the “hot spot” at the counter. Excellent surveys of these early techniques can be found in (Birk et al, 1989) or (Glew and Hwu, 1991).

2.2 Barrier Synchronization

2.2.1 Origin of the Term: Jordan’s Barrier Synchronization

The term barrier synchronization was coined by Henry F. Jordan in 1978 (Jordan, 1978). On page 265, he described a barrier as a division between two stages of a computation, in which “no processor should start the latter until all complete the former”.

4
Jordan’s implementation of a barrier used dedicated hardware. The machine used message passing with “first-in first-out” buffering, but did not have direct hardware support for shared memory. Furthermore, it had two independent communications networks: A set of dedicated processor-to-processor links between select pairs of processors in a fixed topology; and a time multiplexed, common bus linking all processors.

Jordan’s scheme required that each processor have two single bit flag registers, called the barrier flag and report flag respectively. The flags of a given type were wired together by priority chains, each consisting of parallel daisy chained “and” and “or” lines. Upon reaching the barrier, a processor set its report flag true and then waited. One processor was designated the controller, and was responsible for coordinating the end of the barrier when all report flags were true by clearing the barrier flag. At that time, all processors would then proceed. Jordan also proposed the use of multiple barrier flags so that more complicated protocols could be implemented (Jordan, 1978).

2.2.2 Brook’s Butterfly Barrier

In 1986, Eugene D. Brooks published a barrier synchronization technique, based in part upon Lamport’s Logical Clocks (Lamport, 1978), which used the existing shared memory (Brooks, 1986). On page 295, Brooks describes barrier synchronization as “all of the participating processors are required to meet at the barrier before any are allowed to proceed”.

Given P processors, Brook’s barrier uses \(\log_2(P)\) phases of pair-wise processor synchronizations. These paired processor synchronizations used shared memory, and are used in
turn as building blocks to create larger barrier synchronizations. Each processor has a single, unique location in shared memory, called its flag, which it uses for all synchronization operations. For narrative purposes, each processor’s associated flag in shared memory shall be called its local flag, and the flag of the processor with which it is synchronizing shall be called the remote flag. During a given step of the synchronization, a processor may access one flag, either by clearing it (setting it to zero), setting it to one, or busy waiting until its value is a specified value. Each processor in a synchronization pair executes an identical set of four operations using the flags.

From the point of view of one processor, in a pair performing a paired processor synchronization, four steps are required:

1. The processor busy waits until its local flag is zero; (Essentially waiting to ensure that step 4 has completed from any previous calls.)
2. The processor sets its local flag to one;
3. The processor busy waits until it detects that other processor has set the remote flag to one;
4. The processor clears (sets to zero) the remote flag.

Brooks points out that if the flags are distributed on different memory banks, then this can be done without creating a performance threatening, frequently accessed and contested “hot spot” (Brooks, 1986).
Larger barrier synchronizations are performed by performing multiple sets of concurrent paired processor synchronizations. The design works optimally for systems with numbers of processors equal to integral powers of two, but is adaptable to systems with other numbers of processors by using the pattern for the next largest power of two, and ignoring the synchronizations with non-existent processors. If the system has \( N \) processors, where \( N \) is an integral power of two, then \( x = \log_2 N \) iterations are required. Assume that the processors are numbered from 0 to \((N - 1)\), so that operations may be performed on the processor numbers in order to identify the processor to be synchronized on a given iteration. Given processor number \( p \) in \([0, (N-1)]\), during the \( j \)th iteration, with \( j \) in \([1, \log_2 N]\), each processor \( p \) performs paired processor synchronizations with processor number \( p' \), where \( p' = p \oplus 2^{j-1} \), and \( \oplus \) is the bitwise exclusive or operator. Observe that the processor numbers could be interpreted as the addresses of nodes in a hypercube and the pairs which execute the two processor barrier synchronizations would be immediate neighbors in the hypercube topology.

For example, consider a butterfly barrier synchronization on multi-processor with eight processors, labeled 0 through 7. This would require \( 3 = \log_2 (8) \) iterations. The processor pairs which perform paired processor synchronizations on each iteration are as follows.

1. \((0, 1), (2, 3), (4, 5), \) and \((6, 7)\)
2. \((0, 2), (1, 3), (4, 6), \) and \((5, 7)\)
3. \((0, 4), (1, 5), (2, 6), \) and \((3, 7)\)

This is also depicted in Figure 1.
2.2.3 Tournament or Fan-In Tree Barrier Synchronizations

In 1988 Debra Hensgen, Raphael Finkel, and Udi Manber published a brief paper which proposed two new techniques, the dissemination algorithm and tournament algorithm (Hensgen et al, 1988). The dissemination algorithm was very similar to Brook’s algorithm, except that during each phase, each processor interacted with two other processors, not one, and we shall not discuss it further here. The tournament algorithm was very influential, and is discussed below.

Figure 1. A Butterfly Barrier Synchronization for Eight Nodes
addition, they developed some new concepts related to barrier synchronizations and timing. These authors describe their algorithms in terms of processes, not processors. Since a process is essentially an abstraction of a program executing a virtual processor, we need not concern ourselves with the conceptual differences. Since a program may need to synchronize an arbitrary number of times, a specific execution of a barrier synchronization was referred to as an episode, and the time for pre-barrier operations was referred to as an epoch.

The tournament algorithm also used flags in shared memory and busy waiting, but reduced the total number of operations by using a tournament tree pattern. Just as before, a system with \( N \) equal to an integral power of two processes, requires \( x = \log_2 N \) iterations. In this case however, a binary tree is used and only \((N-1)\) shared memory operations are needed to ensure that all processes have reached the synchronization point. During iteration \( J \) from 1 to \( \log_2 N \),

1. if a process number has a one in its \( 2^{(J-1)} \) place, then it notifies the process with the same number except with one in its \( 2^{(J-1)} \) place, and wait for barrier completion;

2. if a process has a number with a zero in its \( 2^{(J-1)} \) place, but is not yet waiting for barrier completion, then it receives notification from another process.
Ultimately only process zero receives $\log_2 N$ notifications and the barrier synchronization is completed. Figure 2 shows an example of the fan-in tree, but not the notification of barrier synchronization completion, which requires another tree traversal.

2.2.4 NYU Ultra-Computer’s Fetch-And-Add

The NYU Ultra-Computer supported an atomic fetch and add operation, which could be used to implement a barrier synchronization (Gottlieb et al, 1983). This operation incremented
the value stored at a location in memory, and returned the pre-update value. This could be used to implement a barrier by using a counter located in memory, which is initialized to zero. Each processor then increments the value and busy waits on the counter until it reaches the value of the number of processors participating in the barrier, at which time the barrier has taken place.

One of the unique features of the Ultra-Computer’s fetch-and-add operation was that it was supported by the interconnection network. Specifically, multiple messages to perform fetch-and-add at a common memory location could be combined into a single request in the network, summing the values to be added to the location, and requesting the pre-operative value at the location be returned to multiple processors. This allowed greater concurrency in the accumulation, since N increments could potentially be done in the same time as one increment. It also reduces contention.

2.2.5 Connection Machines Two and Five

Both the Connection Machine Two and the Connection Machine Five used message delivery to perform synchronization. Essentially the processors sent pre-barrier packets, and then waited for all of those packets to be delivered, at which time the barrier was assumed to have been completed.

The Connection Machine Two used a dedicated empty bit on every switch, which had the value one when the switch contained no undelivered packets (Hillis, 1985). These bits were ANDed together by dedicated hardware. The processors could then busy wait on this value, and
when it became one, then all packets had been delivered, ensuring all outstanding operations completed.

The Connection Machine Five used a global counter which kept track of the number of outstanding packets (Leiserson et al, 1992). The counter was incremented when a packet was sent, and decremented when one was received. Processors sent their pre-barrier packets, and then busy waited on the global counter. When the counter reached zero, then the barrier had been reached and the processors could proceed.

We note that these mechanisms can, at most, support a single partial barrier at any given time.

2.3 Packet Switching in the Abstract

In order to describe Integrated Network Barriers, packet switched interconnection networks, switch design, and routing must be clearly defined.

2.3.1 Flits, Packets, and Queues

A multi-processor is a computer with more than one processor (Liess, 1995). Each processor has a unique identifier, called a processor number, which allows its location to be determined relative to other parts of the network. Let the uppercase P denote the set of all processors.

An interconnection network enables communication between the processors in a multi-processor. A packet switched interconnection network routes packets between the processors.
A packet is the smallest routable unit of information (data and/or control), and must contain an address field specifying its destination (Culler et al, 1999). A packet typically contains other control information in its header, such as a length field. These details may be ignored for our discussion. A phit is the unit of transfer based on the transfer media. A flit, or flow control unit, (sometimes called a flow control digit) is the smallest unit of information the transfer of which may be accepted or rejected. The sizes of phits and flits are obviously dependent on the switching technology, and flits are typically more than 1 phit in length. The sizes of packets, as well as the capacities of queues, shall both be assumed to be a discrete number of flits.

While routing through the network, packets are stored in queues. These queues resemble the abstract data type of the same name, in that packets are removed from them (de-queued) in the same order in which they are inserted (en-queued). Unlike the abstract data structures, these queues have a finite maximum capacity, and can be full and hence block, in which case they are unable to receive more packets until some have been removed. The front of the queue from which packets are removed is called the head of the queue, and the end where they are inserted in called the tail of the queue. In the literature about routing, a queue is often drawn as depicted in Figure 3.

We shall make three further assumptions about the operations on queues.

1. A queue may have at most one flit to be inserted per cycle, and at most one flit can be removed per cycle.
2. An individual flit must be completely inserted before it can begin to be removed.
3. A queue may simultaneously insert one flit while removing a distinct, previously inserted flit.
Definition. Virtual Cut Through Switching - a packet switching technique in which only the head of a packet may block (Tannenbaum, 1999).

Let the uppercase letter $Q$ denote the set of all queues in the network. There are two special subsets of $Q$ which deserve mentioning. The source queues are those queues by which packets enter the network, and to which no queue in the network may transfer packets. Let $Q_S$ denote the set of all source queues. Similarly, the destination queues are those queues by which packets leave the network, and from which no packet is transferred to another queue in the network. Let $Q_D$ denote the set of all destination queues. Each source or destination queue is
associated with a unique processor, which can be viewed as a black box which acts as the ultimate producer and consumer of packets. When we speak of a packet being addressed to a processor, it is actually addressed to a destination queue associated with that processor. Also, when we speak of a packet being from a processor, its point of origin within the network is actually a source queue associated with that processor.

Figure 4 demonstrates this arrangement, in which we show the source and destination queues associated with each processor. While each processor must have at least one pair of such queues, they will in general have multiple pairs of queues.

The connections between queues shall be called links. While at this time we would assume that this is some type of conductor, typically copper or aluminum with the electronics to drive them, this will be deliberately left abstract. A Cycle shall be defined as the unit of time required to transfer (or refuse to transfer) a flit of information across a link. Hence a packet of N flits in size would require N cycles to transfer from one queue to another. For our purposes the size of a flit and the length of a cycle shall be deliberately left abstract, rather than give artificial, and eventually laughably dated, values in fractions of seconds or multiples of bytes.
Figure 4. Schematic of Processor, Queue, and Interconnection

2.4 Memory Operation Completion

The connection machine techniques demonstrated how packet or message delivery, which is essentially ensuring that operations have completed their effect on memory, could be used to implement a barrier. There are two issues associated with this. First, how do we know when every processor has initiated its barrier synchronization? The Connection Machine Two was SIMD, but the Connection Machine Five was not. Secondly, synchronization also means
emptying out the network of pre-barrier packets. We now need to examine operation completion outside of the context of a barrier, before we can integrate it into a barrier.

The traditional approach to ensure operation completion is to require that all messages be acknowledged, by having the destination processor or memory explicitly return an acknowledgement message to the original sender.

2.4.1 Mohan Ahuja’s Flush Primitives

In 1990, Mohan Ahuja described a family of flush operations for use on distributed systems, which guaranteed order of message arrival relative to a flush operation, even when used over a channel which did not guarantee first-in first-out arrival order of messages (Ahuja, 1990). A channel which can support flush operations was called an F-Channel. Channels are assumed to be uni-directional links between two specific processes, and hence those being used by a process were divided into two types: incoming channel and outgoing channels. A flush operation transmits a special message from one process to another over an F-channel, which partially orders messages sent over that channel. A flush operation must have one or more of the following properties: (a) a message sent before the flush message must arrive before the message associated with the flush operation, or (b) a message sent after the flush message must arrive at its destination after the flush operation. This results in three types of flush operations.

1. A flush operation which enforces pre-flush arrival before the flush message but does not guarantee arrival of post-flush messages after the flush message, is called a forward-flush.
2. Similarly, a flush operation which ensures that post-flush messages arrive after the flush message, but makes no guarantees about the arrival of pre-flush messages before the flush message is called a **backward-flush**.

3. A flush operation which segregates pre-flush and post-flush messages is said to be a **two-way-flush**.

While Ahuja’s paper was not concerned with barrier synchronization per se, clearly it has direct applications for the ordering of message arrival.

### 2.4.2 Renade’s Network Merging Algorithm

Renade’s merging algorithm runs on a dancehall network, in which processors are located on one side of the network, and memories are located on the opposite end (Renade, 1989). The intermediate switches have two input ports, two output ports, and internal buffering which can store a packet. A packet is generated by a processor and routed through a series of switches until it reaches the memory containing the location to be accessed. Packets to the same location pass through at least one switch in common. Hence request to access a common location can be combined within the switches where their paths merge.

Renade added the further restriction that within a given phase, memory was only accessed in increasing address order. This was needed so that the switch could merge accesses to common locations, similar to the merged fetch-and-adds of the NYU Ultra-Computer. In order to ensure operation completion the following protocol is followed.
1. Packets requesting the access of a given location are held at the switch until either they are combined with another request for the same location from the other input port, or the arrival of a packet containing a request to access a higher location arrives, which guarantees that no other request for this location will arrive.

2. When a packet containing a request to access a memory location is forwarded to one output port, a second “ghost packet” containing the address but no actual request is forwarded out of the other port, which notifies switches downstream that no more requests for locations below this address will arrive from this switch.

2.4.3 Mahapatra and Dutt’s Termination Detection Algorithm

In (Mahapatra and Dutt, to appear), an algorithm is developed to detect when a program running on multiple processing elements, (PEs), and communicating via messages, has completed. This is called the termination detection problem. The termination detection is similar to a barrier, and addresses the issues of synchronization and operation completion with arbitrary communications.

They assumed only connectedness of the communication network. This assumption means that the processing elements can be logically connected in a termination tree, which is a spanning tree embedded within the network. Furthermore, the root of the termination tree is assumed to be the initiator of the computation, and also responsible for coordinating termination detection.
The non-termination portion of the program is referred to as the **primary calculation**. A PE which is executing part of the primary computation is said to be **busy**, otherwise it is **idle**.

Messages used in the primary calculation are referred to as **primary messages**. Primary messages need no be exclusively routed between PEs which are adjacent in the termination tree, and require that an **acknowledgement message** be sent back to the messages originator when it has arrived at its destination. A PE which has sent one or more messages for which it has not yet received an acknowledgment is said to be **loaded**, while one with no un-acknowledged messages is said to be **free**.

The computation and communications related to termination detection are called the **secondary calculation** and **secondary messages** respectively. Secondary messages include **stop messages**, which report that a set of PEs are out of work to do, and **resume messages**, which cancel report messages, and termination messages, which are broadcast to all PEs when termination has been detected.

A processing element is assumed to initially be busy, while executing some portion of the primary calculation. When it becomes idle, is free of unacknowledged packets, and has received stop messages from all of its children in the termination tree, it may then send a stop message to its parent in the termination tree. This is considered a transition from being **active** to **inactive**.

However, an inactive PE may receive a primary message, which it must process, forcing it to return to active status. When it makes that transition, it sends a resume message to its parent.
in the termination tree, and the resume message is then forwarded up the tree until it reaches the first PE which is not inactive. This returns all PEs on the path to active status, until they receive stop messages again. Then this highest active PE sends an acknowledge message back down the tree to the original recipient of the primary message, who then in turn forward it to the original sender of the primary message. When this node again meets the criteria to become inactive, it does so again and sends another stop message to its parent in the termination tree.

After stop messages have reached to root of the termination tree, then it has detected that the primary calculation has terminated, and then broadcasts termination messages to all PEs.

The termination detection problem is more general than a barrier, since it can allow a PE to return to an active state, whereas barriers cannot be rolled back. The termination detection algorithm requires explicit acknowledgement of primary messages, and hence does not allow pipelining. Unlike INBs, the termination detection algorithm as presented is intended to be run once, vis-à-vis INBs which intended to allow multiple instances, which can themselves be pipelined. It would be interesting to investigate the application of INBs to the termination detection problem; however that is beyond the scope of this thesis.

2.5 The Route Step Function and Routes

Now that the general concepts of queues and switches have been described, we turn our attention to routing and queue dependence. Given a packet at the head of an input queue, the control mechanism must determine to which output queue to transfer the packet. Collectively the sequence of these decisions will move a packet to ultimately reach its final destination. This
control process is described by the route step function, whose parameters are the packet’s current location and ultimate destination, and which returns the set of queues into which the packet may be transferred. The routing step function depends upon how queues are connected, and the routing algorithm being used.

Definition. Route Step Function. $\sigma : Q \times Q \rightarrow P(Q)$. Specifically, for a given queue $x$, where $x \in Q$, and a given queue $y$, where $y \in Q$, the route step function, denoted $\sigma (x, y)$, where $\sigma (x, y) \subseteq Q$, specifies the set of queues to which a packet at the head of queue $x$ may be directly transferred in order to ultimately route to $y$.

In a deterministic routing scheme, the route step function always produces sets of no more than one element, and furthermore if there is no legal route from $x$ to $y$, $\sigma (x, y)$ will be empty. A route then shall be considered an ordered sequence of queues which a packet would traverse, based on the values of the route step function.

Definition. Route: a route from queue $q_1$ to queue $q_n$, shall be defined as a sequence of queues, $[q_1, q_2, \ldots, q_{n-1}, q_n]$, in which $\forall x \in [1, n - 1]$, $q_x \in \sigma (q_{(x-1)}, q_n)$. A route is said to be from its initial queue $q_1$ and to its final queue $q_n$.

The term routing algorithm refers to the overall scheme which determines routes through the network as a whole (Tanenbaum, 1999). The route step function represents the restrictions on the queue level decision of selecting the next queue a packet should route to,
essentially enforcing the routing algorithm at the lowest possible level. In a deterministic routing scheme, the step function determines the route.

From the route step function, we define two sets which are fundamental to integrated network barriers. These are the successor set and predecessor set, and they are related to a set of routes. For a given queue, its successor set is the set of all queues to which it may route packets. Similarly, the predecessor set is the set of queues which may route packets to the queue in question. These are more formally defined as follows.

**Definition. Successor Set.** For a given queue \( x \) in \( Q \), the successor set of \( x \), denoted \( \text{Succ}(x) \), is the set of all queues to which a packet may be transferred from \( x \). Formally, this is denoted as:

\[
\text{Succ}(x) = \bigcup_{y \in Q} \sigma(x, y).
\]

**Definition. Predecessor Set.** For a given queue \( x \) in \( Q \), the successor set of \( x \), denoted \( \text{Pred}(x) \), is the set of all queues which may directly transfer a packet to \( x \). Formally, this is denoted as:

\[
\text{Pred}(x) = \{ \forall y \in Q \mid x \in \text{Succ}(y) \}.
\]

We will next consider a very simple queue topology and define its attendant routing information.
Example. Consider the arrangement of queues in Figure 5. Let us assume that the queues depicted are the only ones in the network, and that the arrows show all legal transfers between queues. Under those circumstances, the following conditions apply. The successor and predecessor sets for each individual queue are summarized in Table I. In addition, the route step function is summarized in Table II.
Table I. Successor and Predecessor Sets for the Network in Figure 5.

<table>
<thead>
<tr>
<th>Queue X</th>
<th>a</th>
<th>b</th>
<th>c</th>
<th>d</th>
<th>e</th>
<th>f</th>
</tr>
</thead>
<tbody>
<tr>
<td>Succ(X)</td>
<td>{c, d}</td>
<td>{d}</td>
<td>{e}</td>
<td>{f}</td>
<td>{}</td>
<td>{}</td>
</tr>
<tr>
<td>Pred(X)</td>
<td>{}</td>
<td>{}</td>
<td>{a}</td>
<td>{a, b}</td>
<td>{c}</td>
<td>{d}</td>
</tr>
</tbody>
</table>

Table II. Route Step Function Values for the Network in Figure 5.
Using successor and predecessor sets allows formal definition of some traits of source and destination queues.

Lemma: \( q \in Q_s \Rightarrow \text{Pred}(q) = \{\} \)

Proof: By the definition of source queues, if \( q \) is in \( Q_s \), then there are no queues which may route into \( q \). Hence \( q \) may have no predecessors.

Lemma: \( q \in Q_d \Rightarrow \text{Succ}(q) = \{\} \)

Proof: By the definition of destination queues, if \( q \) is in \( Q_d \), then there are no queues to which it may packets. Hence \( q \) can have no successors.

2.6 Queue Dependence and Queue Dependence Graphs

Queue dependence connects together the static structure of the interconnection network (the queues) and routing (the dependencies). It is used primarily to verify that routing schemes are deadlock free.

Definition. Queue Dependency – A Queue A is said to depend upon queue B if packets can move from queue B to queue A while routing on some valid route (Pifare et al, 1994).

Queue Dependence depends upon the network geometry and the routing algorithm being used.

Definition. Queue Dependency Graph (QDG)- a directed graph \( G=(V,E) \), where

- each vertex in \( V \) corresponds to a queue in the switching system, and
• each edge \((u, v)\) indicates that a packet may route from the queue corresponding to vertex 
  \(u\) to the queue corresponding to vertex \(v\) (Pifare et al, 1994).

The digraph in Figure 5, which shows the flow of packets, is in fact a queue dependency 
graph.

Lemma. An acyclic Queue Dependency Graph corresponds to a deadlock free routing scheme 
(Pifare et al, 1994).

The Queue Dependency graph is related to the Channel dependency graph, (CDG) but 
provides more information (Dally and Seitz, 1987), (Stamatopoulos, 1996). Specifically, channel 
dependency graphs look at the links between switches, and hence present courser grain 
information on the routing algorithm (Dally and Seitz, 1987). If each queue is viewed as a 
switch, then the channel dependency is the queue dependency graph. However, as we shall see, a 
switch may have multiple queues incident upon the end of a link. Unlike the CDG, the QDG 
provides all the necessary properties of ordering and routing on which Integrated Barrier 
Networks are based. See (Stamatopoulos, 1996) for further discussion of this point.

2.7 Plies and Virtual Interconnection Networks

Having defined the concepts of a queue, queue dependency graphs, and routes, we next 
create a higher level of abstraction which encapsulates these concepts. Specifically routes will be 
used to construct the concept of a ply and a virtual interconnection network.
The term ply was first used by Jerry Stamatopoulos in his dissertation (Stamatopoulos, 1996); however his definition more closely matches our definition of Virtual Interconnection network (VIN) below.

Our concept of ply only requires that each route be totally contained within a single ply. We shall call this property the total embedding property. We do not to require that an individual ply contain a route between all possible source–destination pairs, as does (Stamatopoulos, 1996).

**Definition. Ply** – the union of a set of routes, which has the property that any route which traverses a queue within a ply, exclusively traverses queues within that ply.

Since a single ply is not required to contain routes between all valid source-destination pairs, we need a conceptual equivalent of a sub-network which does. We shall call a collection of plies, which has a route between every directed pair of processors, a Virtual Interconnection Network. An actual interconnection network may contain an arbitrary number of virtual interconnection networks, each of which is logically independent of the others.

**Definition. Physical Ply** - a ply which contains the closure of all queues which are either predecessor or successors of some queue.

**Definition. (VIN) Virtual Interconnection Networks** - a minimal collection of plies which contains at least one route between each ordered pair of processors. (Alternatively, this would be
between each valid source-destination pairs if some sources are not allow to send to some
destinations.)

The plies discussed in (Stamatopoulos, 1996) are in fact both virtual interconnection
networks, and single ply. Furthermore, the routes in (McShane, 1996a), while not described as
such, are generally grouped into plies.

2.8 Interconnection Networks

In general, we focus on the queues and their interconnection, and their analysis via the
QDG. However, it is sometimes useful to view the system at the network topology level.

A packet switched interconnection network contains the following elements.

1. **Sources** and **drains** of packets. Typically these would be processors which send
   packets to each other. For our purposes, a processor may be thought of as both a
   source and drain of packets.

2. **Switches** route packets between the sources and drains of packets.

3. **Links** act as the connections between the switches, sources, and drains of packets.

The logical, but, not necessarily spatial, arrangement of the switches and links is
commonly called to the **network topology** (Zargham, 1996). The term topology is used since the
logical arrangement may be in a shape not physically possible in three dimensions, such as a
Klein bottle or five dimensional hypercube. Note that in this thesis, we are almost always
focused on **queue topology**, and not the coarser grained network topology. The network
topology focuses on the arrangement of nodes (switches) which are themselves sets of queues. Some of the more common network topologies: Hypercube, N-Dimensional Mesh, Iliac, Torus, and the Omega network (Zargham, 1996). Please see Figure 6 to see a Two-Dimensional Mesh topology.

A switch contains two types of queues: **input queues** which receive packets from outside sources, and **output queues** into which packets bound for external destinations are transferred. A very simple switch for a two dimension switch is depicted in Figure 7. Note that from the point of view of the switch, the processor is simply another external source and destination for packets.

A common optimization in switch design is to split input queues into separate **turn queues** and **rifle queues**. The rifle queue only accepts that packets which are not changing direction and pass through “like a rifle shot.” In contrast, the turn queue accepts the packets which are changing direction at this switch. This can give greater concurrency and prevents packets which are not turning from being forced to wait behind ones that are turning. This arrangement is similar to an overpass on a highway.

**Definition. Dimension Routing** – a deadlock free routing scheme for n-dimensional meshes, in which packets are routed minimally to the correct coordinate in each dimension in turn, in a fixed order (Tanenbaum, 1999). Dimension routing in grids is also called **metropolis** routing, or **Manhattan** Routing.
Figure 6. A Two Dimensional Mesh Network Topology
2.9 Integrated Network Barriers

Informally, Integrated Network Barriers, sometimes referred to simply as barriers, are a refinement of barrier synchronization. They essentially combine the features of barrier synchronization with the operation completion aspects of Ahuja’s Flush primitives. This makes them ideal for synchronization in packet switched interconnection networks, since they reduce overhead and are pipelineable.
Definition. **Integrated Network Barrier** – (INB) a barrier implementation in which the completion of pending memory operations is integrated with barrier synchronization (Solworth and Stamatopoulos, 1993; Birk et al, 1989).

2.9.1 The Integrated Network Barriers of Birk, Gibbons, Sanz, and Soroker

While the first published use of the term Integrated Network Barrier would not be coined for another four years the earliest proposed barrier implementation using barrier markers and wave fronts of which the author is aware was developed by IBM and described in (Birk et al, 1989). In a short paper in ICPP, these barriers were described for use on multistage dancehall networks, where all of the processors were located at one side of the network of switches, and all of the memory modules were located on the opposite end of the network (Birk et al, 1991). Please see Figure 8 to see an example of such a network. A more comprehensive technical report described conditions to implement an INB on a variety of networks, including a hypercube and mesh network (Birk et al, 1989).

The terminology used here is taken from Stamatopoulos’s thesis (Stamatopoulos, 1996). In this implementation special packets, normally called barrier markers, were inserted into each input queue into the network by each processor (Birk et al, 1989). Barrier markers act as dividers within the network, segregating packets sent before, called **pre-barrier** packets, from those sent after the barrier, called **post-barrier** packets. In this sense they act like the flush messages in Ahuja’s Flush primitives.
In order to perform a barrier using this protocol, all a processor (source of packets) needs to do is:

1. complete pre-barrier operations and inject pre-barrier packets,
2. insert a barrier marker into its queue into the network,
3. start its post barrier operations and continue injecting its post barrier packets (Birk et al, 1989).

The monotonic routing nature of the dancehall switch made coordination of the barrier markers relatively simple to understand. Within each switch, there are two input queues and two output queues. When barrier markers reached the heads of both of the input queues, then clearly no more pre-barrier packets can arrive (Birk et al, 1989). Hence at that time, new barrier markers were inserted into each output queue, and then the markers at the heads of the input queues were removed. This advancing wall of barrier markers thus progresses from queue to queue, and hence was referred to as a wave. At the memories, the arrival of a barrier marker indicates that all pre-barrier packets have arrived, and that in the future, only post barrier packets will arrive.

Consider the network in Figure 8, which depicts a partially completed barrier on an omega network. Processors are depicted as circles on the left, and memories as hexagons on the left. Due to space considerations, queues within the switches are represented without internal lines. A processor, memory, or queue which has been swept by a barrier marker is drawn with a diagonal line through it, and one with a barrier marker at its head, being on the wave front, has two diagonals drawn through it making an “X”. This network has completed the barrier as far as possible until the third processor from the top inserts a barrier marker. At that time the third
switch from the top will have markers at the heads of both of its input queues, and will be able to
insert markers into its output queues, and then remove the markers from its input queues. The
markers will be forwarded across the links and into the two switches at the top of the second
column, which will each then repeat the process. After those last two switches in the second
column have completed propagating the wave, all four switches in the second column will follow
suite by inserting markers into their output queues, deleting the markers at the heads of their
input queues, and forwarding the markers to the memory modules in the rightmost column.

Birk, Gibbons, Sanz, and Soroker then generalize this INB technique for other network
generics, which may require the use of multiple waves to sweep all possible routes through the
network’s switches. While their solution is straightforward, clean, and optimal for an Omega
network, their solutions for other networks were generally not as efficient as their technique for
the Omega network. Their correctness properties were existential, and therefore did not give rise
to an implementation. In particular, their solution for the most studied topology, the two-
dimensional mesh, was particularly inefficient. For their sample implementation on a two-
dimensional mesh, they suggested using two waves which each swept the network twice, once
from a corner to the opposite corner and back to the initial corner, for a total of four traversals of
the network. Please see Figure 11 for an example of a corner wave.

In the event of a partial barrier, processors which were not participating in the barrier
were still required to insert barrier messages, although these were special barrier messages which
could be overtaken by data packets, or could overtake data packets. The final wave would be of
markers which were unable to either be overtaken by data packets, or to overtake data packets.
While not providing optimal solutions for non-omega networks, Birk, Gibbons, Sanz, and Soroker did in fact give the three necessary and sufficient conditions for correct operation of an integrated network barrier. These conditions are:
1. **Flushing Property** – A barrier message traverses a link only after all pre-barrier packets.

2. **Restraining Property** – A barrier message traverses a link before all post-barrier packets.

3. **Clear Route Property** – The path of a pre-barrier packet is always clear of post-barrier packets.

2.9.2 **Stamatopoulos and Solworth’s Edge Wave Integrated Network Barriers**

Over the period from 1993 to 1996, Jerry Stamatopoulos and Jon A. Solworth developed an integrated network barrier technique which used the queue dependency graph. When implemented in n-dimensional mesh, it used a single wave (Solworth and Stamatopoulos, 1993, 1994, 1995, 1996, 2002; Stamatopoulos, 1996). Ironically, Jorge Sanz was an author in both the first INB paper (Birk et al, 1989) and the first paper of queue dependency graphs (Pifare, et al 1994), but apparently did not see the application of Queue Dependency Graphs to Integrated Network Barriers. The use of QDGs revealed that what mattered was the markers swept the queues in order of routing, not based on their relative position in the network geometry. Hence this gave a standardized methodology for creating a set of waves which traverse the queues and ensure that pre-barrier and post-barrier packets were kept segregated.

**Edge wave integrated network barriers**, so named because on the grid the barriers emanate from the edges, use a single ply per VIN. They were developed on a mesh network with dimension order (metropolis) routing, which has an acyclic queue dependency graph (Solworth and Stamatopoulos, 1993). Essentially, since dimension ordering resolves routing one dimension at a time in a fixed order, the queue dependency graph cannot contain a cycle. By using barrier
waves which sweep the network from one edge to the opposite, in routing order, all queues are swept free of pre-barrier packets (Solworth and Stamatopoulos, 1993, 1994, 1995, 1996, 2002; Stamatopoulos, 1996).

The rules for barrier marker propagation are based on the successors and predecessors to a given queue. After a barrier marker reaches the head of the queue, it will be propagated to each successor queue. The propagation occurs for successor S when every predecessor of S has a barrier marker at the head of its queue.

This is further refined by the use of barrier marker counters. To keep track of the number of barrier markers which had been inserted into and removed from a queue, barrier marker counters are associated with each queue (Solworth and Stamatopoulos, 1993). Associated with the head of each queue, a head barrier marker counter keeps count of the number of barrier markers which had been removed from the queues head (Solworth and Stamatopoulos, 1993). In addition, each queue has a tail barrier marker counter at its tail, which counts the number of barrier counters which have been en-queued at the queues tail. When all predecessors of a queue have head barrier marker counter values than the queues tail barrier marker counter, then (1) a barrier marker is inserted into the queue, and (2) the queues tail barrier marker counter is incremented. Packets are then only allowed to transfer between queues if the originating queues head barrier marker counter has the same value as the receiving queues tail marker counter. This allows the markers to be immediately de-queued when they reached the head of the queue, freeing up the space, and also this can allow greater concurrency at splits.
Example. Consider the arrangement of queues in Figure 9. If a barrier marker is at the head of queue c, but not at the head of queue a. Using the rules from (Birk et al, 1989) the barrier wave cannot propagate into either b or d until there are markers at the head of both a and c. By using barrier marker counters, the barrier marker can be immediately propagated into queue d. In addition, if a packet behind the marker in queue c should route to queue d, it may do so, although if a packet to route to queue b is behind the marker, then it and all packets behind it must still wait.

Example. Consider a simple two dimensional mesh topology, with a single ply VIN, with a single input and output queue from each neighbor in the grid and from the processor, and upon which dimension order routing is implemented routing on Y first, then X. A generic, simple switch is shown in Figure 7 above.
• Consider a switch on the lower edge (minimal \( y \) ordinate) of the network, where the input queue from the processor has just had a barrier marker propagate to its head. Since the output queue in the \( y \) positive direction has only the input queue from the processor as a predecessor, the wave may immediately propagate in the \( y \) positive direction. A node on the upper edge of the network with a barrier marker at the head of its input queue from the processor may similarly propagate the wave in the \( y \) negative direction.

• Consider a switch on the left hand side (minimal \( x \) ordinate) of the network, where the input queue from the processor has just had a barrier marker propagate to its head. It may not propagate the barrier in the \( x \) positive direction until all of the predecessors of the output queue in the \( x \) positive directions have been swept by a wave for this barrier. This is testing by determining if their head barrier marker counters are all greater than the tail barrier marker counter of the output queue in the \( x \) positive direction. In this case, the predecessors are: the input queues from (1) the processor, (2) the \( y \)-positive direction, (3) the \( y \) negative direction. At such point in time when all these predecessors have been swept, then the barrier wave may be propagated. A node on the rightmost (maximal \( x \) ordinate) edge will behave symmetrically with respect to propagating the wave in the \( x \) negative direction.

• A switch in the center of the network will forward a wave into an output queue only after all of that queue’s predecessors have been swept, as described above. Output queues in the directions have at most two predecessors: the input queue from the processor and the input queue from the opposite \( y \) direction. Output queues in the \( x \) direction can have four predecessors, being the input queue from the processor, the input queues from both \( y \)-directions, and the input queue from the opposite \( x \) –direction.
• All switches forward the wave to the processor via the processors output queue only after all predecessors of the output queue to the processor have been swept. This requires all four input queues from the four cardinal directions.

2.9.3 Barriers and Congestion Control

The primary focus of Solworth and Stamatopoulos’s work was to use INBs as a traffic congestion control device in the networks. The time to complete the barrier, called barrier latency, threatened to undermine performance and counteract the benefits for short haul traffic, which packets only traveled a few links. In order to hide barrier latency, Stamatopoulos used four virtual interconnection networks, and cyclically alternated active VIN, upon which the processors sent data packets, followed by a barrier to flush the VIN before switching to another VIN. This required the processor to coordinate barriers, since pre-barrier packets on one VIN may arrive at the processor after post barrier packets on another. (Stamatopoulos used the term ply instead of VIN, since his VINs consisted of a single ply each.) To see this scheme graphically, consider the Gantt chart in Figure 10. Please note that the barrier frequency is based on the packet and barrier latencies and so that barriers would complete on a given VIN before post barrier data packets were injected on that VIN. Otherwise a packet might become trapped behind the wave and suffer large latencies.
2.9.4 McShane’s Perfectly Pipelinable Barriers and Corner Waves

Eric McShane wrote two unpublished papers on barriers in 1996. In the first of these papers, he proposed a variation on integrated network barriers which he called perfectly pipelineable barrier waves. Firstly, he noted that a single barrier could be composed of waves propagating on multiple plies, which he referred to as “virtualizing the buffers” (McShane, 1996a). Secondly, he noted that by reducing the links in the queue dependence graph within a ply, he could prune the graph in such a way as to traverse all queues in the ply with a single wave of barrier markers. In particular, he noted that by only allowing packets to move in one direction per dimension, a wave starting from one corner and traversing to the opposite corner of a two-dimensional mesh could sweep all queues traversed by routes of packets moving in the same directions as the wave. Furthermore, since these waves moved on separate plies, they did not have to wait for each other, but rather run independently in their own plies, unlike the previous two dimensional mesh barriers by Stamatopoulos and Solworth.

![Figure 10. Gantt Chart of Staggered Data and Barriers on a Four VIN network](image-url)
In addition, pairs of waves from opposite corners used disjoint sets of queues and link, and could theoretically be embedded in the same ply. For example, the northeast (X positive Y positive) and southwest (X negative Y negative) waves could run on one ply, and northwest (X negative Y positive) and southeast (X positive Y negative) could run on another. Packets could then be routed on the one of the two plies which had a wave which swept its route, and the whole network flushed by a barrier using all four waves. Please see Figure 12 which shows a sample northeast (X Positive Y Positive) wave on a four by four, two dimensional mesh network.

While McShane’s solution for speeding up the barrier by maximizing independence of the waves was elegant, his method for partial barriers was complex and required significant overhead to coordinate the different plies. Essentially he would use markers from the waves pre-existing global barriers which would be marked as being also used in the partial barrier, by the addition of a “done” marker as they entered the region to be synchronized (McShane, 1996b). In addition, the waves used would be selected ahead of time from different pre-scheduled global barriers, based on how far the edge of the region was from the corner in which the wave originated, in order to ensure that the waves entered the region at the same time.

McShane’s partial barriers have a number of issues.

1. It would be extremely difficult to pre-schedule the barriers ahead of time, due to dynamic changes in network latency.
2. If the global barriers arrive too early, then the rest of the network is delayed while the partial barrier completes.

3. If the global barriers arrive too late, then partial barrier is held up.

It is desirable to have the partial barrier have minimal, preferably no, interaction with the rest of the network.
2.9.5 An Extended Example: A Linear Network of Three Nodes

The following is an extended example to demonstrate how separating a VIN into two plies can allow waves to propagate independently of each other, as well as create greater routing opportunities.

In order to tie together the concepts of VINs, plies, and barrier waves, we shall demonstrate the simplest possible network with different paths. A network with a single node has no need for a switch, and one with two nodes has two channels communicating in each direction, and hence lacks any real opportunity to investigate switching. Assume that we have a network of three nodes connected in a straight line. These nodes will have processors numbered 0 through 2, and we shall assume that the spatial ordering of the scheme will be named after the X axis, hence nodes 1 and 2 are to the X positive direction of node 0, and node 1 is in the X negative direction of node 2. These two directions shall be abbreviated “X+” for X positive and “X-” for X negative in diagrams. Furthermore, queues which are associated with the processor shall be marked as “Proc”. In addition, a queue polarity are considered to be Input (abbreviated In) if they receive packets being routed into the switch, and output (abbreviated Out) if they are used to route packets out of the switch to another switch or to the processor.

As is generally the case, the processor input queues are source queues, and the processor output queues are destination queues. Hence a queue may be uniquely specified by a triple listing: its node number, direction, and polarity; followed by an optional dash and ply number if a queue position is duplicated between two plies, as may sometimes be required in a multi-ply VIN. For example (1, X-, Out) denotes the queue on node number 1, in the X- side, used to route
packets out. In addition, (1, Proc, Out)-1 denotes an output queue to the processor on node 1 in ply number 1 in a multiple ply VIN. As per our standard assumptions, all nodes may generate packets addressed to any other node in the network, and may also receive packets from any other node in the network. The queue dependency diagram for a single ply VIN version of this network is show in Figure 12.

Now consider a simple barrier, using the techniques of (Solworth and Stamatopoulos, 1993) on this 1 ply VIN. Please recall that barrier markers are placeholders within the queues

![Figure 12. One Dimensional Array in a 1 Ply VIN](image-url)
which separate pre-barrier and post-barrier packets, and their insertion or removal from a queue indicates the presence of the wave front. As the processor on node 0 enters barrier, it inserts a barrier marker, into \((0, \text{Proc, In})\) after the last of its pre-barrier packets. Since the queues always route the packets in FIFO order, when the marker arrives at the head of \((0, \text{Proc, In})\), all pre-barrier packets have already been routed into \((0, \text{X+, Out})\). The packets and marker then incrementally proceed in order through \((0, \text{X+, Out})\) into \((1, \text{X-, In})\). When the marker eventually reaches the head of \((1, \text{X-, In})\) and is de-queued, a new marker cannot be inserted into \((1, \text{X+, Out})\) until a marker has also reached the head of \((1, \text{Proc, In})\), since a marker cannot be inserted into \((1, \text{X+, Out})\) until all of its predecessor queues have been traversed by a marker. Since the arrival of the marker also indicates that all pre-barrier data packets from the processor on node 1 have already been routed, the propagation of the wave into \((1, \text{X+, Out})\) guarantees that all pre-barrier packets that could route into \((1, \text{X+, Out})\) have already been routed. After a marker has been inserted into \((1, \text{X+, Out})\), the packets and markers proceed in order through \((2, \text{X-, In})\) and \((2, \text{Proc, Out})\) to the processor on Node 2. Note again that the arrival of the barrier marker at the processor on node 2 guarantees that all pre-barrier packets which would route to node 2 have already been delivered.

A barrier wave starting at Node 2 proceeds in the same manner, just in the opposite direction. It is worth noting that the two terminal nodes, on the far ends of the network, can only receive packets from one direction, and only need to receive a marker (or wave front) from that direction from which they receive traffic in order to complete. In fact, a node may have received waves and completed the barrier before it has started its own wave.
The queue (1, Proc, Out) does not receive a barrier marker until both (1, X+, In) and (1, X-, In) have had barrier markers reach their heads. Since this queue can receive packets from both directions, it requires two wave fronts, one from each direction, to insure that it has received all of the pre-barrier packets which route to its processor.

Now, consider if the VIN is partitioned into two separate plies, based on the direction of the flow of packets. Ply 0 contains the queues required to route packets in the X negative direction, and ply 1 contains the queues required to route packets in the X positive direction. In the case of node 1, duplicates of the input and output queues for the processor are needed, as node 1 must send and receive packets on both plies. Hence (1, Proc, Out)-0 represents the processor destination queue for node 1 on ply 0, which would receive packets routed to node 1 from node 2, which would have been routed in the X negative direction. This restructuring of the network is depicted in Figure 13.

Upon inspection of Figure 13 it should be obvious that a barrier must be implemented by two separate waves, one per ply, following the conventions of (McShane, 1996). Furthermore, since the two plies are essentially independent, waves do not interact within the switches, but rather must complete separately at the processors. At the processors, the waves’ arrivals are coordinated for barrier completion.
Now consider a simple barrier on this newly re-configured network. As the processor on node 0 enters the barrier, it inserts a barrier marker, into (0, Proc, In) on ply 1 after the last of its pre-barrier packets. The wave in the form of a barrier marker progresses through (0, X+, Out) and (1, X-, In). From there the wave is inserted into (1, Proc, Out) -1, and the wave front arrives

Figure 13. One Dimensional Array in a 2 Ply VIN
at processor 1. At this time, both node 0 and node 1 have entered the barrier on ply 1, independent of the action on ply 0. However, the wave may not propagate to (1, X+, Out) until a marker has also been received from (1, Proc, In) -1. When the processor on node 1 has entered the barrier and inserted a barrier marker into (1, Proc, In)-1, and it has reached the head of the queue, then the wave may progress through (1, X+, Out), (2, X-, In), and (2, Proc, out) -1.

A similar wave would traverse ply 0 in the opposite direction. No processor would complete the barrier until it has participated in both waves. Note that node 0 and 2 each initiate a wave in the direction in which they send packets, and each receives a wave from the direction from which they receive packets. Node 1, which sends and receives packets in both directions, must insert and also receive a barrier marker on each ply.
In this chapter, a set of techniques for efficient implementing partial integrated network barriers for a packet switched interconnection network is developed. A partial barrier is a barrier in which only a subset of the processors participate. We shall assume that the partial barrier takes place over routes contained in some number of plies, all of which are contained in a single virtual interconnection network. These operations are independent of other virtual interconnections in the actual network. Furthermore, the techniques given here are general enough that they can be used with the barrier protocols in (Birk et al, 1989), (Solworth and Stamatopoulos, 1993), or (McShane, 1996a).

One of the contributions of this work is a very general notion of ply. This is in contrast with the works of both Stamatopoulos and McShane, in which plies always contained a complete set of routes to or from every processor. In particular, this means that the plies discussed here do not decompose the interconnection network.

Rather, our collection of plies forms a lattice, whose nodes are sets of routes, whose dominance relation $\geq$ is the superset $\supseteq$ on routes, whose top is the set of all routes, ($\top = R$), where $R$ is the set of all routes, and whose bottom is the empty set ($\bot = \{\}$).

To implement partial barriers, the barrier will usually need only trace the possible routes of messages. This enables partial barriers to significantly less of an impact upon the network bandwidth, and also removes the need to coordinate with processors and queues not participating in the barrier. This reduces latency and overhead.
In general, we are interested in a set of routes which connects a set of sources to a set of destinations. These routes are sufficient for communication from these sources to these destinations, and if we can implement partial INBs over this set of routes, we can ensure barrier semantics.

One way of achieving these ends is to create a separate physical ply for each pair of source-destination sets. Obviously, for P processors this would require exponential \(2^n \times 2^n = 2^{2P}\) physical plies, and hence is clearly impractical.

Rather, our approach here is to have a set of physical plies, (a VIN) in which any possible required set of routes can be selected as subsets. To achieve these ends, we need a way to mask out parts of the VINs physical plies which are not used in the required routes.

3.1 Source, Destinations, Routes, and Pickets

Consider a set of routes R’, which in general is part of one or more physical plies. Routing proceeds on R’ by the routing algorithm and no special precautions are required for routing due to our partial barriers. However, complications arise when those routes must be swept by a barrier wave. Because plies can be embedded within other plies, efficient means are needed to delineate their resources from those of a surrounding ply. This can be accomplished by isolating the queues inside R’ from those in R-R’, essentially temporarily disconnecting their queue dependency graphs from the queues outside the region.
Queues outside the region which have the potential to interact with those inside the region, but are to be ignored are marked with **pickets**. There are two types of pickets, based upon their specific functions:

1. **Entry pickets**, which prevent queues not traversed by routes in R’ (or their barrier waves) from interfering with the operations within the queues traversed by the routes.

2. **Exit Pickets**, which restrain barrier waves and their associated barrier markers from leaving the queues traversed by the routes.

Before we can proceed to more formal definitions of pickets and in turn regions, we need to define some notation. Let Q’ be the queues traversed in R’ and the queues in (Q - Q’) are all queues not traversed by the routes in R’.

### 3.1.1 Entry Pickets

An entry picket marks queues in (Q - Q’) which have successors in Q’. Such queues are predecessors in the physical ply of one or more queues within Q’. They are to be ignored for purposes of checking predecessors for barrier joins when a barrier wave traverses the queues in Q’. A sample entry picket marked with its conventional symbol is shown in Figure 14.

*Definition. Entry Pickets* – a queue a is marked as an entry picket when:

\[ a \in Q - Q' \land \exists b \in Q' \mid b \in \sigma(a, b) \]
3.1.2 Exit Pickets

Exit pickets are used to mark queues not in $Q'$, and hence into which the barrier wave should not propagate. Please see Figure 15 to see an example of a queue marker as an exit picket.

**Definition. Exit Pickets** – a queue $b$ is marked as an exit picket when:

$$b \in Q - Q' \land \exists a \in Q' \mid b \in \sigma(a, b)$$
3.1.3 Picket Implementation

Entry and exit pickets can be implemented with a single bit flag per picket per queue.

3.2 Regions

We shall next define a region which is relative to a set of routes \( R' \) whose queues are \( Q' \).

Let \( S' \) be the set of source queues in \( Q' \) and let \( D' \) be the set of destination queues in \( Q' \).

Moreover, let \( \text{Proc}(Q'') \) be the set of processors which are adjacent to either a source or destination queue in \( Q'' \).

*Definition.** Source Region relative to \( R' \) is \( S' = (Q' \cap Q_S) \).

*Definition.** Destination Region relative to \( R' \) is \( D'= (Q' \cap Q_D) \).
There are two types of regions which we shall consider interesting cases, classified by the relationship between sets of sources and destination processors. These types are:

- **Closed Region** – a region in which any packets generated from Proc(S’) can be routed only to processors in Proc(S’). These have: Proc(S’) = Proc(D’)

- **Open Region** – a region in which packets generated Proc(S’) may also route to a processor p not in Proc(S’). These have Proc(S’) ⊆ Proc(D’)

### 3.2.1 Closed Regions

In a closed region, Proc(S’) = Proc(D’), and hence all processors reached by the routes also participate in partial barriers within the region. Hence queues in the region have no need to interact with those outside, and hence they can effectively be isolated from the rest of the VIN. Hence an open region can be implemented by placing pickets as follows.

1. Entry pickets must be placed on each queue in Q-Q’ which has one or more successor in Q’.

2. Exit pickets must be placed on each queue in (Q-Q’) which has one or more predecessors in Q’.

Consider the omega network depicted in Figure 16. The processors, queues, and memory modules in a closed region, which have been traversed by a barrier wave in the style of (Birk et al, 1989), have been marked with a diagonal slash. Note that the queues marked with the circle bisected with a horizontal line are entry pickets and were ignored for barrier propagation. In addition, please note that the queue marked with a circle with a superimposed cross are exit
pickets, and the wave was not propagated to these queues. If these exit pickets were removed, then the wave would have propagated to all of the memory modules and the region would have been an open region. It is also worth noting that in the figure the processors and memory modules were contiguous. This is not required and the mechanisms would have worked regardless.

3.2.2 Open Regions

Open regions are no more complicated than closed regions. Of course the routes associated with open regions are more extensive and in the case of the grid extend beyond the processors in the source processors.

One may think of a closed region as operating in isolation from the rest of the network. In contrast, while an open region only synchronizes the processors within Proc(S’), it also emits packets into other part of the network, possibly the entire network. If the exit pickets were removed from the simple network in Figure 16, then the network would have an open region with four source processors and eight destination processors.
3.3 Why Are Open Regions Interesting?

Open regions are interesting because the performance cost of executing a barrier is limited by the size of the region, but the packet emanate from the open region, into a potentially much larger portion of the network, and arrive sorted in by barrier order. Consider the potential
in a very large network to have transmitted packets which arrive in guaranteed order relative to fixed events.

3.4 An Extended Example: Open Region in the Center of a Two Dimensional Mesh

The nodes of a two-dimensional mesh can be labeled as grid coordinates on a Cartesian plane. Let $Y^+$ (respectively $Y^-$) denote the direction of increasing (respectively decreasing) $Y$ coordinates. Furthermore, let $X^+$ (respectively $X^-$) denote the direction of increasing (respectively decreasing) $X$ coordinates. In some literature $Y^+$ and $Y^-$ are referred to as North and South, and $X^+$ and $X^-$ are referred to as East and West respectively. Moreover, the notation $Y^+X^+$, $Y^+X^-$, $Y^-X^+$, and $Y^-X^-$ mean a route which goes first routes in the specified $Y$ direction, and then in the specified $X$ direction.

Consider a VIN on a two-dimensional mesh, using dimension order routing which routes the $Y$ dimension first, and then the $X$ dimension. Furthermore, we shall assume the integrated network barrier techniques of (Solworth and Stamatopoulos, 1993) to be in use. Now consider a rectangular open region located in the center of this mesh. The mesh is partitioned into nine different areas, which shall be enumerated similarly to the keys on a telephone keypad, and which is depicted in Figure 17. We shall assume that Proc($S'$) corresponds to the processors in area 5, and Proc($D'$) corresponds to all processors in the entire network. Area 5, shown with crosshatching, is the open region itself, and areas 1-4, and 6-9 correspond to portions of the network with distinct routing properties relative to the open region. The open region (area 5) runs from Lower$X$ to Upper$X$ in the X-direction, and Lower$Y$ to Upper$Y$ in the Y direction.
The routing follows the conventions of metropolis routing, and is resolved in the Y-direction first, and then the X-direction.

- To route from the open region to regions 1, 2, or 3, packets first route in the Y+ direction, then in the X+ or X- direction as appropriate.

- To route to regions 4 or 6, packets route in the Y direction routing before leaving the region, and thereafter only route in the X+ or X- directions once outside the region.

- To route to regions 7, 8, or 9, packets are routed in the Y- direction and then in the X+ or X- direction, as appropriate.

The following rules describe the placement of entry pickets to allow correct operation of the open region. Because this open region is the only open region in the VIN, it is possible to have more pickets than may be strictly needed.

1. Since data packets and barrier waves emanate from the open region, and none can enter the open region, the open region has entry pickets on all queues whose successors are in 5.

2. No processors outside the open region may insert packets or barrier markers into this VIN. Hence all input queues from processors outside the open region are marked as being on an entry picket.

3. Barrier waves cannot arrive from the direction opposite the open region. Hence:

   A. Switches above the region have entry pickets on the input queues from the Y+ direction.
B. Switches below the region have entry pickets on the input queues from the Y-direction.

C. Switches to the right of the open region have entry pickets on the input queues from the X+ direction.

D. Switches to the left of the open region have entry pickets on the input queues from the X- direction.

(4) All packets and barrier waves leaving the open region route first in the Y direction. This has two implications.

A. Once a packet or wave has propagated outside of the X range of the region, it may no longer move in either of the Y directions. Hence in nodes to the right or left of the region, which have x coordinates outside those of the open region, the input queues in both Y directions are on entry pickets.
B. The X side boundaries of the open region must be extended to the top and bottom of the network. In the regions above and below the open region, which have X coordinates the same as the region but Y coordinates outside the regions range, barrier waves will not be arriving from outside the region, hence entry pickets are required.

3.5 **An Open Region and Four Ply Virtual Interconnection Networks**

When four ply VINs are in use, packets are segregated into each of the four plies within the VIN according to the direction in which they must route, which must match the directions in
which the wave that sweeps the ply propagates. These are summarized in the following Table III and graphically in Figure 18.

<table>
<thead>
<tr>
<th>Ply Number</th>
<th>Wave Name</th>
<th>Packet Direction</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Y+X+</td>
<td>Y+X+, Y+</td>
</tr>
<tr>
<td>1</td>
<td>Y-X+</td>
<td>Y-X+, X+</td>
</tr>
<tr>
<td>2</td>
<td>Y-X-</td>
<td>Y- X-, Y-</td>
</tr>
<tr>
<td>3</td>
<td>Y+X-</td>
<td>Y+X-, X-</td>
</tr>
</tbody>
</table>

Table III. Wave Numbers and Packet Vectors

Nominally these waves, and their associated plies, can be identified by the directions of the diagonal routes on which their markers route. Furthermore, they are numbered clockwise: Y+X+ (0), Y-X+ (1), Y-X- (2), and Y+X- (3). In addition, routes which involve moving in one dimension only are assigned to the channel immediately clockwise to their direction.

Alternatively, the numbering scheme could be thought of as numbering the axis included in the channel, starting at the Y+ axis and proceeding clockwise, with the channel also including the quadrant clockwise from the included axis. Hence packets routing only in the Y+ direction are routed in channel Y+X+, packets routing only in the X+ direction are routed in channel Y-X+ channel.
There is an interesting geometric complication of this partitioning of packets by the wave that sweeps their routes. Consider how each given wave propagates on only one of the two axis adjacent to the quadrant it sweeps. This is illustrated in Figure 19. For example, the wave in ply 0 routes in the Y+ and the Y+X+ directions, but not in the X+ only direction, which instead is covered by wave 1. This means that a row of nodes outside an open region, in the X+ direction
from the extreme Y-edge of the region, are not swept by wave 0. There is a notch that is not reachable from any of the nodes in the source region on that wave. Similar notches occur on the clockwise faces of each wave’s region.
Figure 19. Area Swept by Wave on Ply 0
3.6 Recursive Tiling of a Network

A mesh network can be recursively divided into regions of a given type. We shall assume the network was partitioned into regions using a recursive division algorithm, which alternatively splits each sub grid horizontally and then vertically. The regions are then assigned numbers based on the order in which they would be created by the algorithm, the first region being
number one, second being number two, and so on. Please see Figure 20 which shows how the square mesh was divided up into two to sixteen regions. Also note that each region is labeled with the dimension it would have if the entire network were 16 x 16.

In the case of closed regions, which do not interact with surrounding network components, this tiling can divide a single VIN into disjoint sub grids. This result is interesting because a supercomputing installation can buy one large parallel processor, and dynamically partition it into sub-grids. A sub-grid can operate and synchronize in isolation, without the cost associated with a machine wide barrier. Hence the sub-grid will have no performance loss relative to a smaller sized parallel processor. Consider Figure 21 in which a single VIN is decomposed into four regions. Note that the crosshatched partitions are non-permeable to either packets or barrier markers, and would hence contain both entry and exit pickets.
In the case of open regions, the routes from the source processors to the destination processors must be kept separate. If all processors are going to be destinations, or if there is any potential for intersection of paths, then the regions must be kept on separate VINs. Consider the tiling in Figure 22, in which the network is divided into four open regions, each of which resides on its own VIN, and hence allows any processor to be a legal destination.

Hence in the diagram, the shaded regions represent the source regions, while the entire VIN represents the destination region. Hence each region can perform a partial barrier over one
forth of the overall network, yet the packets from any processor to any other processor routing
over the network will arrive at their destinations in barrier order.

Figure 22. Open Region Tiling across Four Virtual Interconnection Networks
4: Experimental Methods and Results

4.1 Network Model

In order to confirm the correctness and measure the performance of the techniques developed in this thesis, a simulator was written in C++ which simulated the activity of two-dimensional mesh networks. This simulator, named NetQSim for “Network of Queues Simulator”, consists of 452 kilobytes of C++ using standard template libraries. It uses compiler directives to allow compilation on either gnu C++ or on Microsoft Visual C++, where it runs in a command prompt window.

NetQSim replaces an earlier simulator named InetQSim for “Interconnection Network of Queues Simulator,” which was written in an older (1993) version of C++ without modern extensions, but more importantly had become too large and cumbersome to modify.

The simulator is designed to “fail fast”. As the simulator runs, it does internal consistency checks and if a questionable condition arises prints out error messages and halts. There are too many fatal conditions to list here, but some examples include: a counter overflow; a packet arriving out of order relative to a barrier; a packet at the head of a non-destination queue without a valid successor; a packet being inserted into a queue with an incorrect VIN designation; a packet with a negative latency value; or a processor receiving a packet addressed to a different processor. Hence completion of the simulation gives greater assurance that such erroneous conditions did not occur during the run.
The simulator models a network with the following properties.

1. Packet switching using virtual cut through switching.
2. Packet objects, albeit containing minimal data payload, are generated in memory and transferred via pointers, through the queues.
3. The sizes of packets and the capacities of queues are a compile time constant. The default size used was one flit packets and 64 flit queues.
4. The network has a geometry of a two-dimensional Mesh (x, y).
5. Dimension routing, routing first in the Y dimension, and then on X is used to ensure deadlock free routing.
6. Separate rifle and turn input queues are used on external wires, and separate X-turn and Y-turn input queues are used to inject packets from the processor. This mimics the switch design used in Stamatopoulos and Solworth’s work.
7. Multiple piles and multiple VINs are supported.
8. The type of VINs used, either based on Stamatopoulos and Solworth’s edge waves (one ply per VIN), or on corner waves as proposed by McShane (four plies per VIN), or a compressed form of corner waves (two plies per VIN), may also be set at the command line.

Traffic issues have been separated out from the structure of the processor.

1. Traffic patterns generated depend upon command line parameters. These parameters include the following.
a. The rate at which processors attempt to inject packets,

b. The frequency of barriers, and

c. Whether the barriers are based on simulator time or on the number of packets generated,

2. The traffic generation mechanism can generate uniform traffic or a short haul traffic distribution, (packets are generated and sent to processors no more than two links in any given dimension). This enables the reproduction of the results in (Stamatopoulos, 1996) in order to verify the simulator.

Definition. **Uniform Traffic Distribution** - a synthetic traffic pattern for packets, in which a processor may send packets to any other processor with equal probability, but not to itself.

The reason for disallowing the generation of packets addressed to one self is simple. A packet destined for its originating process would route not through the interconnection network.

4.2 **Performance Metrics**

Before discussing the experiments, we first need to define and justify the performance metrics which have been used. The two types of measurements are useful to understand network performance: Packet Latency and Bandwidth (as measured by achieved load and bisection utilization.)
4.2.1 Packet Latency

The first and probably most intuitive metric for network performance is packet latency, which measures how long it takes for the packet to arrive at its destination. In general, we will be measuring the mean packet latency of all packets transferred during the simulations. While the lower bound for latency on an individual packet is easy to define, the expected average and upper bound remains nebulous.

Contributing factors to latency include but are not limited to: queue topology of the network (as per the QDG), queue capacity, the routing algorithm being used, the destination distribution being used, congestion (i.e. pre-existing packets stored) in the network, arbitration logic for wire allocation, the injection rate, and barrier frequency.

Definition. Packet latency is the number of cycles it takes for the first flit of a packet to be removed from the destination queue starting from the time that the first flit is inserted into the source queue.

Based on the properties we have specified for our network, we shall briefly analyze how latency arises within the network. As a packet routes through the network, it alternatively propagates and waits, both of which take time.

4.2.1.1 Packet Latency on an Empty Network

In order to understand how packet latency relates to network performance, we first examine unimpeded packet latency, which is the latency on an empty network. This defines the
lower bound on packet latency. Hence for the time being, we shall assume that all queues are empty.

Consider an empty queue, which contains no flits, into which we are inserting the first flit of a packet. Since the queue is empty, the flit being inserted must wait at least one cycle. This is because in Section 2.3.1, we specified that a flit must be completely inserted into a queue before it can begin to be removed, which would require one cycle.

Now consider a link between two switches. To traverse a switch, two queues must be traversed. Each queue requires a minimum of one cycle to traverse. Hence for any packet to traverse any single switch requires at least two cycles.

Next we consider the minimum latency to route between two processors across part of the network. Assume that the source processor is on a switch which required L inter-switch links in order to reach the switch associated with the destination processor. It requires 1 link to get from the source queue on the originating processor to the switch, L links to reach the destination processor’s switch, and 1 link to get from the switch to the destination queue on the destination processor. Hence (L+2) links are traversed, and hence $2(L+2) = 2L + 4$ queues are traversed, for a minimum latency of $2L + 4$ cycles.
4.2.1.2 Non-Empty Queues

We next consider a network in which the queues traversed are not empty. Since there is no contention between the queues, this corresponds to having flits in queues only directly along the path a packet is taking.

Section 2.3.1 also stipulated that flits flow into and out of a queue at a rate of one flit per cycle. Hence if the queue contains flits (D>0), then the new flit must wait at least D cycles, for those flits to transfer out. Note that since one of the flits may be transferring out at the same time as the new flit is transferring in, the time is the same for one flit or no flits of data already in the queue.

4.2.1.3 Contention Between Queues

Next we consider when packets compete for resources with packets routing on different paths.

Consider the case of a queue with multiple predecessors, such as in Figure 23. Since we stipulated in section 2.3.1 that only one flit may be transferred into a queue at once at any given time, only one of the predecessors may transfer one of its flits at a time to the successor queue. The other queues must wait if they have packets at their heads which must route through the successor queue.
Now consider if each of the N predecessor queues contained was full and contained $C_q$ flits. A new flit inserted into one of the predecessor queues could wait for as many as $(N \times C_q)$ flits to transfer into the successor queue first.

Now consider if predecessor queues are not given equal consideration, but one is favored over the others when selecting which one may transfer packets to the successor queue. **Routing Priority** is a relative weight assigned to a queue, based on the packet at the queues head, which allows it to be favored by the transfer scheduling mechanism.
Finally we note that these factors cascade. This is particularly true for non-empty queues and contention, which causes pooling. This prevents transfer of packets upstream, until those queues fill up and block. This phenomenon is called **Tree Saturation** (Pfister and Norton, 1985).

4.2.1.4 **Barrier Latencies**

If barriers are in use, then they enforce a routing priority scheme in which post barrier packets must wait for pre-barrier packets to route first. In some systems, a time stamp might also be used to allow chronologically older packets to route first. Hence the use of INBs inherently increased the latency of packets behind the wave on a given ply. This is called **Barrier Latency**, and can be assumed to always be present if barriers are in use (Stamatopoulos, 1996).

Counter intuitively, INBs can increase performance and by increasing fairness in the system, and is described in 2.9.3, and was a major result in (Stamatopoulos, 1996).

4.2.1.5 **In Saturation**

If packets are inserted into the predecessors faster than they are removed, then the queue will gradually fill up. We call this process of packets accumulating in queues in areas of the network **pooling**. Eventually the congestion will back up to the source queues, and they will become full and hence be unable to receiving packets continuously. When this happens the processors are said to **block**, suspending initiating operations until they are able to inject packets again.
In general, increasing the amount of storage in the network increases packet latency if the injection rate exceeds the ability of the network to deliver messages. Increasing the capacity of queues allows greater pooling. Increasing the number of queues in parallel, such as by adding turn queues or increasing the number of virtualized buffers can introduce greater delays were the different paths join, as shown above.

In this case it is more reasonable to measure bandwidth, since that will determine how long it takes to perform a computation. In such a case it matters little whether the latency is in the network (packet latency) or if the processor which must wait to insert a packet when the source queue is full.

4.2.2 Bandwidth

Bandwidth measures the rate at which data is transferred. In this case, it is measured in flits per cycle. This can be measured in two ways. **Offered load** refers to the attempted rate at which packets are intended to be generated by a given processors, and **achieved load** is the sustainable rate at which they can actually be generated and injected. For the experimental results given here, this is recorded as the average for all processors. It is worth noting that although as packets tend to pool near the center of the network, processors on the edge of the network are often able to achieve considerably higher peak offered load than those in the center of the network. The achieved load is highly dependent on the specific traffic pattern in use and the exact queue topology of the network. In a network of two nodes, the queues simple form two chains between the two processors and 100% offered load is trivially achieved. For uniform
traffic distributions, achieved load may be estimated by determining which load will saturate the network bisection.

It is also interesting to measure bandwidth at potential bottlenecks in the network. The goal is ultimately to drive the bottleneck to 100% usage. What constitutes a bottleneck in the network depends on exactly what the offered load and traffic patterns are. In the case of uniformly distributed traffic on a Mesh, the bisection of the network represents such a bottleneck. We shall show how congestion along the bisection limits achieved load.

**Definition Bisection Bandwidth:** The bandwidth across a set of links, which divides the network in half. If the network is non-symmetrical, the smallest value is used, since this is assumed to be a worst-case metric (Hennessy and Patterson, 1998).

**Definition X Bisection:** The set of links, which divides the network in half, so that half of the columns (with x values constant) are in each half.

**Definition Y Bisection:** The set of links, which divides the network in half, so that half of the rows (with y values constant) are in each half.

In order to keep the X bisection and Y bisection distinct it is useful to remember the following. The links in the Y bisection can be crossed by a line parallel to the X axis, which divides the network into two sets of row, which have variable X and constant Y. Similarly, the links of the X bisection can be crossed by a line parallel to the Y axis, and which divides the
network into two sets of columns, which have variable Y and constant X. Please see Figure 24 which illustrates this arrangement.

*Definition Bisection Utilization:* The fraction of the bandwidth across a bisection, which is actually used to transfer flits. This may be further specified as the **X Bisection Utilization, Y Bisection Utilization**, or unless otherwise specified, the **Mean Bisection Utilization** (average of the X and Y.)
Bisection utilization is measured for a simulation by keeping track of the total number of busy and idle cycles for the links on the bisection during the run. The total number of busy cycles is divided by the sum of busy and idle cycles. Generally the X bisection and Y bisection converge for uniform traffic. Significant variance between the two values indicates that something is wrong with the traffic pattern being generated, or that the routing algorithm is not functioning properly.

Consider a network as depicted in Figure 25, consisting of one row of R nodes, where R is an even, non-zero, positive integer. Assume that a uniform traffic pattern is used, so a packet's source node (sender) and destination (receiver) are distinct. Hence for any given node x, there are (R-1) legal destinations for a packet originating at x. In addition, for x above, (R/2) of the legal destinations are across the bisection from x. Hence, for any given source node, the probability that a packet originating at that node will require crossing the bisection to reach its destination is given by the equation:

\[ P(R) = \frac{(R/2)}{(R-1)} = \frac{R}{2(R-1)} \]

Furthermore, by applying L'Hôpital's rule to P, we find the limit of P(R) as R goes to infinity.

\[ \lim_{R \to \infty} P(R) = \frac{(R')}{(2(R-1))'} = \frac{1}{2} \]
This implies that as $R$ diverges to infinity, $P(R)$ asymptotically approaches $1/2$ from above. Hence under uniform traffic; at least half of all packets will need to route across the bisection. Since only two packets, one in each direction, may route across the bisection, the bisection links become a bottleneck, and it is desirable to keep their utilization as close to one hundred percent as possible.

By assuming that half of all packets need to cross the bisection, we can estimate the maximum achievable load for a given two dimensional mesh by the length of its sides. This is illustrated in Table IV.
<table>
<thead>
<tr>
<th>Mesh Size</th>
<th>Total Nodes</th>
<th>Nodes per Side of Bisection</th>
<th>Links Across One Bisection</th>
<th>Bisection Links Per Node</th>
<th>Load to Saturate Bisection</th>
</tr>
</thead>
<tbody>
<tr>
<td>2 x 2</td>
<td>4</td>
<td>2</td>
<td>2</td>
<td>1</td>
<td>200.000%</td>
</tr>
<tr>
<td>4 x 4</td>
<td>16</td>
<td>8</td>
<td>4</td>
<td>0.5</td>
<td>100.000%</td>
</tr>
<tr>
<td>8 x 8</td>
<td>64</td>
<td>32</td>
<td>8</td>
<td>0.25</td>
<td>50.000%</td>
</tr>
<tr>
<td>16 x 16</td>
<td>256</td>
<td>128</td>
<td>16</td>
<td>0.125</td>
<td>25.000%</td>
</tr>
<tr>
<td>32 x 32</td>
<td>1024</td>
<td>512</td>
<td>32</td>
<td>0.0625</td>
<td>12.500%</td>
</tr>
<tr>
<td>64 x 64</td>
<td>4096</td>
<td>2048</td>
<td>64</td>
<td>0.03125</td>
<td>6.250%</td>
</tr>
</tbody>
</table>

Table IV. Load To Saturate the Bisection in a Two Dimensional Mesh

4.3. Simulations without Regions

4.3.1 Single Ply Virtual Interconnection Networks and Edge Waves

In order to verify the behavior of the simulator, these results are included to calibrate the simulator, and demonstrate results are similar versus those generated by Stamatopoulos. The network was divided into VINs consisting of a single ply each, and barrier waves propagated away from the Y-edges of the network, then the X edges. The numbers are not exactly the same as those in (Stamatopoulos, 1996), since slight changes in switch design or operational parameters can significantly impact the performance numbers. However, they show similar behavior.
The following sets of simulations were run using the following parameters, unless otherwise specified.

1. a 16 x 16 Mesh;
2. single flit packets;
3. 64 flit queues;
4. a uniform traffic distribution;
5. The offered load was 100%, meaning that processors attempted to send a packet every cycle;
6. The number of physical plies used ranged over 1, 2, 4, 8, and 16, using one physical ply per VIN; and
7. When no barriers were used, packets were sent by alternating VIN in a round robin fashion. Hence if N VINs were in use, each VIN transferred one packet out of every N packets. When barriers were used, the conventions followed those in (Stamatopoulos, 1996). In which packets were injected into a VIN until it was time for a barrier, when a barrier marker was inserted after the last data packet. After that barrier, data packets were injected into the next VIN in a cyclic order, so that data packets were never trapped behind a barrier.

When viewing these results, remember that as discussed in section 4.2.3.2, and listed in Table IV, a 16 x 16 Mesh network with uniform traffic would have an estimated maximum sustainable achieved load of 0.25 which would drive the bisection usage to 1.00 from below.
Consider the results in Table V. These contain the unbarriered case in which for each of the N VINs a processor injects a packet at $1/N_{th}$ of the effective offered load. The achieved load is per processor, the bisection is where the bandwidth is limited. Hence these results will be tabulated in terms of bisection bandwidth. As consistent with Table IV, the bisection is approximately four times the achieved load for a 16 x 16 Mesh with a uniform traffic distribution. In spite of increasing the available storage from 1 to 16 plies, the peak offered load bisection is 0.933248 and when 8 plies are in use, which is below our ideal of 100%.

Performance incrementally improves at each step up to 8 plies, but each increment requires twice the amount of storage in each switch. Comparing the 1 ply to 8 ply cases, to get approximately a 5.8% increase in bisection usage, 8 times as much queue space and switching resources are required.
As discussed earlier, the INBs were developed in (Solworth and Stamatopoulos, 1993, 1994, 1995, 1996, 2002; Stamatopoulos, 1996) were intended as a congestion control device. Furthermore, two plies were found to be adequate to get peak performance when barriers with the properly tuned frequency were used (Stamatopoulos, 1996). Now consider the cases recorded in Table VI, in which 1 barrier was performed every 128 cycles. Notice that the single ply case showed a decrease in performance relative to the unbarriered case, which corresponds to packets being trapped behind a barrier marker and having to wait for the wave to clear before progressing, but that the two ply case shows a 16.8% increase in bisection usage 0.995813, which is within 0.42% of the theoretical maximum value. Further increases in the number of plies show negligible gains, within the range of statistical noise, for doubling the cost in

<table>
<thead>
<tr>
<th>Number of VINS (Plies)</th>
<th>Packets per Barrier</th>
<th>Achieved Load</th>
<th>Mean Bisection Usage</th>
<th>Number of Packets</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 (1)</td>
<td>N/A</td>
<td>0.218031</td>
<td>0.875156</td>
<td>16,777,216</td>
</tr>
<tr>
<td>2 (2)</td>
<td>N/A</td>
<td>0.221921</td>
<td>0.891153</td>
<td>16,777,216</td>
</tr>
<tr>
<td>4 (4)</td>
<td>N/A</td>
<td>0.227668</td>
<td>0.914092</td>
<td>16,777,216</td>
</tr>
<tr>
<td>8 (8)</td>
<td>N/A</td>
<td>0.232451</td>
<td>0.933248</td>
<td>16,777,216</td>
</tr>
<tr>
<td>16 (16)</td>
<td>N/A</td>
<td>0.226611</td>
<td>0.909408</td>
<td>16,777,216</td>
</tr>
</tbody>
</table>

Table V. Single Ply VINs No Barriers
hardware. It is also worth noting that as the number of plies doubles, so does the amount of storage in the network.

<table>
<thead>
<tr>
<th>Number of VINS (Plies)</th>
<th>Packets per Barrier</th>
<th>Achieved Load</th>
<th>Mean Bisection Usage</th>
<th>Number of Packets</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 (1)</td>
<td>128</td>
<td>0.206503</td>
<td>0.827663</td>
<td>16,777,216</td>
</tr>
<tr>
<td>2 (2)</td>
<td>128</td>
<td>0.249267</td>
<td>0.995813</td>
<td>16,777,216</td>
</tr>
<tr>
<td>4 (4)</td>
<td>128</td>
<td>0.249810</td>
<td>0.996114</td>
<td>16,777,216</td>
</tr>
<tr>
<td>8 (8)</td>
<td>128</td>
<td>0.250696</td>
<td>0.995131</td>
<td>16,777,216</td>
</tr>
<tr>
<td>16 (16)</td>
<td>128</td>
<td>0.249652</td>
<td>0.994756</td>
<td>16,777,216</td>
</tr>
</tbody>
</table>

Table VI. Single Ply VINs, 1 Barrier Per 128 Packets

4.3.2 Four Ply Virtual Interconnection Networks and Corner Wave Barriers

These results are the first experimental measurements of barriers using the wave pattern type proposed by McShane. Each VIN contains four plies, which contain routes which propagate away from one corner to the opposite corner. Each ply is also swept by exactly one corner wave.

Otherwise, these simulations follow the same conventions as the simulations above. They were run using a 16 x 16 Mesh, with 1 flit packets, 64 flit queues, a uniform traffic distribution,
and an attempted offered load of 100%. Since each VIN requires four physical plies, they ranged over 4, 8, and 16, producing 1, 2, and 4 VINs respectively. The same load sharing and VIN alteration conventions were used as in the single ply VIN simulations above.

Consider the unbarriered cases of four ply VINs tabulated in Table VII. Not only is there no improvement in performance as the number of VINs is double, achieved load and bisection usage appear to actually decrease slightly as the number of VINs increases. Clearly in the unbarried case, a single VIN of four physical plies is adequate to get peak performance.

<table>
<thead>
<tr>
<th>Number of VINS (Plies)</th>
<th>Packets per Barrier</th>
<th>Achieved Load</th>
<th>Mean Bisection Usage</th>
<th>Number of Packets</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 (4)</td>
<td>N/A</td>
<td>0.235023</td>
<td>0.943571</td>
<td>16,777,216</td>
</tr>
<tr>
<td>2 (8)</td>
<td>N/A</td>
<td>0.232429</td>
<td>0.932956</td>
<td>16,777,216</td>
</tr>
<tr>
<td>4 (16)</td>
<td>N/A</td>
<td>0.233641</td>
<td>0.937871</td>
<td>16,777,216</td>
</tr>
</tbody>
</table>

Table VII. Four Ply VINs No Barriers

Comparing the multi-ply VINs with those in the single ply VIN cases requires some care. Obviously, these VINs contain more plies than the single ply case, however these plies are not the same. Each of the multi-ply VINs only route packets in two directions each, rather than four. Hence the switches here contain nine queues per ply in the general case, rather than the fifteen
required in the general case of single ply VIN ply. Hence if we ignore the edges and corners, a four ply VIN here has \((4*9)/15 = 2.40\) times as many queues as a VIN in the single ply VIN.

Now comparing the performance numbers between the two unbarriered cases we see that for four ply VINs, the worst specific results were for two VINs (eight plies). Nevertheless, the performance was better than the best case performance for the single ply, unbarriered case, which was for eight VINs (also eight plies).

Now consider the case in which one barrier is run for every 128 packets sent, as summarized in Table VIII. Even in the single VIN case, we are within 0.7976 % of the maximum theoretic achieved load, and 0.6474 % of the maximum bisection utilization. Hence addition VINs beyond the first yield negligible gains.
### Table VIII. Four Ply VINs 1 Barrier per 128 Packets

<table>
<thead>
<tr>
<th>Number of VINS (Plies)</th>
<th>Packets per Barrier</th>
<th>Achieved Load</th>
<th>Mean Bisection Usage</th>
<th>Number of Packets</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 (4)</td>
<td>128</td>
<td>0.248006</td>
<td>0.993526</td>
<td>16,777,216</td>
</tr>
<tr>
<td>2 (8)</td>
<td>128</td>
<td>0.249775</td>
<td>0.995089</td>
<td>16,777,216</td>
</tr>
<tr>
<td>4 (16)</td>
<td>128</td>
<td>0.250650</td>
<td>0.994989</td>
<td>16,777,216</td>
</tr>
</tbody>
</table>

#### 4.3.3 Summary

In conclusion we see that for mesh networks under uniform traffic, the use of full barriers can increase network usage, as measured by bisection usage, regardless of whether edge or corner waves are used. Please see the graph in Figure 26 which illustrates this point.

This graph shows the following.

1. Multi-ply VINs perform better than single ply VINs.
2. Except in the single VIN of the single ply VIN case, simulations with 1 barrier per 128 messages outperform those with no barriers.
3. The performance between single and multi-ply VINs without barriers are comparable.
4.4 Tiling with Closed Regions

The previous section considered full barriers as proposed by Stamatopoulos and Solworth. In this and later sections, we consider partial barriers on a closed region. A number of tests were performed to determine if a closed region behaved in a manner consistent with a whole network of the same size. These tests were performed for closed regions of (4 x 4), and (8 x 8) topologies, and found they produced identical results up to statistical noise. Achieved load for these results are tabulated in Table IX.
This result is interesting because a supercomputing installation can buy one large parallel processor, and dynamically partition it into sub-grids. A sub-grid can perform barriers within the sub-grid, without the cost associated with a machine-wide barrier. Hence the sub-grid will have no performance loss relative to a parallel processor of smaller size for other computations.

<table>
<thead>
<tr>
<th>Size</th>
<th>VIN Type</th>
<th>Packets Per Barrier</th>
<th>Whole Network</th>
<th>Closed Region</th>
<th>Difference</th>
</tr>
</thead>
<tbody>
<tr>
<td>4x4</td>
<td>Single Ply</td>
<td>N/A</td>
<td>0.697166</td>
<td>0.645012</td>
<td>7.481%</td>
</tr>
<tr>
<td>4x4</td>
<td>Multi-Ply</td>
<td>N/A</td>
<td>0.871756</td>
<td>0.869616</td>
<td>0.245%</td>
</tr>
<tr>
<td>4x4</td>
<td>Single Ply</td>
<td>128</td>
<td>0.684921</td>
<td>0.684336</td>
<td>0.085%</td>
</tr>
<tr>
<td>4x4</td>
<td>Multi-Ply</td>
<td>128</td>
<td>0.923706</td>
<td>0.921448</td>
<td>0.244%</td>
</tr>
<tr>
<td>8x8</td>
<td>Single Ply</td>
<td>N/A</td>
<td>0.404888</td>
<td>0.405028</td>
<td>-0.035%</td>
</tr>
<tr>
<td>8x8</td>
<td>Multi-Ply</td>
<td>N/A</td>
<td>0.464582</td>
<td>0.462824</td>
<td>0.378%</td>
</tr>
<tr>
<td>8x8</td>
<td>Single Ply</td>
<td>128</td>
<td>0.379011</td>
<td>0.378258</td>
<td>0.199%</td>
</tr>
<tr>
<td>8x8</td>
<td>Multi-Ply</td>
<td>128</td>
<td>0.488734</td>
<td>0.487776</td>
<td>0.196%</td>
</tr>
</tbody>
</table>

Table IX. Equivalence of Closed Regions and Small Networks

4.5 Tiling with Open Regions

The purpose of these experiments was to determine if a set of disjoint open regions, which covered all processors of the network, could use partial barriers within the regions to give barrier semantics (i.e., segregation of packet arrival relative to barrier order) for packets originating in the same region, with comparable performance to a full barrier.
Since an open region cannot contain all processors in the network, a closed region covering the entire network is tabulated for purposes of comparison. Note that for a single ply without barriers, the results are identical down to the last decimal point to the whole network, single ply, and unbarriered case. These were in fact separate simulations the behavior is both theoretically and experimentally identical. Since open regions cannot share routes or queues without interfering with each others barriers, and since we want to simulate uniformly distributed traffic, only one open region may be created per VIN. The simulator can efficiently simulate 16 physical plies, which can be partitioned into 16 VINs using edge-based waves, or 4 VINs using corner-based waves. Hence using the edge waves simulations could be run for 1, 2, 4, 8, and 16 open regions. For corner waves, 1, 2, 4 open regions were possible.

The network was partitioned into regions using a recursive division algorithm, which alternatively split each horizontally and then vertically. The regions were assigned numbers based on the order in which they would be created by the algorithm, and these numbers also indicate the order in which they are assigned to VINs starting at the lowest number.

Consider Table X which shows the performance of a 16 x 16 mesh network, divided into regions such that the source regions tile the entire network. The destination regions for each of these source regions are the entire network. As noted above, the single ply case is identical to the whole network case. The peak performance is at eight plies, however the peak bandwidth utilization is only 0.937404. Furthermore at 16 VINs, the amount of storage and joins at external links start to drive up the latency and performance starts to decay.
Note that other than the configuration of the barriers which sweep from each source region to every processor, this is the same load as discussed in section 4.3.2.

<table>
<thead>
<tr>
<th>Number of VINS (Plies)</th>
<th>Size of Region</th>
<th>Packets per Barrier</th>
<th>Achieved Load</th>
<th>Mean Packet Latency</th>
<th>Mean Bisection Usage</th>
<th>Number of Packets</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 (1)</td>
<td>16 x 16</td>
<td>N/A</td>
<td>0.218031</td>
<td>421.942</td>
<td>0.875156</td>
<td>16,777,216</td>
</tr>
<tr>
<td>2 (2)</td>
<td>8 x 16</td>
<td>N/A</td>
<td>0.217884</td>
<td>421.433</td>
<td>0.874708</td>
<td>16,777,216</td>
</tr>
<tr>
<td>4 (4)</td>
<td>8 x 8</td>
<td>N/A</td>
<td>0.229617</td>
<td>426.815</td>
<td>0.921866</td>
<td>16,777,216</td>
</tr>
<tr>
<td>8 (8)</td>
<td>4 x 8</td>
<td>N/A</td>
<td>0.233529</td>
<td>434.866</td>
<td>0.937404</td>
<td>16,777,216</td>
</tr>
<tr>
<td>16 (16)</td>
<td>4 x 4</td>
<td>N/A</td>
<td>0.205927</td>
<td>651.004</td>
<td>0.826806</td>
<td>16,777,216</td>
</tr>
</tbody>
</table>

Table X. Open Tiled Network, Single Ply VINs, No Barriers

Now consider tables XI which summarizes the performance for a 16 x 16 Mesh, 64 flit queues, 1 flit packets, single ply VINs, open tiled regions, and 1 barrier per 128 packets. Note that the achieved load and bisection usage increased as the number of VINs in use increased. While the performance still lags behind the corresponding cases with whole network VINs, we can see how the achieved load and bisection usage are converging toward optimal values as the number of regions increases.
<table>
<thead>
<tr>
<th>Number of VINS (Plies)</th>
<th>Size of Region</th>
<th>Packets per Barrier</th>
<th>Achieved Load</th>
<th>Mean Bisection Usage</th>
<th>Number of Packets</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 (1)</td>
<td>16 x 16</td>
<td>128</td>
<td>0.206503</td>
<td>0.827663</td>
<td>16,777,216</td>
</tr>
<tr>
<td>2 (2)</td>
<td>8 x 16</td>
<td>128</td>
<td>0.182461</td>
<td>0.731272</td>
<td>16,777,216</td>
</tr>
<tr>
<td>4 (4)</td>
<td>8 x 8</td>
<td>128</td>
<td>0.236909</td>
<td>0.951210</td>
<td>16,777,216</td>
</tr>
<tr>
<td>8 (8)</td>
<td>4 x 8</td>
<td>128</td>
<td>0.239884</td>
<td>0.962992</td>
<td>16,777,216</td>
</tr>
<tr>
<td>16 (16)</td>
<td>4 x 4</td>
<td>128</td>
<td>0.246048</td>
<td>0.987719</td>
<td>16,777,216</td>
</tr>
</tbody>
</table>

Table XI. Open Tiled Network, Single Ply VINs, 1 Barrier per 128 Packets

Now we must consider the case of four ply VINs and open tilings, as tabulated in Tables XII and XIII. This combination of two performance boosting features gives surprisingly disappointing results. The addition of more VINs appears to cause performance degradation rather than improvement. Furthermore, while the use of barriers increases performance in the non-tiled case, it appears to hamper performance in the 2 VIN and 4 VIN cases.
<table>
<thead>
<tr>
<th>Number of VINS (Plies)</th>
<th>Size of Region</th>
<th>Packets per Barrier</th>
<th>Achieved Load</th>
<th>Mean Bisection Usage</th>
<th>Number of Packets</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 (4)</td>
<td>16 x 16</td>
<td>N/A</td>
<td>0.235023</td>
<td>0.943571</td>
<td>16,777,216</td>
</tr>
<tr>
<td>2 (8)</td>
<td>8 x 16</td>
<td>N/A</td>
<td>0.232593</td>
<td>0.934043</td>
<td>16,777,216</td>
</tr>
<tr>
<td>4 (16)</td>
<td>8 x 8</td>
<td>N/A</td>
<td>0.227271</td>
<td>0.912324</td>
<td>16,777,216</td>
</tr>
</tbody>
</table>

Table XII. Open Tiled Network, Four Ply VINs, No Barriers

<table>
<thead>
<tr>
<th>Number of VINS (Plies)</th>
<th>Size of Region</th>
<th>Packets per Barrier</th>
<th>Achieved Load</th>
<th>Mean Bisection Usage</th>
<th>Number of Packets</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 (4)</td>
<td>16 x 16</td>
<td>128</td>
<td>0.248006</td>
<td>0.993526</td>
<td>16,777,216</td>
</tr>
<tr>
<td>2 (8)</td>
<td>8 x 16</td>
<td>128</td>
<td>0.223827</td>
<td>0.898744</td>
<td>16,777,216</td>
</tr>
<tr>
<td>4 (16)</td>
<td>8 x 8</td>
<td>128</td>
<td>0.214966</td>
<td>0.862787</td>
<td>16,777,216</td>
</tr>
</tbody>
</table>

Table XIII. Open Tiled Network, Four Ply VINs, 1 Barrier per 128 Packets

Consider the plot of these results in Figure 27. It demonstrates the following.

1. We see that for the single ply, barriered case, our performance converges to the
ideal as the number of open region increases, but that in the unbarriered case, performance starts to decline after 8 VINs.

2. Additionally, we see that the best performance for a single VIN is the single region, multi-ply VIN with barriers, followed by the single region, multi-ply VIN with barriers.

3. We see that the overall trend in multi-ply VINs is to have performance degrade as the number of VINs increases.

The single ply VIN, open region tiling approaches the same performance as (Stamatopoulos, 1996). However, note that these semantics are much stronger, due to the concurrent execution of multiple partial barriers. Hence there is a very small degradation for using multiple partial barriers, while in other schemes the overhead is substantial.
4.6 Time Stamp Versus Barrier Arbitration

When using multi-ply VINs with barriers, bandwidth usage tends to drop as the number of VINs, (and hence numbers of plies) aggravates the length of time packets wait to transfer across the shared external link on inter-switch connections.

Figure 27. Graph of Mean Bisection Usage With Open Region Tiles
If we allow the age of packets to be used to weight the priority with which packets traverse the external wire, this can reduce this effect by reducing the amount of time a queue can go without servicing. Table XIV compares two sets of simulations, using a 16 x 16 Mesh, 64 flit queues, 1 flit packets, 128 packets per barrier, and 100% offered load. Each simulation ran until 16,777,216 packets had been delivered.

<table>
<thead>
<tr>
<th>Number of VINS (Plies)</th>
<th>Size of Region</th>
<th>Round Robin Arbitration</th>
<th>Time Based</th>
<th>Differences</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>Achieved Load</td>
<td>Mean Bisection Usage</td>
<td>Achieved Load</td>
</tr>
<tr>
<td>1 (4)</td>
<td>16 x 16</td>
<td>0.248006</td>
<td>0.993526</td>
<td>0.248540</td>
</tr>
<tr>
<td>2 (8)</td>
<td>8 x 16</td>
<td>0.223827</td>
<td>0.898744</td>
<td>0.225691</td>
</tr>
<tr>
<td>4 (16)</td>
<td>8 x 8</td>
<td>0.214966</td>
<td>0.862787</td>
<td>0.223151</td>
</tr>
</tbody>
</table>

Table XIV. Open Tiled Network, Four Ply VINs, 1 Barrier per 128 Packets

4.7 Acknowledgements and Sequential Consistency

To implement sequential consistency is sufficient to require that each processor have one message outstanding at a time. To implement this, each packet receives an explicit acknowledgement packet from the destination of the original packet before it may inject another into the system. In Table XVI to XVIII, the results of some experiments on a 16 x 16 network with 1 flit packets, 64 flit queues, single ply VINs, two VINs in use with one for acknowledge
packets only, and the other with a 8 x 8, 4 x 4, and a 2 x 2 open region in the center, and a uniform traffic distribution on the original packets. Table XV contains the same operational parameters for a 16 x 16 closed network for comparison.

The results tabulated in this section include the following fields.

- **Acknowledgement** – specified the type of operations for which the processors will wait for an acknowledgement packet to be returned to them before they proceed to produce and inject another packet. In the case of read operation, this requires the returned value. In the case of a write request, it is a message indicated that the update to storage has been completed. A 50% mix of operations, (equal reads and writes) is being performed in these simulations.

- **Barriers** – Barriers are either injected after each message to ensure sequential consistency, or no barriers are used and the acknowledgements are used to ensure sequential consistency.

- **Bisection** – bisection usage as defined in 4.1.2.6. Note that this is for the entire network, not just the smaller open region, and hence should only be used relative to other entries in the same table. (Barriers markers do not contribute to measured bisection usage.)

- **Achieved Load** – the injection rate for processors in the source region, not the entire network. This is comparable to other data from other experiments.

- **Latency** – the mean packet latency for packets from inside the region.
Consider the performance of a 16 x 16 closed region, as tabulated in Table XV. Operation with one barrier per packet without waiting for any acknowledgements produced 4.03 times the achieved load as the case in which the processors waits for each packet to be explicitly acknowledged before injecting a new packet. As can be seen from the table, with one message per barrier, the barriers add significant latency if the processors are not waiting for acknowledgements, since the time to complete a barrier is longer than the average empty network packet latency. When one barrier is issued between each operation and each processor waits for reads to be acknowledged (i.e. the issuing processor blocked until the value was returned,) produces a 6.47% increase in achieved load over when each packet sent required the issuing processor block pending receiving an acknowledgement. Although the 4.03 times performance improvement (i.e. the barriers and no acknowledge case) only occurs for 100% write loads, it is indicative of the performance when multiple unacknowledged packets are outstanding.
Table XV. 16 x 16 Closed Region Acknowledgements Versus Barriers

<table>
<thead>
<tr>
<th>Acknowledge</th>
<th>Barriers</th>
<th>Bisection</th>
<th>Achieved Load</th>
<th>Latency</th>
</tr>
</thead>
<tbody>
<tr>
<td>Read, Write</td>
<td>None</td>
<td>0.101569</td>
<td>0.025295</td>
<td>39.5261</td>
</tr>
<tr>
<td>None</td>
<td>1 Per Packet</td>
<td>0.391185</td>
<td>0.101949</td>
<td>1798.61</td>
</tr>
<tr>
<td>Read</td>
<td>1 Per Packet</td>
<td>0.108103</td>
<td>0.0269311</td>
<td>50.1123</td>
</tr>
</tbody>
</table>

Next consider the same results tabulated in Table XVI for an 8 x 8 open region centered in a 16 x 16 network. Operations with one partial barrier per packet sent over the region had 6.44 times the achieved load over the use of full acknowledgements. Furthermore, the use of barriers and blocking pending read acknowledgements produces a 39.96% increase in the achieved load over the full acknowledgement case. We start to see a divergence between the achieved load and bisection bandwidth improvements. The reason for this is that the bisection bandwidth includes the bandwidth used by the acknowledgement packets on writes, including those from outside the source region, which are not required on writes when INBs are used. In contrast, the achieved load is computed only on packets originating within the region, and without including
acknowledgement packets. Hence INBs also reduce the load on the network for supporting sequential consistency.

<table>
<thead>
<tr>
<th>Acknowledge</th>
<th>Barriers</th>
<th>Bisection</th>
<th>Achieved Load</th>
<th>Latency</th>
</tr>
</thead>
<tbody>
<tr>
<td>Read, Write</td>
<td>None</td>
<td>0.0363827</td>
<td>0.0181619</td>
<td>33.7144</td>
</tr>
<tr>
<td>None</td>
<td>1 Per Packet</td>
<td>0.117433</td>
<td>0.116998</td>
<td>903.159</td>
</tr>
<tr>
<td>Read</td>
<td>1 Per Packet</td>
<td>0.0382499</td>
<td>0.025419</td>
<td>42.5394</td>
</tr>
</tbody>
</table>

Table XVI. 8 x 8 Open Region Acknowledgements Versus Barriers

Table XVII summarizes the same results for a 4 x 4 open region in the center of a 16 x 16 network. The use of one barrier per operation without waiting for acknowledgments produces 9.22 times the achieved load over operation in which the processors block pending acknowledgement of each operation. Also, the use of one barrier per packet and blocking pending read replies yields a 47.06 % increase in achieved load over the full acknowledgement case.
Table XVII. 4 x 4 Open Regions Acknowledgements Versus Barriers

<table>
<thead>
<tr>
<th>Acknowledge</th>
<th>Barriers</th>
<th>Bisection</th>
<th>Achieved Load</th>
<th>Latency</th>
</tr>
</thead>
<tbody>
<tr>
<td>Read, Write</td>
<td>None</td>
<td>0.01067</td>
<td>0.021229</td>
<td>26.763</td>
</tr>
<tr>
<td>None</td>
<td>1 Per Packet</td>
<td>0.04903</td>
<td>0.195775</td>
<td>523.79</td>
</tr>
<tr>
<td>Read</td>
<td>1 Per Packet</td>
<td>0.01178</td>
<td>0.031219</td>
<td>34.118</td>
</tr>
</tbody>
</table>

Finally, the 2 x 2 open region centered in a 16 x 16 mesh is tabulated in Table XV. Operation in which each packet sent is followed by a barrier yields 14.54 times the achieved load over waiting for acknowledgements for each packet sent. Furthermore, the use of barriers after each operation and waiting for read acknowledgements (replies) yields 65.22% increase in achieved load over waiting for acknowledgement of all packets.

The ratios discussed above a summarized in Table XIX, and graphed in Figures 28 and 29. Figure 28 plots the ratio of the achieved load when the partial barriers are run over the source region after each processor injects a packet, and when no acknowledgements are in use, to the baseline case of the processors blocking pending acknowledgment of both read and write packets. Figure 29 plots the ratio of the achieved load of the case with barriers after every packet, and the processor waiting for read acknowledgements, to the same baseline case. Note that as the open region size becomes smaller, the performance gain from using partial barriers increases.
Table XVIII. 2 x 2 Open Regions Acknowledgements Versus Barriers

<table>
<thead>
<tr>
<th>Acknowledge</th>
<th>Barriers</th>
<th>Bisection</th>
<th>Achieved Load</th>
<th>Latency</th>
</tr>
</thead>
<tbody>
<tr>
<td>Read, Write</td>
<td>None</td>
<td>0.00289</td>
<td>0.02307</td>
<td>23.305</td>
</tr>
<tr>
<td>None</td>
<td>1 Per Packet</td>
<td>0.02086</td>
<td>0.33538</td>
<td>285.35</td>
</tr>
<tr>
<td>Read</td>
<td>1 Per Packet</td>
<td>0.00361</td>
<td>0.03811</td>
<td>27.53</td>
</tr>
</tbody>
</table>

Table XIX. Ratios of Achieved Load Versus Full Acknowledged Case

<table>
<thead>
<tr>
<th>Region Size</th>
<th>Side Of Region</th>
<th>Ratio of Achieved Load to the Acknowledged Read and Writes without Barriers for:</th>
<th>Barriers &amp; No Acknowledgements</th>
<th>Barriers &amp; Read Acknowledgements</th>
</tr>
</thead>
<tbody>
<tr>
<td>2 x 2</td>
<td>2</td>
<td>14.53762</td>
<td>1.652177</td>
<td></td>
</tr>
<tr>
<td>4 x 4</td>
<td>4</td>
<td>9.222272</td>
<td>1.470617</td>
<td></td>
</tr>
<tr>
<td>8 x 8</td>
<td>8</td>
<td>6.441947</td>
<td>1.399578</td>
<td></td>
</tr>
<tr>
<td>16 x 16</td>
<td>16</td>
<td>4.030401</td>
<td>1.064681</td>
<td></td>
</tr>
</tbody>
</table>
Figure 28. Plot of Achieved Load Ratio for Barriered No Acknowledgements
Figure 29. Plot of Achieved Load Ratio for Bariered and Read Acknowledgements
5: Conclusions and Future Work

5.1 Conclusions

5.1.1 Conceptual Contributions

The contributions of this thesis have been both conceptual and experimental. The conceptual contributions include the following.

- Virtual Interconnection Networks are distinct from plies. Earlier work assumed that plies contain routes between all valid source destination pairs. This assumption now matches our new concept of a Virtual Interconnection Network. In the language of Graph Theory, routes are simply walks on the queue dependency graph (Chartrand, 1985). Hence plies can be viewed as unions of arbitrary sets of non-disjoint routes.

- Virtual interconnection networks can be viewed as a set of plies, or alternatively it can be decomposed into a set of plies.

- The Total Embedding Property requires that a route traverse queues in exactly one ply. This guarantees that if a wave of barrier markers sweeps a ply, it traverses all routes that use resources in the ply.

- Plies can be embedded within physical plies, and pickets used to isolate the queues in the ply from those adjacent in the physical ply, but not in the ply in question. Since a full set of both entry and exit pickets requires two single bit flags on each queue, the implementation cost is very low.

- The use of open regions, which allow the set of source processors to be different from the set of destination processors.
5.1.2 Experimentally Verified Observations

The experimental aspects of this work include the following.

- The characteristics of single-ply VINs were tested and found consistent with previously published work by Stamatopoulos and Solworth.
- The first experimental confirmation of multi-ply VINs and barriers using corner waves, as proposed by McShane, were performed.
- Closed regions were confirmed to behave and have the same performance characteristics to small networks of the same size.
- When a network of single ply VINs is tiled with open regions, one tile per VIN, which ran partial INBs, the performance was found to approach the performance of an ideal network as the open regions became smaller. In the case of multi-ply VINs, performance degraded as more tiles and plies were added, and this is believed to be due to the larger number of queues competing to route over the external links.
- Time stamp based wire arbitration was found to only partially offset the problems at the external links in the multi-ply VIN case.
- The use of open regions and partial barriers were found to be very efficient for implementing sequential consistency where the source processors are the source of memory operations, and the destination processors are the locations of memory. Experiments were performed assuming an equal number of read and write requests, distributed uniformly through the network. When the use of barriers is compared to the popular technique which allows one outstanding operation per processor, it can achieve as much as a factor of 14 times the achieved load if no replies are used with
barriers, or as much as a 65% increase in achieved load if processors still block pending replies to read request in conjunction with using barriers.

5.2 Future Work

This thesis focused on two dimensional mesh topologies. Since one of the key insights in this work was to ignore network topology and focus on the queue topology, this work should be extended to other network topologies. It would be particularly powerful to develop rules on when and how pickets can create regions in arbitrary network topologies, such as those that might arise in an office local area network.

The application of these techniques to general networking and distributed computing could show real promise. This is particularly true in the database and transaction processing areas, where consistency of records requires exact control of the timing of updates to the systems, in order to prevent their being left in an inconsistent state.

It would be interesting to see if INBs can be applied to the Termination Detection Problem. In this problem, the base assumptions about the network are different, and it is legal to roll back part of the synchronization. This is very different than INBs and hence further investigation of the two together could yield interesting results for both subjects.

There are some potential applications of advanced mathematics to extend the work in this thesis. The application of queuing theory to the analysis of packet latency in a non-empty network, and the application of topological graph theory to route embedding seem promising.
5.3 Applications

When work was first started for this thesis, the obvious application of the results was for the design of an interconnection network for a multiprocessor based supercomputer. While the current trends in parallel and distributed computing appear to have drifted toward distributed and cluster computing, the construction of large multiprocessors cannot yet be entirely dismissed.

These techniques are not limited to supercomputer design. Consider the viability of a small network hub for a local area network, which could link to a collection of workstations via dedicated Ethernet cables. The hub could contain a structure of queues similar to the network model in this thesis. The workstations could inject barrier markers as packets addressed to the hub itself, and the hub could use partial INBs to ensure barrier order packet arrive at their destinations within the hubs local network. Such a product could surely be sold and used in transaction processing environments.

Furthermore, consider a large digital telecommunications network, using proprietary hardware. The functionality for these partial barriers could be easily built into the network and used not only to provide congestion control and fairness, but also to ensure barrier ordering for transaction security.
Cited Literature


NAME: Bryan Barney Reagan

EDUCATION: Mechanical Engineering, Valparaiso University, Valparaiso, Indiana, 1986-89

B.S., Mathematics and Computer Science, University of Illinois at Chicago, Chicago, Illinois, May 1991

M.S., Electrical Engineering and Computer Science, University of Illinois at Chicago, Chicago, Illinois, July 1995

Ph.D., Computer Science, University of Illinois at Chicago, Chicago, Illinois, July 2004


Gold Key Honor Society, University of Illinois at Chicago, 1991

Phi Kappa Phi Honor Society, University of Illinois at Chicago, 1991

PROFESSIONAL MEMBERSHIP: American Mathematical Society (AMS)

Association for Computing Machinery (ACM)

Institute of Electrical and Electronics Engineers (IEEE)

Mathematical Association of America (MAA)

Phi Mu Alpha Sinphonia


Co-op Engineer, Technology Division, Inland Steel Company, East Chicago, Indiana, 1988-89.

Lab Assistant, Material Science Lab, Valparaiso University, Valparaiso, Indiana, 1989.

Teaching Assistant, University of Illinois at Chicago, Chicago, Illinois, 1997 - 1998 Academic Year

Teaching Associate, University of Illinois at Chicago, Chicago, Illinois, 1998 - 1999 Academic Year.


PUBLICATIONS:


Efficient Implementation of Partial Integrated Network Barriers

Bryan Barney Reagan, Ph.D.
Department of Computer Science
University of Illinois at Chicago

Integrated Network Barriers are a technique for synchronizing a set of processors (sources and destinations of packets) in a packet switched interconnection network, which incorporates operation completion with synchronization. They use dedicated packets, called barrier markers, which traverse the network based on queue topology, as derived from the Queue Dependency Graph.

In a partial barrier, only some subset of the processors participates in the barrier. Source processors participate in the barrier and generate packets, while destination processors receive packets, segregated in barrier order. A given processor may be a source, a destination, both, or neither. Partial barriers can be implemented by marking queues with flags called pickets. Entry pickets mark as queue to be ignored when checking predecessors for barrier propagation, and exit pickets mark queues to prohibiting barrier propagation into the queues.

Pickets can be used to create sub-networks called regions. By using both entry and exit pickets, a closed region is created, which acts in isolation from the rest of the network, and behaves like a smaller network. Using only entry pickets, an open region can be created in which only the set of processors with participate in the barrier may send packets, but those packets arrive at the destination processors in barrier order, and only the performance cost of synchronizing the source processors is incurred. Open regions can improve network performance, and also can efficiently implement sequential consistency.