A Routing Network for the Grid Processor Architecture

Vincent Ajay Singh* Stephen W. Keckler Doug Burger

Computer Architecture and Technology Laboratory
Department of Computer Sciences
*Department of Electrical and Computer Engineering
Tech Report
The University of Texas at Austin
cart@cs.utexas.edu — www.cs.utexas.edu/users/cart

ABSTRACT

Abstract

This is a technical report on the proposed in–grid network/router for the grid architecture. This router architecture demonstrates a lightweight and robust solution for on-chip operand networks, and incorporates backpressure and dynamic routing techniques. A representative design was implemented in Verilog to test the functionality, and implemented at the circuit level in order to examine worst case delay. The results show that, in the common case, operands will be available to the next processor without incurring anything but transmission delay. Worst case delay estimates for the control logic and the transmission delay are presented for 100nm and 35nm technologies.

1 Introduction

For the last twenty years, the processor community has been able to sustain an annual performance growth rate of 60%. For the last decade, much of this performance increase has come from the ability to ramp up the clock rate at incredible speed (from 33MHz in 1990 to 2GHz in 2001).

Unfortunately, soon it will no longer be advantageous to increase the clock rate faster than device technology [1]. One on the reasons for this is the increasing ratio of wire delay to gate delay. As the feature size decreases, the relative speed of communication increases, making today's superscalar architectures more difficult to design due to their reliance on operand forwarding and larger globals structures.

One architecture proposed to address these problems is the Grid Processor Architecture (GPA)[2]. This architecture is designed to allow faster clocks and higher instruction level parallelism (ILP) than today's modern superscalar designs. The GPA consists of multiple arrays of small functional units, each of which has its own instruction memory, execution unit, and operand router to interface with a point-to-point interconnection network (see Figure 1). These arrays are controlled by a single thread of control that maps large predicated blocks onto the array. Once the instructions are mapped onto the array, the computations proceed in dataflow order, with each computational unit executing its instruction when the required operands arrive. This model of execution allows out-of-order execution while eliminating the dependence on a central instruction issue window. The point-to-point operand network also replaces the larger broadcast network needed to achieve bypassing in conventional architectures, which would have poor performance as relative wire speed decreases due to the long wires and high degrees of fan-in/fan-out required to bypass operands to all stages in a superscalar design. The rest of this paper will focus on the design on this operand network and the corresponding routers.

One of the key factors in the performance of the GPA is the delay of the operand network. In order to operate efficiently, the architecture calls for a very low latency (under a quarter of a cycle) routing network [2]. The latency will depend largely on the distance the operands must travel as well as the richness of the interconnect network. A larger distance will mean going through more intermediate routing nodes. A richer interconnect will decrease this number, but make each hop more costly. Another factor that must be considered is the amount of contention and the flow control protocol overhead extant in the network design. Since latency is critical, delays other than straight transmission time must be kept to a minimum.

The rest of this paper is structured as follows. Section 2 describes the proposed architecture for the router and network design, as well as some of the tradeoffs associated with its design. Section 3 presents some performance results, while section 4 gives possible alternate designs. We then conclude in Section 5.

2 Architecture

2.1 Basic Architecture

Dominant Design Constraints While one of the GPA's strongpoints is the existence of only short point to point wires between nodes, it could also be a limiting factor since all communication must go through this network. Accordingly, one of the dominant design constraints for this network is low operand transmission latency. Ideally, the only delay would be the transmission time from the producing node to the consuming node. This would require a full crossbar switch to directly connect all the nodes to all the others. Unfortunately, such a network would scale poorly, and

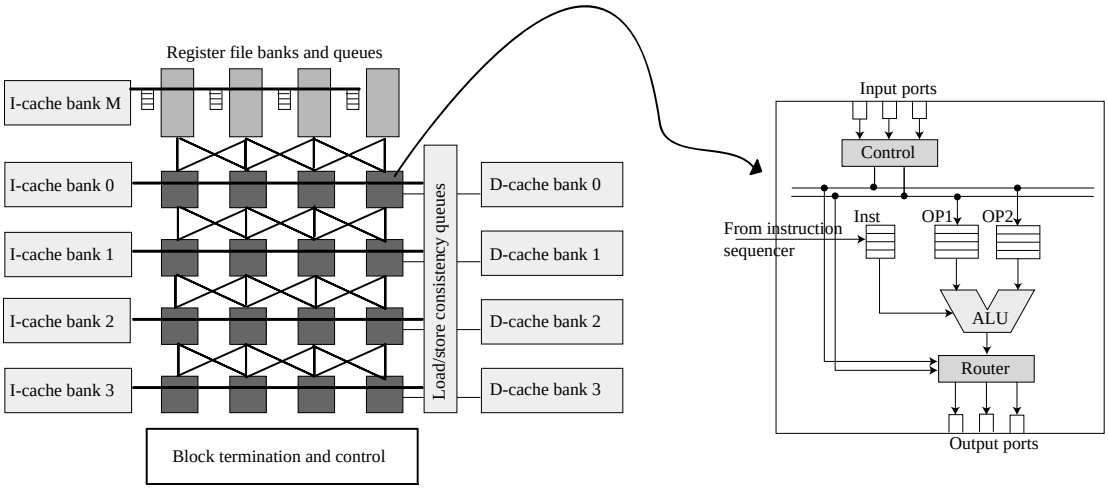

Figure 1: High-level Grid Processor organization

would be to slow to accommodate the grid's needs. Instead, the common case must be made fast and direct, while the rare cases may take longer. This introduces the flow control mechanism as one of the dominant constraints as well.

Reserving Channels in Advance In order to satisfy these primary design constraints, the network design incorporates the idea of reserving network paths for operands a cycle in advance, similar to the Flit Reservation Flow Control mechanism proposed in [3]. In order to do this, two networks are implemented. One for control data and one for operand data. The control data arrives in advance of the operand data, and reserves a path (if one will be available) for the operand data, thereby taking the routing and decision making logic off the critical path. If no path will be available (due to contention through the node) a buffer slot is reserved for the incoming operand. This information is available a cycle in advance of the operand data, since the destination is encoded into the instruction itself and can be processed during the time taken to produce the operand (see Figure 3). With this advance knowledge, circuit techniques (such as domino logic) can be used to increase the speed of the operand transmission. It is also possible to use this advance knowledge to save switching power, by disabling unused links.

Flow Control Advance knowledge of incoming operands also allows an efficient flow control mechanism. With knowledge of how many open buffer slots are available and the communication latency to immediate producer nodes, a simple throttling mechanism can be implemented. If we assume that it takes one cycle to send an operand from one node to the next, we would assert the throttling signal when a control flit arrives for a buffer which will only have two slots open when the data arrives. We need one slot for the operand arriving on the next cycle, and one for the operand that could be sent while the throttle signal is traveling back to the producing node. Once the producing node receives the throttle signal, it will cease to transmit data until the signal is taken away. This way, in the common case, a producing node can send the operand without the need to receive an acknowledgment, since it is guaranteed that storage space will be available at the consumer node if there is contention on the routing path. Only in the uncommon case is any backward information required.

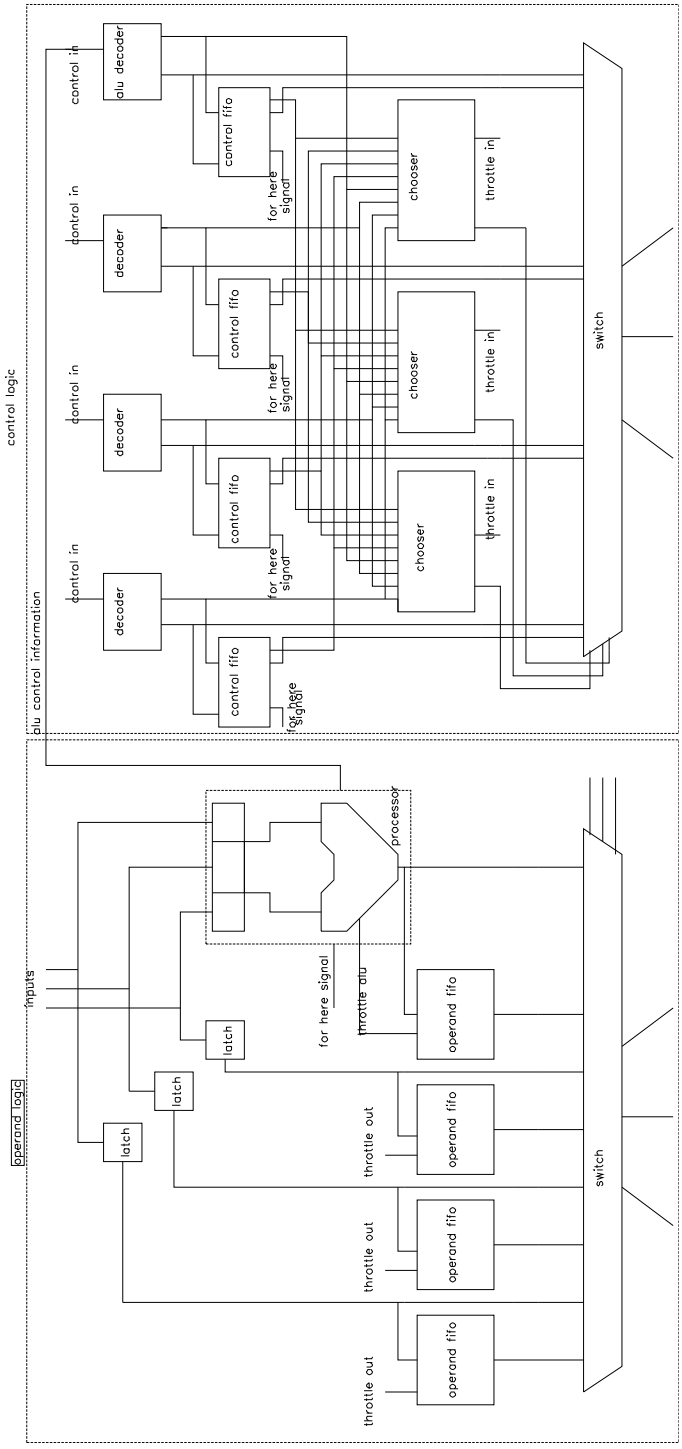

Figure 2: The operand buffers are FIFOs used to store operands that cannot be immediately sent when they arrive because of contention for the link or because of a throttled output channel. The control buffers are FIFOs holding the control data for the operands which must be buffered. The operand and control switches are multiplexors that control which data is put on which output channels. There is one chooser per output channel, and it is the module tasked with deciding which input channel to route to the chooser's output channel on the next cycle. The choosers also generate control information for the various buffers. The module which decodes the incoming control data is the decoder. There is one decoder per incoming channel, and it must decode the destination information encoded in the control data and produce the new control data to be forwarded to the next destination (if the data is destined for some other node). This module produces either new control data, or a flag telling the processor that the data coming on the next cycle is for it. The delay elements (latches) are on the bypass paths and are to allow the control logic to have sufficient time to process the control information.

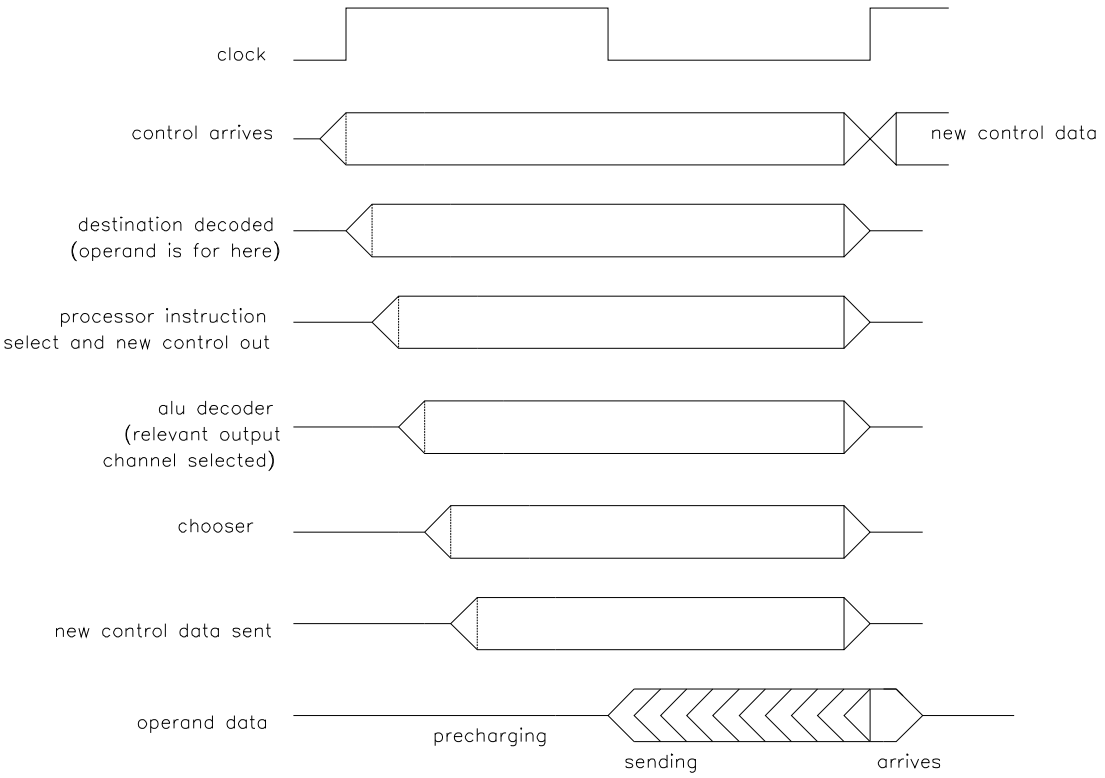

Figure 3: This is a timing diagram illustrating the longest path through the router. An arriving control packet is paired to an operand destined for this node, and the processor wishes to use this data immediately. In this case, the arriving control must pass through the input channel decoder to determine that the operand is destined for this node. This fact must be forwarded on to the processor, which responds with control information corresponding to the new operand to be sent next cycle. This data must be decoded to determine which output channel the operand will travel on, and then the chooser must generate the switch control signals and forward on the new control packet..

Example Architecture A diagram of the router architecture is shown in Figure 2, assuming a fan-out and fan-in of three, which yields the GPA array depicted in Figure 1. Figure 2 shows six components. They are the operand buffers, the control buffers, the operand switch, the control switch, the chooser, and the decoder, and some delay elements. The operand buffers are FIFOs used to store operands that cannot be immediately sent when they arrive because of contention for the link or because of a throttled output channel. The control buffers are FIFOs holding the control data for the operands which must be buffered. These buffers are typically out of phase with each other, since the control data leads the operand data. The operand and control switches are multiplexors that control which data is put on which output channels. There is one chooser per output channel, and it is the module tasked with deciding which input channel to route to the chooser's output channel on the next cycle. The choosers must also generate control information for the various buffers, since they decide which data can go and which must be kept. The module which decodes the incoming control data is the decoder. There is one decoder per incoming channel, and it must decode the destination information encoded in the control data and produce the new control data to be forwarded to the next destination (if the data is destined for some other node). Accordingly, this module produces either new control data, or a flag telling the processor that the data coming on the next cycle is for it. The delay elements are on the bypass paths and are to

allow the control logic to have sufficient time to process the control information. Otherwise the operands would soon catch up to the control data, and would be mishandled.

Timeline An example timeline is shown in Figure 3. This timeline depicts the longest delay through the router, which is the case where the operand arriving is destined for this node and the processor is going to produce a new operand in response. The incoming control must go through the incoming channel's decoder, the wake-up logic, the alu-decoder, the chooser, and finally the control switch. The operand itself gets routed directly to the processor when it arrives, and the resulting operand goes out on the next cycle, according to its destination field (produced the cycle before by the control).

Multicasting Abilities An advantage of this architecture is that it can be easily extended to allow multicasting. All that this requires is adding duplicates of a couple of the functional units (such as the control FIFOs) and some minor glue logic. This allows the router to multicast operands while only incurring an extra muxing level of delay.

2.2 Implications and tradeoffs

Fan-in and Fan-out There are some implications of the design as presented in Figure 2 that should be discussed. One is the in/out degree of the router design. The richness of the interconnect can have a large impact on the performance of the grid processor, since a low direct connectivity limits the compiler's scheduling of instructions as well as increasing the average number of hops that must be traversed while going from producer to consumer. The design presented here allows the fan-in and fan-out of the router to be increased by replicating functional units and using them in parallel. The only units that must be redesigned (and thereby possibly slowed) are the output switches and the chooser module, since they must be able to choose between all possible inputs and outputs.

Buffering Options There are also tradeoffs associated with the operand buffer locations. Figure 2 shows separate buffers on the input side of the router. This placement was chosen because it allows only one producing node to be throttled if the corresponding buffer becomes full, allowing the other producers to continue to operate unaffected. However, this orientation obscures the order of operand arrival from the router decision making logic. This logic currently uses a simple priority scheme that merely looks at the head of the FIFO's and determines which operand to transmit. Since an operand which arrives later on a less used input channel might go before an older entry in a full buffer, the operands are not necessarily transmitted in order of arrival. This could be fixed in the implementation of the chooser module, at the cost of some additional complexity.

Another ramification of separate buffers (vs. one larger consolidated buffer) is buffer utilization. Since the individual buffers would be smaller than one combined buffer, this design will throttle earlier in the case where only one input channel is active (since the other input buffers would be unused but unavailable). Besides the throttling implications of a single buffer (which would have to throttle all incoming paths, but which would do so less often) the number of ports required on a unified buffer would make the structure much slower. Further investigation is needed to determine the ideal buffer size vs. throttling frequency tradeoff.

Link Priority In this implementation of the design, a simple static priority scheme has been assumed in the design of the chooser module. This decision must be examined, since it introduces

the possibility of starvation into the network. A simple round robin scheme would alleviate this problem, but the optimal rule needs to be determined. It may be that prioritizing the bypass paths would yield a better result.

Multicast Implementation Also, this implementation of the multicast abilities trades speed for hardware and power savings, since much of the hardware is replicated. If this is unacceptable or unneeded, the same hardware could be double pumped and muxed to get the same result in twice the time.

3 Performance Discussion

To verify the router design, it was first implemented and tested in the Verilog hardware description language. For the purposes of the design, the processing unit was modeled as a data consumer/producer which produced random data values and destinations.

After designing and verifying the architecture in Verilog, schematics for the major functional units of the router were implemented and verified against the Verilog models. This implementation was meant to be a worst case estimate, so most logic used was purely combinational, and the transistor sizing has not been optimized at all.

These schematics were then extracted to SPICE models, and analyzed for feature sizes of 100nm and 35nm. As is evident from the motivation of the grid processor, the operand data bus delays must also be taken into account, so a range of bus lengths and wire widths were analyzed using an in-house wire delay analysis tool developed by Vikas Agarwal . The lengths were based on empirical studies of various ALU areas in terms of transistor widths [4]. The wire delays are presented in Figure 4, and the various component delays are presented in Figure 5.

As the worst case control path is through the destination decode, wakeup logic, ALU destination decode, and then the chooser, the worst case control delays are about 460ps and about 150ps in 100nm and 35nm technology respectively. It should be noted that the speed of the wire transmission in 100nm technology is much slower relative to the combinational speeds than it is in 35nm technology.

4 Design Alternatives and Future Work

There are a number of assumptions that have been made thus far that should be mentioned, since there are other viable (and possibly more efficient) ways to accomplish the same things. These options need to be evaluated in order to get a more efficient design.

Reducing the Bypass Latency One of the most limiting design decisions made was the decision to delay the operand bypass paths in order to allow the control logic time to complete. This limits the maximum rate that an operand can travel through the network to one node per cycle. After examining the performance results, it can be seen that the logic delay is still much longer than the transmission delay, making this design decision appear to be a poor one. Further examination of the performance numbers will show that the majority of the time spent in the control logic is spent calculating the new control data packet. In the bypass case, this information can be precalculated in parallel with the creation, since the routing is deterministic.

By precalculating the control data, all subsequent control through bypassing nodes must only pass through the chooser logic (to examine whether there is contention on that link or not). The

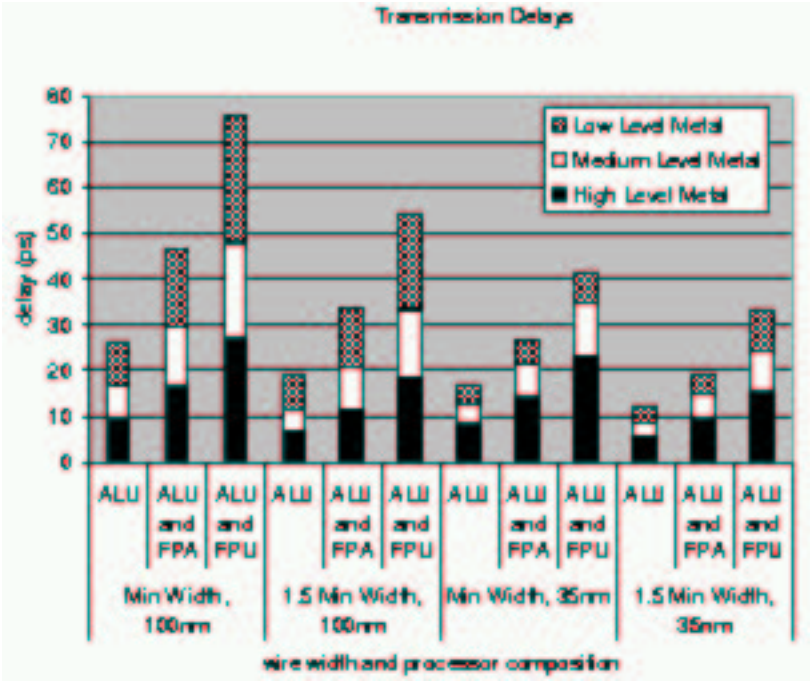

Figure 4: Transmission delays for communication along the length of a square integer ALU, an ALU and floating point adder, or an ALU and full floating point unit on various levels of metal in 100nm and 35nm technologies.

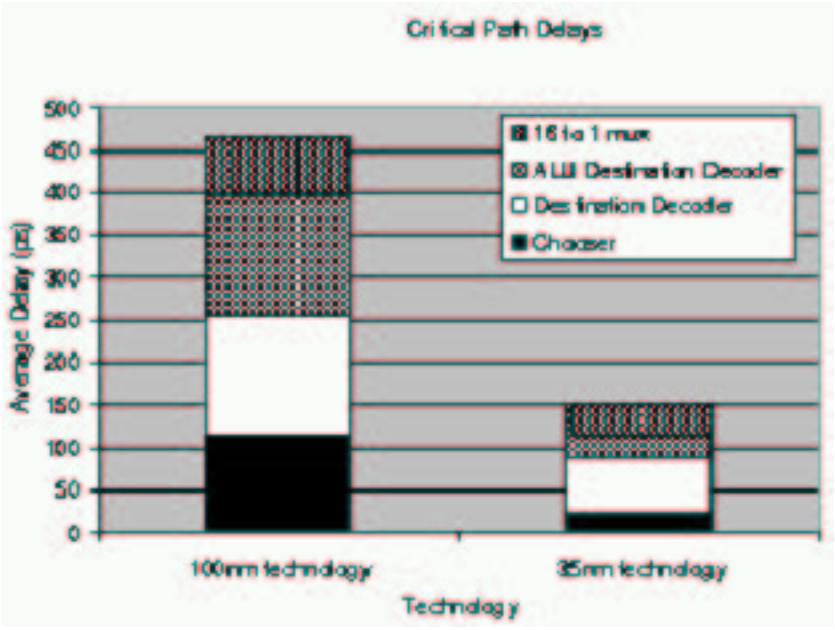

Figure 5: Average component delays for critical path in 35nm and 100nm technology. The delays reported are the average of the rising and falling delays.

time required to make this decision should, after the logic is optimized, be small enough to hide by speeding up the control transmission path by widening the wires or going up a metal level. This would allow the routers to output stored and bypassing data more than once per cycle, with the full control logic penalty only paid for new operand data packets. This would speed up the bypass network considerably.

Multi-cycle ALU operations During the evaluation of this architecture, it was assumed that the processor had uniform latency operations. This is not likely to be the case in a modern implementation, so multi-cycle operations must be handled as well. Multi-cycle operations can be handled by simply delaying the control data from the processor by n cycles, where n is the latency of the operation, and handling the case where more than one operand is produced at the processor in one cycle. This can be done either by replicating the ALU-decoders and widening the chooser modules and the output switches, or by prioritizing the processor outputs and storing operands.

The tradeoffs here are not clear cut, since the result depends heavily on how often the processor will be completing more than one operand in a cycle. This can only be answered by examining applications on the GPA.

Static vs. Adaptive Routing Another important assumption that has been made deals with the routing decisions made at each node. The current design makes routing decisions based solely on whether the destination node is on the right, left, or directly below the current node. This is a static and deterministic routing decision which does not take into account any link or node contention. The efficiency of the network could possibly be increased by accounting for these factors and moving to a more adaptive routing scheme. This decision should be made on the basis of the contention measurements in the network.

Relative vs. Absolute Addressing There are also a couple of different addressing modes that can be used within a grid, relative addressing or absolute addressing. The current design assumes relative addressing, so a message arriving for address (0,0) would be at its destination. In this mode, the routing direction of a message can be determined by a comparison to zero (less than means left, equal to means here, and greater than means right). However, this also means that every stage must modify the destination fields at every node, incorporating some addition/subtraction logic into the router. The alternative addressing mode (absolute addressing) would require each node to know its address, and a subtract and compare to zero would still be required to determine the destination of the message. The tradeoffs here are not clear cut, and more investigation needs to be done to determine the better alternative.

In Order vs. Out of Order Buffers Another artifact of the present design is that the buffers are implemented as FIFOs, which was assumed for simplicity. This assumption has the consequence that a single output channel blocked could stall many operands that could otherwise proceed. This situation is analogous to an in-order pipeline stall in a processor, where instructions which could be issued must stall because of a previous instruction. The operand buffers could be implemented in a manner which would allow non-blocked operands to issue ahead of operands which must stall due to resource conflicts, avoiding this problem.

Buffer Consolidation There is also the possibility of using the operand buffers existent in the processing node as the router buffer. This would incur the problems of a single larger operand storage buffer, but should be looked at if the size of the router logic becomes a concern.

Variable Length Operands Another possible optimization that should be examined in the case of variable length data values is the possibility of scheduling and reserving links more than one cycle in advance. This idea is much closer to the architecture addressed in [3].

Network Traffic vs. Contention Additionally, there is a possibility of trading higher network traffic for faster transmission times when the operands are not going to an immediately connected node. Here single destination operands would be multicast to different immediate nodes under the assumption that, if one of the nodes is blocked for some reason, the other would not be. Although this introduces a lot of extra network traffic, increasing the chance that a node will be blocked in the first place, as well as requiring nodes to be able to detect that they have already received a copy of the operand, this possibility should be examined.

5 Conclusions

We have presented an architecture for a lightweight and robust on-chip network geared towards the grid processor architecture. This architecture was analyzed for worst case timing in 100nm and 35nm technologies. The results show that the control path delays will be less than 450ps in 100nm technology, and 150ps in 35nm technology. The next step in evaluating this design is to explore some of the design alternatives and tradeoffs mentioned under real world applications running on a GPA, as well as investigating the low level implications of some of the assumptions which have been made and the conclusions drawn from the benchmark characterization results. The most pressing topics include improving the bypass speed, deciding on the right fan-in and fan-out degrees for applications, finding an effective priority scheme for the choosers, and adding multi-cycle operations to the model.

References

[1] Vikas Agarwal, M.S. Hrishikesh, Stephen Keckler, and Doug Burger. Clock Rate Versus IPC: The End of the Road for Conventional Microarchitectures. In *Procedings of the 27th Annual International Symposium on Computer Architecture* , June 2000.

[2] Ramadass Nagarajan, Karthikeyan Sankaralingam, Doug Burger, and Stephen W. Keckler. A Design Space Evaluation of Grid Processor Architectures. In *Proceedings of the 34th Annual International Symposium on Microarchitecture*, December 2001.

[3] Li-Shiuan Peh and William J. Dally, Flit-Reservation Flow Control, In *Proceedings of the 6th International Symposium on High-Performance Computer Architecture*, January 10-12, 2000. pp. 73-84.

[4] S. Gupta, S.W. Keckler, and D.C. Burger. Technology Independent Area and Delay Estimations for Microprocessor Building Blocks. Technical Report TR-00-05, University of Texas at Austin, Computer Science Department, February 2001.