Packet Chaining: Efficient Single-Cycle Allocation for On-Chip Networks

George Michelogiannakis
Electrical Engineering Dept.
Stanford University
Stanford, CA 94305
mihelog@stanford.edu

Nan Jiang
Electrical Engineering Dept.
Stanford University
Stanford, CA 94305
njiang37@stanford.edu

Daniel Becker
Electrical Engineering Dept.
Stanford University
Stanford, CA 94305
dub@stanford.edu

William J. Dally
Electrical Engineering Dept.
Stanford University
Stanford, CA 94305
dally@stanford.edu

ABSTRACT

This paper introduces *packet chaining*, a simple and effective method to increase allocator matching efficiency and hence network performance, particularly suited to networks with short packets and short cycle times. Packet chaining operates by chaining packets destined to the same output together, to reuse the switch connection of a departing packet. This allows an allocator to build up an efficient matching over a number of cycles like incremental allocation, but not limited by packet length. For a 64-node 2D mesh at maximum injection rate and with single-flit packets, packet chaining increases network throughput by 15% compared to a highly-tuned router using a conventional single-iteration separable iSLIP allocator, and outperforms significantly more complex allocators. Specifically, it outperforms multiple-iteration iSLIP allocators and wavefront allocators by 10% and 6% respectively, and gives comparable throughput with an augmenting paths allocator. Packet chaining achieves this performance with a cycle time comparable to a single-iteration separable allocator. Packet chaining also reduces average network latency by 22.5% compared to a single-iteration iSLIP allocator. Finally, packet chaining increases IPC up to 46% (16% average) for application benchmarks because short packets are critical in a typical cache-coherent chip multiprocessor.

Categories and Subject Descriptors

B.4.3 [**Hardware**]: Input/output and data communications—*Interconnections*; C.1.2 [**Computer systems organization**]: Multiple data stream architectures—*Interconnection architectures*

General Terms

On-chip networks, allocation, iterations, packet chaining

1. INTRODUCTION

Networks-on-chip (NoCs) have been developed to efficiently serve the communication requirements of large-scale systems with hundreds or thousands of logic blocks [5, 16]. NoCs can consume considerable amounts of area and power, as well as affect application execution time [23]. Therefore, much research has focused on improving NoC efficiency.

The performance of a NoC is extremely sensitive to the matching efficiency of the allocators used to allocate switch ports and virtual channels (VCs) in the NoC's routers. Due to their complexity, allocators are in the critical path of many routers [21, 2]. Therefore, past research has examined the trade-off between matching efficiency and cycle time [2].

Many NoCs use separable allocators [6] which use separate input and output arbiters in sequence to perform allocation. iSLIP separable allocators [18] use round-robin arbiters and update the priorities of each arbiter when it generates a winning grant. Separable allocators are often used because they can operate within an aggressive cycle time. However, making arbitration decisions independently at each port degrades matching efficiency. While a separable allocator's matching efficiency can be increased by performing multiple iterations, this is typically not feasible within a single clock cycle [2, 6]. Wavefront [25] and augmenting paths [7] allocators also increase matching efficiency by guaranteeing maximal and maximum matchings, respectively. These guarantees increase delay and cost [2, 10], making such allocators infeasible within a tight timing budget.

To provide the efficiency of multi-iteration allocation without extending cycle time, past work has proposed pipelined [9] and incremental [20] allocations. In both schemes, allocation extends over multiple cycles, during any of which new requests can be added. In pipelined allocation, results are only available at the end of the last iteration. In contrast, incremental allocation makes the results of each iteration available, such that intermediate grants can be generated. Incremental allocation has been implemented using a separable, single-iteration allocator and holding resources granted for the duration of a packet [20, 13]. However, both schemes provide no benefits to single-flit packets, and small benefits to short packets.

Many systems, such as typical cache-coherent chip multiprocessors (CMPs), send primarily short packets. For instance, 53% of the packets in the applications we simulated in this paper were single-flit and received no benefit from incremental allocation. Short packets are important because they transfer time-critical control messages. Short packets stress allocators due to increasing the number of head flits which correspond to new allocator requests even with incremental allocation. Long packets are affected by short packets because the allocation problem remains large causing inefficient matchings, and also because long packets can be blocked behind short packets in the same VC. In addition, long packets can starve other packets [20, 13].

In this paper we introduce *packet chaining* [19], a method for improving allocation efficiency for iterative allocators that is particularly suited to networks with short packets and short cycle times. Packet chaining operates by chaining packets, potentially from different inputs, destined to the same output together, facilitating reuse of a departing packet's switch connection. Even with uniform random traffic, a significant number of packets request the same output at every router. This allows an allocator to build up an efficient matching over a number of cycles like incremental allocation [20], but not limited by packet length. To provide limited fairness, we add starvation control which either operates by releasing a connection after it has been held for a predetermined number of cycles, or by releasing a connection when a competing packet has been blocked for a predetermined number of cycles.

Packet chaining provides excellent matchings, especially with single-flit packets. Packet chaining enables long packets to be divided into shorter ones to avoid performance degradation from long packets due to constant-sized buffers, without loss of allocation efficiency. Packet chaining is implemented by adding a separate packet chaining (PC) allocator, and thus doubles the area and power for allocation. The allocation timing path is lengthened only marginally, since the PC and switch allocators operate in parallel. This is in contrast with wavefront which is typically used to increase allocation efficiency but requires up to 1.25× more area, 1.5× more power and 36% higher delay compared to packet chaining in the mesh. This is intensified for high-radix routers such as the flattened butterfly (FBFly) [11], where wavefront requires up to 1.35× more area, 3× more power and 37% higher delay than packet chaining.

We evaluate packet chaining on an 8×8 mesh and on a 64-terminal FBFly. Packet chaining increases the allocation efficiency of a single-iteration separable allocator to be comparable to or higher than more expensive and slower allocators. For single-flit packets at maximum injection rate, packet chaining increases network throughput by 15% compared to a highly-tuned router with incremental allocation using a single-iteration iSLIP switch allocator [18]. Packet chaining also outperforms multi-iteration iSLIP allocators and wavefront allocators [25] by 10% and 6% respectively, and gives comparable throughput with an augmenting paths allocator [7]. Multiple-iteration iSLIP, wavefront, and augmenting paths allocators are typically infeasible within an aggressive cycle time [10]. Because of their cost, wavefront allocators can only be reasonably considered for low-radix routers or full-custom implementations. Throughput at maximum injection rate is a more representative measure than saturation throughput because throughput-limited systems operate the network beyond saturation. Without elaborate throttling, it is very difficult to hold a network at the point of saturation.

Furthermore, packet chaining reduces average network latency by 22.5% compared to a single-iteration iSLIP allocator by increasing allocation efficiency and consequently reducing the number of cycles packets spend blocked requesting to traverse the router switch by 7.5%-21.5%. Performance gains decrease as packet length increases because incremental allocation creates connections as well. However, packet chaining still provides better or comparable throughput with more expensive and slower allocators for packets of any length. For instance, saturation throughput is comparable (packet chaining gains drop to 2%) for packets of four to sixteen flits. Finally, packet chaining increases IPC up to 46% (16% average) for a CMP executing application benchmarks, which generates traffic with short and long packets. Given the maturity of NoC routers and allocators, these performance gains are significant.

2. DETAILED DESCRIPTION

2.1 Conventional Allocation with Short Packets

With short packets, a conventional single-iteration separable allocator gives poor matching efficiency because it is frequently restarting the allocation process and in a single iteration is not able to compute an efficient matching. While multiple iterations or more complex allocators could improve matching efficiency, they are typically not feasible within a tight timing budget.

The poor allocation efficiency of a conventional iSLIP allocator in the extreme case of single-flit packets is illustrated in Figure 1. The figure shows three cycles of allocations for a 6×6 router. Each input port has four VCs each containing a single one-flit packet. Each packet is labeled with the output port it requires. No additional packets arrive over the three cycles illustrated. Dark squares denote both requests and grants generated by the input and output arbiters. With input-first allocation, an input arbiter first selects one request from each row (input grants) and then an output arbiter selects one surviving request from each column (output grants). After a single iteration the resulting allocation is poor—with just three grants out of a possible five.

The situation is repeated in cycles 1 and 2, but with the iSLIP allocator rotating the input and output arbiter priorities for ports that are granted. Outputs are left idle because many input arbiters picked the same output, which can only serve a single input at a time. If there was time for multiple iterations, these idle outputs would be connected to unmatched inputs.

Figure 3(a) shows the activity on the output channels. For a cycle in which an output is busy, a dark rectangle is labeled with the input and the VC that is using that output. Output 2 is idle for all three cycles because no packet requests it. There are five other idle output cycles. A better allocator could fill most of these cycles resulting in higher throughput.

2.2 Packet Chaining

Packet chaining [19] performs more efficient allocation without increasing cycle time by starting with the existing set of connections and holding any finishing connections that can

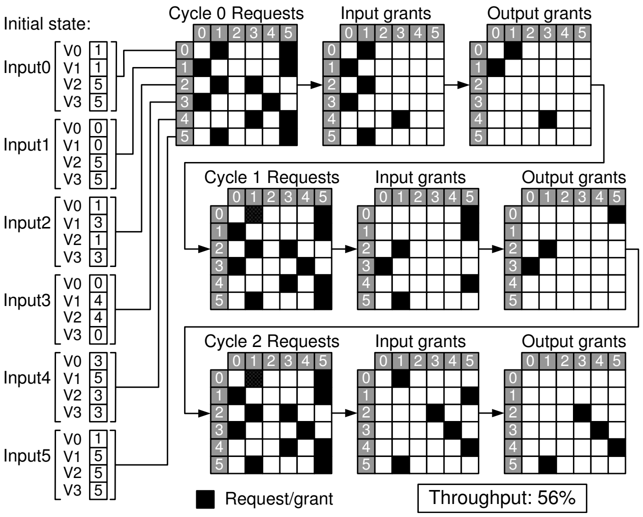


Figure 1: Example allocation of iSLIP without packet chaining.

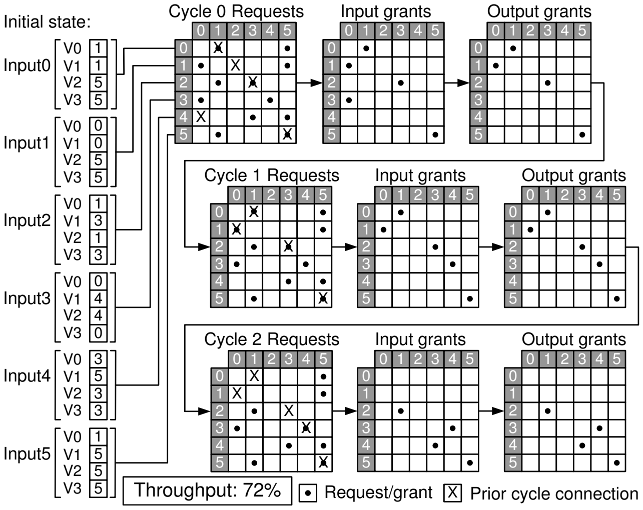
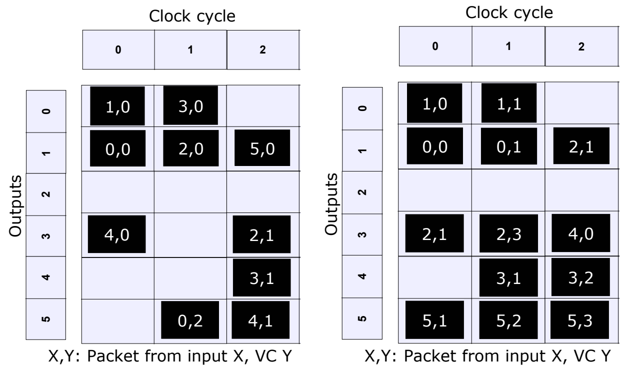


Figure 2: Example allocation of iSLIP with packet chaining.



Outputs	0	1	2
0	1,0	3,0	
1	0,0	2,0	5,0
2			
3	4,0		2,1
4			3,1
5		0,2	4,1

(a) Without packet chaining.

Outputs	0	1	2
0	1,0	1,1	
1	0,0	0,1	2,1
2			
3	2,1	2,3	4,0
4		3,1	3,2
5	5,1	5,2	5,3

(b) With packet chaining.

Figure 3: A timeline of output port usage using the same example.

be used by waiting packets. Packet chaining in effect *chains* packets together, even if they are from different inputs or VCs, so they look like one longer packet to the switch allocator. The switch allocator does not start from scratch, but from this initial state of chained connections. The result is comparable to running multiple iterations of a conventional allocator.

Packet chaining finds a new packet, potentially from any input and VC, destined to the same output to chain onto a departing tail flit. A new waiting packet is suitable if (a) it has been routed to the same output as the tail flit, (b) there is a free output VC it is eligible to use, and (c) there is at least one credit for that output VC. The chained packet need not be at its start; partially transmitted packets can be chained, in which case the only eligible output VC is the one to which the packet is already assigned, as stored in control state logic of input VCs.

Figure 2 illustrates the same example as Figure 1 with packet chaining. In this figure, an X denotes a connection during the previous cycle and a dot (•) denotes requests and grants. In cycle 0, the allocator starts with five connections inherited from cycle -1. Three of these connections are reused by new packets requesting the same output and thus chaining onto the departing packet as denoted by a square in the cycle 0 request matrix with both an X and a •. The other two connections, denoted by an X without a •, are terminated. The three chained packets eliminate competing requests for the same input and output ports before the input arbiters. The result of arbitration at the inputs includes the three chained packets as well as additional requests. Only one of the new requests is granted by the output arbiter, giving four packets transmitted in cycle 0.

As shown in the second row of Figure 2, all four connections from cycle 0 are chained in cycle 1. The allocator makes one additional grant giving a total of five packets transmitted in cycle 1. In cycle 2, only two of the five connections are chained and the allocator makes two additional grants, resulting in four packets being transmitted.

Figure 3(b) shows the activity on the output channels. Other than output 2 being idle because there are no requests for it, the figure shows two other idle output cycles, compared to five for allocation without chaining. Of these two idle cycles, only the first—on output 4 in cycle 0—is avoidable. A better allocator could have assigned (3,1) or (3,2) to this channel. An idle cycle on either channel 0 or channel 4 in cycle 2 is unavoidable since only input 3 has requests for these two outputs. The allocator thus generates maximum matchings in cycles 1 and 2.

Overall, packet chaining increases the quality of allocation, resulting in more packets being sent (13 vs. 10 in this example). This is accomplished by reusing existing connections where possible rather than returning the inputs and output ports to the switch allocation pool where they run the risk of being idled due to inefficient allocation as described in Section 2.1. This is equivalent to performing multiple allocator iterations, one per cycle, while at the same time using the results of each iteration, as in incremental allocation [20] but independent of packet length. Packet chaining does not always result in maximal or maximum matchings. For example, an additional allocator iteration would add a grant from input 3 to output 4 in cycle 0. Packet chaining and incremental allocation provide little benefit to allocators whose matching does not improve with multiple iterations.

Connections are released if they cannot be used productively either because the output VC has no more credits or the input VC becomes empty [13]. To accommodate higher-priority traffic, a connection is released if a higher-priority request exists for the connected output. Chained packets may bypass older packets residing at another VC, but this is also possible without packet chaining if there is more than one VC. Therefore, packet chaining does not cause out-of-order delivery of packets or flits if they would be ordered without packet chaining.

We implement packet chaining on top of a combined switch-VC allocator [13] that reserves output VCs only for packets which win switch allocation. This leaves more output VCs free compared to performing VC allocation in advance, therefore giving more flexibility to packet chaining to find free output VCs. However, packet chaining may also be used in a router with a VC allocator [21].

2.3 Chaining Variations

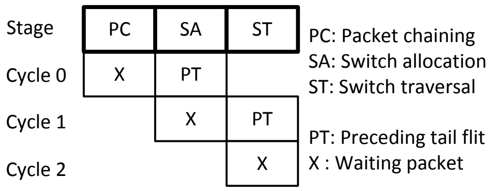
We consider three variations in the set of inputs and VCs that are considered for chaining:

- *Same input VC*: The simplest scheme is to consider only the same input VC as the previous packet that used the connection.
- *Same input, any VC*: This scheme considers all eligible VCs of the same input as the previous packet that used the connection. This multiplies the probability of finding a suitable packet by the number of VCs.
- *Any input, any VC*: This scheme considers eligible packets in any input and any VC. Thus, this scheme increases the probability of finding a packet to chain by the number of other inputs and VCs. Packets are chained in the same way regardless if they are from the same or another input.

The complexity of the chaining logic depends on the packet chaining scheme. Considering only the same input VC requires just a comparator to check if the next packet requests the same output as the departing tail flit. Considering all VCs of the same input requires a similar comparator for each input VC as well as an arbiter to select among eligible input VCs. Finally, the scheme with the most chaining candidates (any input and any VC) requires a complete and separate PC allocator similar to the switch allocator. However, all schemes require the same logic to check for active connections, output VCs and credits. Regardless of complexity and chaining scheme, chaining is performed in the PC stage in the manner described below.

2.4 Packet Chaining Pipeline

Packet chaining adds an extra PC stage to a conventional two-stage VC router with look-ahead routing [8]. Newly arriving packets skip the PC stage and start directly in the switch allocation (SA) stage. Hence, adding the PC stage does not increase router latency. Packets that are not eligible for SA remain in the PC stage and participate in PC allocation. Packets participate in PC allocation if there is an active connection they can use which will be released in the next cycle (the tail flit will be traversing the switch in the next cycle). Packets participate in SA if they are at the head of their input VC and their input and output are not currently connected. As discussed below, packets in the SA stage may participate in both PC and SA. Packets that get chained advance to or remain in the SA stage in the next cycle because they need to remain behind the preceding tail flit, but do not participate in SA. Packets that receive a switch grant advance to the ST (switch traversal) stage.



Stage	PC	SA	ST
Cycle 0	X	PT	
Cycle 1		X	PT
Cycle 2			X

Figure 4: An example of flit X being chained to use PT's connection.

Figure 4 shows a waiting packet X that shares its output with PT (a preceding tail flit). When PT is in the SA stage, requests are submitted to the PC allocator for waiting packets eligible for chaining, to reuse PT's connection. In this example, packet X is granted the connection from the PC allocator during cycle 0. If PT fails SA during cycle 0, PC allocation is cancelled and packets X and PT remain in the same stage—repeating both allocations in the next cycle. In this example, PT receives a switch grant and traverses the switch during cycle 1 while the head flit of X advances to the SA stage. Since X was allocated the connection during the PC stage, it does not participate in SA during cycle 1. The established connection blocks competing packets from X's input and output. Look-ahead routing is performed for packet X during the SA stage. Finally, in cycle 2, the head flit of X traverses the switch. In our latency-optimized two-stage router pipeline, chained flits may not skip the SA stage even if no flit is ahead of them in the ST stage, because that would require a separate VC allocator and would complicate timing with the input channels, buffers and routing logic.

Because packet chaining operates when PT is in the SA stage, it is guaranteed to chain an eligible packet and remove competing packets from consideration by the switch allocator. Biasing the switch allocator to favor maintaining connections across packets does not achieve the same end because competing packets are not removed from consideration, and thus may impact switch allocator decisions negatively.

Because PC allocation considers only outputs currently connected (which will be released in the next cycle) and connected outputs are not eligible for SA, outputs may not participate in both PC and switch allocation. For the same reason, inputs may not participate in both allocations with chaining schemes which consider only the same input, because with those schemes only a single output—that of the previous packet holding the connection—is considered. However, when considering any input for chaining, an input VC may participate in both allocations only if the packet at its head requests two or more outputs, and one is unconnected while the other is connected and available for chaining. Similarly, an input may participate in both allocations as long as it has multiple VCs, because the packets at the heads of that input's VCs may request different outputs. Therefore, conflicts may arise between the two allocators. If the two allocators grant the same input (regardless if they grant the same input VCs), the PC allocator's decision is disregarded and the connection is released enabling packets to bid for that output through the switch allocator.

Eligibility to participate in PC and switch allocation is determined at the beginning of the cycle. However, the eligibility of a packet for chaining may depend on switch allocator decisions in the same cycle as PC allocation. For example, an input VC may contain a packet eligible for chaining except that the input port which contains that VC is part of another connection to another output which has to be released in order for that packet to be chained. Similarly, a tail flit for which there is no connection needs to be granted by the switch allocation in order to form a connection and provide a chaining opportunity for other packets. In each of those two cases, a packet will become eligible for chaining only by a favorable switch allocator decision in the same cycle. Packets in those two cases generate a lower-priority speculative request to the PC allocator. This way, PC allocator requests which may later have to be invalidated do not take resources away from packets which are definitely eligible for chaining. However, sub-optimal chaining decisions are still possible because only some of the lower-priority requests may actually become eligible, but other lower-priority requests may have been granted instead.

For input VCs participating in SA, PC allocator requests are generated based on the flits *behind* the flits at the head of the buffers. Because eligibility for chaining depends on the flits at the head departing, those PC requests are also marked as lower-priority. Enabling this functionality as well as the other lower-priority PC requests described previously is not required for packet chaining and represents a tradeoff between complexity and the number of chaining candidates. Other types of priorities, such as age priorities, are taken into account within each of the two aforementioned priority classes of PC requests.

2.5 Starvation Control

Packet chaining intensifies the fairness and starvation issues of incremental allocation [20, 13] because a connection can remain active indefinitely. To provide limited fairness, we extend packet chaining with starvation control. Starvation control uses age to increase a waiting packet's priority. Since higher-priority requests cause established connections to be released (potentially mid-packet), starvation is prevented. Age should be increased after a predetermined number of cycles that is large enough to preserve the benefits of packet chaining. A simpler alternative is to release a connection and inhibit packet chaining for the affected input and output if a connection has been held for more than a maximum number of cycles. With this mechanism, connections that will reach the starvation threshold at the next cycle are not eligible for chaining. Thus, switch ports held by a long series of packets are returned to the switch allocator pool to be reassigned to waiting packets. The latter mechanism does not require multiple priority levels and thus reduces PC allocator complexity. However, it may not adequately prevent starvation in the rare scenario where a connection is released after reaching the threshold but it keeps being re-established because the other packets that are waiting for the same output and are getting starved cannot request that output because they are in an input which participates in another connection every time the connection causing the starvation is released. In practice this scenario can only occur indefinitely with traffic patterns chosen to exploit this weakness. That is because at low loads input VCs will eventually be left without flits and at high loads output VCs will run out of credits, thus releasing the connection.

3. METHODOLOGY

Evaluation is performed with a modified version of Booksim [6]. The topologies we use are an 8×8 2D mesh and a 4×4 2D FBFly [11]. Routers are connected to one network terminal in the mesh, and four terminals in the FBFly. Therefore, each FBFly router has 10 ports. In the mesh, all channels have one cycle delay. In the FBFly, injection and ejection channels have a delay of one cycle, whereas short, medium and long channels have two, four and six cycles delay, respectively. For the mesh we use deterministic dimension-order routing (DOR) because it is a simple and popular choice. For the FBFly we use universal globally adaptive load-balancing (UGAL) routing [24].

We use VC flow control [4]. Routers use the pipeline described in Section 2.4 and operate at the same clock frequency in all comparisons. Routers require two cycles to generate and transmit credits upstream. In our evaluation, iSLIP [18] and wavefront [25] allocators take into account priorities. The PC allocator uses iSLIP with one iteration (iSLIP-1) because a more complex PC allocator would lengthen the allocation timing path in a router with an iSLIP-1 switch allocator. Unless indicated otherwise, the combined switch/VC allocator also uses iSLIP-1. All separable allocators in our study perform input arbitration before output arbitration. Incremental allocation [20] is used when evaluating networks without packet chaining.

Evaluation is performed using uniform random, random permutation, shuffle, bit complement and tornado traffic patterns [6]. The FBFly also uses transpose and neighbor traffic. They provide little insight for the mesh due to the absence of concentration. Packet length varies from 1 to 16 flits. Injection rates are given in flits. Our evaluation begins with single-flit packets which clearly illustrate the effect of packet chaining, and then proceeds to discuss multi-flit packets and bimodal traffic. Our default configuration has 4 VCs, with 8 buffer slots statically assigned to each. VCs in the FBFly are divided among the two traffic classes required by UGAL. In the default configuration, all packets have equal priority and starvation control is disabled.

We also present execution-driven simulation results for a typical cache-coherent CMP with 64 superscalar, out-of-order RISC CPUs. The CPUs are two-way multithreaded and allow a large number of outstanding memory requests. We use five PARSEC [3] benchmarks and a parallel implementation of FFT. The benchmarks are configured to create two threads per CPU. We use a custom, detailed and timing-accurate CMP simulator which does not simulate the operating system and which interfaces with Booksim. The CMP simulator has an execution-driven front-end and a performance-modeling back-end. The simulator models detailed temporal effects and performance characteristics of the simulated hardware, and influences the front-end in a realistic way. The front-end uses Pin [17] to instrument a native x86 multithreaded binary. The front-end passes RISC-like instructions to the back-end.

For the application simulations, we use the previously described mesh network, packet chaining among all VCs of the same input, and a 64-bit wide datapath. Therefore, short packets are single-flit, while packets carrying our 32-byte cache lines have five flits. For fairness, connections are released if they have been active for eight cycles. L1 caches are 8KB, four-way set-associative, have a single cycle of latency and are private to the cores. L2 caches are shared, non-

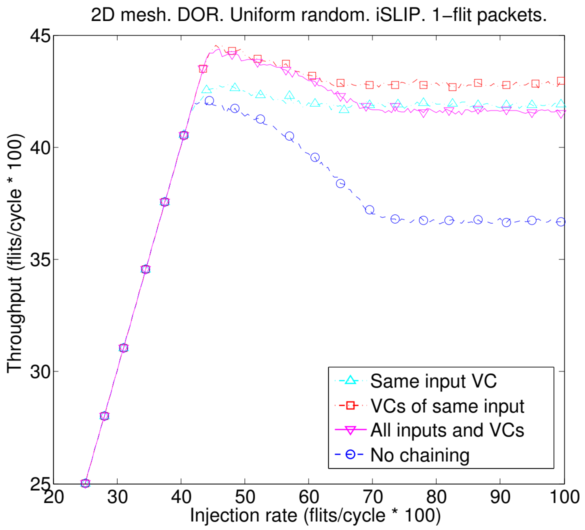


Figure 5: Increasing the injection rate beyond saturation illustrates network instability.

inclusive (they act as victim caches for the L1s), four-way set-associative, have 32KBs per core, and have five cycles of latency. There is one directory and one L2 cache slice located at each core. There is one memory controller in every network quadrant. We assume cores optimized for clock frequency that are clocked at a four times higher clock frequency than the network. Results for IPC correspond to those for execution speedup. On training input datasets IPC results matched measured speedup.

4. EVALUATION

4.1 Throughput Under Heavy Load

Packet chaining stabilizes the network by reducing throughput degradation past saturation. As illustrated in Figure 5, compared to iSLIP-1 (incremental allocation without packet chaining) with single-flit packets, packet chaining increases throughput at maximum injection rate by 15% when considering VCs of the same input. Throughput peaks at saturation injection rate and then decreases because of multi-hop paths of congested packets forming due to a few hot spots, known as tree saturation [12]. Tree saturation still forms with packet chaining because the PC allocator allocates only with local knowledge, but is less pronounced due to the increased allocation efficiency. With packet chaining, throughput drops only marginally (2.5%) past saturation.

Packet chaining does not eliminate instability due to globally unfair allocation. For instance, traffic patterns such as transpose and shuffle that cause instability in mesh networks due to the parking-lot problem [6], where some communicating pairs have to compete for resources more times than others, are not stabilized by packet chaining. In such patterns, stability can be increased by age-based allocation or starvation control. The FBFly is stable both with and without packet chaining.

Performance at maximum injection rate is an important metric because a throttling mechanism might be overly conservative thus reducing available throughput, while the lack of a throttling mechanism may place the network in the instability region. Without elaborate throttling, it is very difficult to consistently operate a network at the point of saturation. Thus, in most systems, network-limited phases of applications operate past saturation where throughput at maximum injection rate dictates performance.

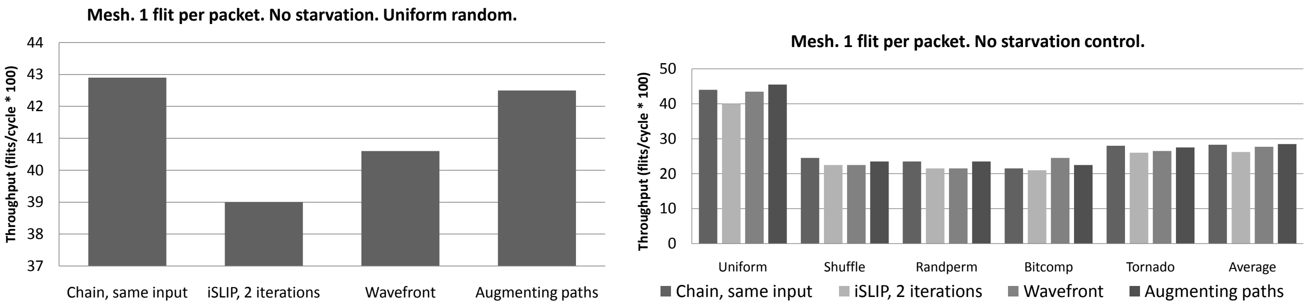
4.2 Comparing with Other Allocators

Figure 6 shows that packet chaining offers comparable or higher throughput than three other popular and more complex allocators. iSLIP-2 refers to iSLIP with 2 iterations. Additional iterations do not considerably improve performance but increase cycle time [18, 6]. Wavefront guarantees maximal matchings [25] at the expense of prolonging the allocation timing path. This is intensified for high-radix routers and makes wavefront allocators reasonably feasible only in small configurations or with full-custom implementations. To exemplify the point, wavefront consumes up to 6× more power and has an increased delay by 36% compared to a separable allocator in a FBFly, and 3× more power and 20% more delay in a mesh [2]. Augmenting paths allocators generate maximum matchings but are too costly for single-cycle implementations [10]. They locate all paths from unmatched inputs to unmatched outputs in the directed bipartite allocation graph [7]. These three allocators, especially augmenting paths, are more costly primarily in cycle time compared to iSLIP-1 with packet chaining. They are used to show that packet chaining improves allocation efficiency without the associated cost of more complex allocators.

Figure 6(a) shows that at maximum injection rate and when considering all VCs of the same input, packet chaining provides a 10% higher throughput compared to iSLIP-2 and 6% compared to wavefront. Furthermore, packet chaining offers comparable throughput (1% more) to an augmenting paths allocator. While an augmenting paths allocator guarantees maximum matchings, it optimizes throughput only locally and does not take into account fairness. Thus, inputs get passed over as long as selecting them prevents a maximum matching. Therefore, packet chaining is able to offer a slight throughput increase at high loads where the fairness issues with augmenting paths lead to increased instability. In addition, packet chaining provides a 22.5% lower average latency than the other allocators—computed as an average from low to saturation injection rates. That percentage becomes 30% if injection rates below 20% are excluded.

Figure 6(b) illustrates the performance of packet chaining in five synthetic traffic patterns. On traffic other than uniform random, packet chaining provides a 4% to 9% higher saturation throughput (5% by average) compared to iSLIP-2 and wavefront, whereas it is comparable to an augmenting paths allocator. These percentage gains increase when evaluating performance at maximum injection rate. The differences between allocators with these traffic patterns are smaller by average compared to uniform random traffic. Uniform random traffic stresses allocators because requests may show up from any input to any output. In contrast, other traffic patterns use only a subset of the inputs and outputs in each router and therefore enable less complex switch allocators to provide efficient matchings.

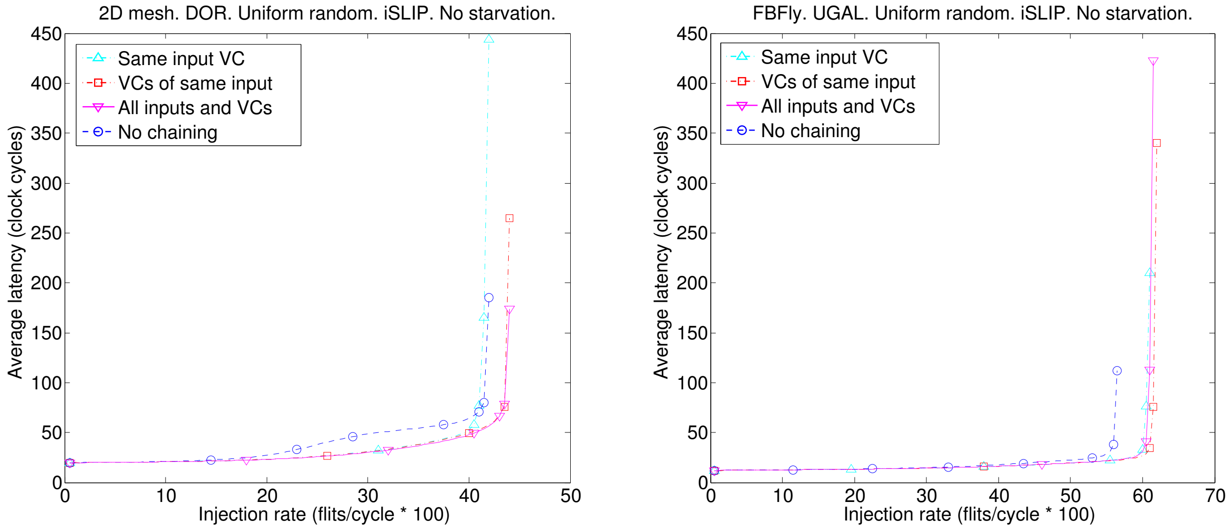
In the FBFly, with single-flit packets and traffic patterns other than uniform random, packet chaining offers a 3% higher saturation throughput than each of the other allocators when selecting among all inputs and VCs. With uniform random traffic, throughput is comparable with an augmenting paths allocator, and 3.5% higher compared to iSLIP-2



(a) Throughput at max. injection rate.

(b) Saturation throughput.

Figure 6: Comparison of packet chaining with other allocators.



(a) 2D mesh.

(b) Flattened butterfly with UGAL.

Figure 7: Injection rate-throughput with single-flit packets.

and wavefront. Finally, average latency is 2%-5% lower with packet chaining. While the relative differences between allocators are the same as the mesh, in the FBFly percentage differences are smaller because packets take fewer hops and therefore bid for the switch less often than in the mesh.

The trends remain the same with longer packets, as shown in Section 4.4.

4.3 Saturation Throughput and Latency

Packet chaining increases saturation throughput and reduces latency compared to iSLIP-1 because packets spend less time blocked at routers. As shown in Figure 7(a), for uniform random traffic, considering all VCs of the same input or all inputs and VCs provides a 5% increase in saturation throughput. By average across traffic patterns, considering all VCs of the same input increases saturation throughput by 6% whereas considering all inputs and VCs by 4%. Packet chaining also provides a 4.5% lower latency by average until saturation. That percentage becomes 16% if statistics for injection rates lower than 20% are excluded. Latency reduction is due to more efficient matchings making flits more likely to advance if their desired output is free. This is similar to the reduction of latency when going from a single to multiple iterations on an iSLIP allocator [6].

To gain further insight, we extract the number of cycles that eligible head flits wait for the connection to their desired output to be released and for a switch allocator grant to be received. This is measured in the mesh at the saturation injection rate for each case; connections are released after eight cycles to prevent starvation and results are compared to iSLIP-1. By average, packet chaining reduces this blocking latency by 13% for single-flit packets, 21.5% for two-flit packets and 7.5% for four- or eight-flit packets. This highlights the increased matching efficiency from packet chaining since packets wait fewer cycles for a switch grant, and therefore more packets traverse the switch in a given time period.

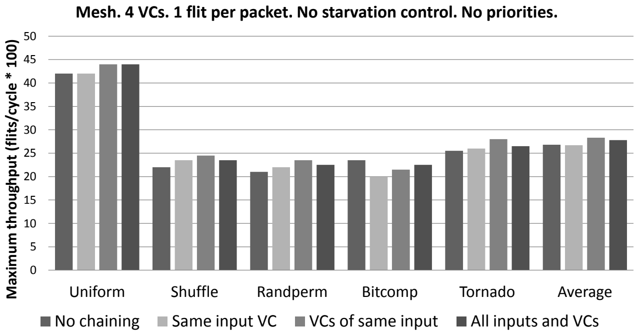
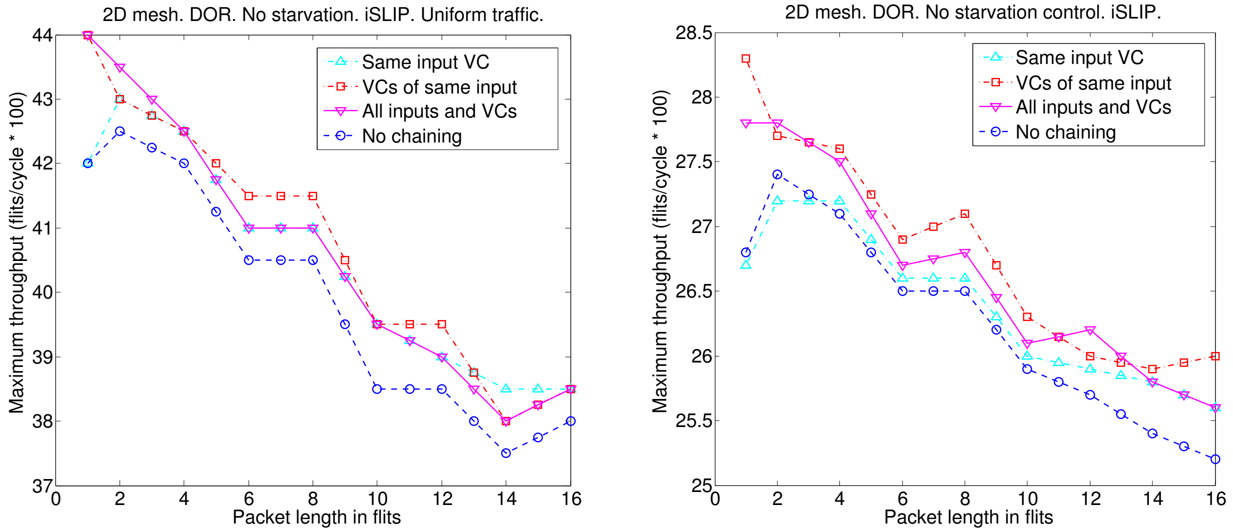


Figure 8: Comparison by traffic pattern.

Results for the FBFly are shown in Figure 7(b). Selecting among all inputs and VCs increases throughput by 9% for uniform random traffic and 4% by average across traffic



(a) Uniform random traffic.

(b) Average across traffic patterns.

Figure 9: Throughput by packet length in flits for the mesh.

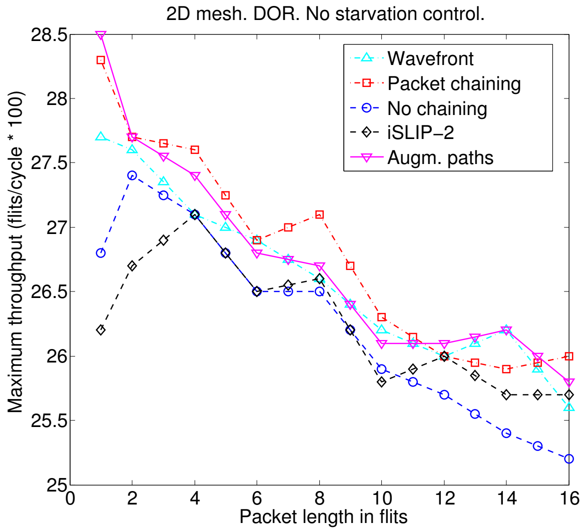


Figure 10: Throughput by packet length for the different allocators averaged across traffic patterns.

patterns, compared to disabling packet chaining. The other selection schemes result in comparable throughput gains.

Figure 8 shows that the advantages of packet chaining remain largely the same across traffic patterns except for *bitcomp* (bit-complement) without starvation control because bitcomp creates continuous flows of traffic which starve other packets. By releasing connections after four cycles with bitcomp, packet chaining is comparable (offers 2% higher throughput) to iSLIP-1 without packet chaining. Results are similar for the FBFly, where packet chaining consistently offers higher throughput, with the exception of *transpose* for the same reason as bitcomp in the mesh. For *shuffle, tornado* and *neighbor*, selecting among VCs of the same input provides a higher throughput than all inputs and VCs.

4.4 Packet Length

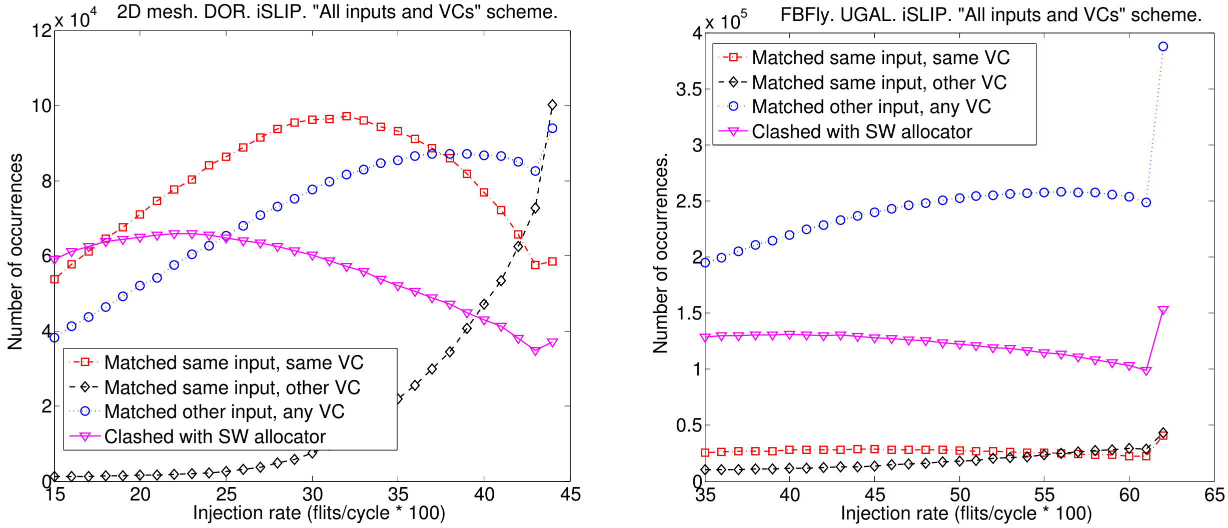
Packet chaining always provides performance benefits, but the benefits decrease when increasing packet length because incremental allocation creates connections and thus improves switch allocation without chaining packets. The effect of packet length is shown in Figure 9. Longer packets translate into fewer packet boundaries and thus fewer packet chaining opportunities. Thus, with long packets the PC allocator has a lower activity factor.

By average across traffic patterns and compared to iSLIP-1 (no chaining), throughput is comparable (2% gain for packet chaining) for eight-flit or longer packets. With starvation control enabled, throughput for uniform random traffic with sixteen-flit packets is no lower than that of iSLIP-1. Disabling starvation control slightly (1.5%) increases throughput for some traffic patterns. The FBFly displays similar behavior as the mesh.

Throughput drops for all test cases with the increase of packet length due to the constant buffer size. Packet chaining enables long packets to be divided into shorter ones to avoid this reduction in performance, without loss of allocation efficiency. The only exception is increasing to two-flit packets with iSLIP-1, which clearly illustrates the gains when incremental allocation is able to form connections.

Figure 10 compares packet chaining to more complex allocators. For eight-flit packets, packet chaining is comparable (outperforms by 2%) to wavefront and iSLIP-2, as well as augmenting paths (outperforms by 1.5%) by average across traffic patterns. For uniform random traffic, packet chaining is comparable to augmenting paths, wavefront (outperforms by 2.5%) and iSLIP-2 (outperforms by 1%). Therefore, packet chaining provides comparable (and slightly increased) throughput to slower and more expensive allocators with long packets. The average throughput of iSLIP-2 with short packets is lower than that of iSLIP-1 because of bitcomp, where locally optimal decisions made possible by the second iteration are not globally optimal.

Traffic patterns which comprise equal amounts of short and long packets (bimodal) still benefit significantly from packet chaining, which increases overall throughput. For instance, when assuming a request-reply protocol with single-flit short and five-flit long packets, packet chaining provides a marginal (1%) throughput increase by average across traf-



(a) 2D mesh.

(b) Flattened butterfly with UGAL.

Figure 11: An illustration of grants by the PC allocator.

fic patterns and a 4% increase for uniform random traffic, when considering all inputs and VCs. These gains are compared to 2.5% for five-flit packets and 5% for single-flit packets under uniform random traffic.

4.5 Optimal Packet Chaining Scheme

The optimal chaining scheme depends on the network configuration which affects what outputs packets are more likely to request. Selecting among all VCs of the same input is optimal for the mesh because it avoids requests from different inputs (outputs) to the same output (input), but still provides the opportunity to chain the majority of departing tail flits because with DOR flits are more likely to remain in the same dimension in each hop. Increasing the density of the request matrix by selecting among all inputs and VCs can decrease matching efficiency of our separable PC allocator, because requests from any input to any output can cause suboptimal decisions at the input and output arbiters.

However, more complex routing algorithms are less predictable, which may necessitate considering all inputs and VCs. In the FBFly with UGAL, flits are less likely to request the same output as the one chained to their input. Considering all inputs and VCs provides gains comparable to considering all VCs of the same input, because the latter still has an adequate number of chaining candidates due to the presence of four VCs per input in our network. With fewer input VCs, considering all inputs and VCs would provide higher performance. The predictability of the outputs requested by packets also depends on the traffic pattern. Furthermore, simpler chaining schemes intensify fairness issues because more input VCs cannot be served before the starvation control mechanism releases the conflicting connection. Selecting among only the same input VC is too restrictive for packet chaining to be effective.

4.6 Packet Chaining Probability

Figure 11 shows the input VCs that are connected with departing tail flits, when considering all inputs and VCs. Connections released because of conflicting switch allocator decisions (grants for the same input or output) are also illustrated. However, failed chaining attempts are not shown.

At low loads, clashes with the switch allocator are more significant because there are only a few chaining candidates and the switch allocator is able to provide efficient matchings. Thus, it is more likely to grant the same inputs and outputs as the PC allocator. The number of clashes initially increases and then decreases when the switch allocator's efficiency decreases. At low loads, a significant percentage of the few packet chaining requests are successful. The number of successful chaining attempts increases with injection rate and remains constant after saturation (0.45 flits/cycle for the mesh and 0.65 flits/cycle for the FBFly).

For the mesh, above 0.32 flits/cycle the number of chains to the same input and VC decreases, and the number of chains to another VC of the same input increases. This is because our network assigns VCs in each traffic class in order starting from the lowest-numbered VC. Thus, at low loads almost all packets are in VC 0. The probability of chaining to another input remains smaller than chaining to the same input because with DOR the input that is already part of the connection is more likely to contain flits routed to the connected output. At saturation, 9% of requests chain to another VC of the same input, 5% chain to the same input and VC, and 8% chain to another input. The low probability of chaining to the same input and VC illustrates the reason for packet chaining's poor performance gains when considering only the same input VC. Results differ for routers at the edges and corners of the mesh because they have fewer inputs and outputs and are also essentially operating at a lighter load due to DOR and the asymmetry of the 2D mesh.

In the FBFly, UGAL routes packets minimally using DOR with one hop per dimension to their intermediate and final destinations [24]. Thus, packets are routed less predictably than in the mesh. However, finding consecutive packets wanting to make the same turn is still likely, as shown by the probability to chain using the same input. Furthermore, due to the two traffic classes required by UGAL, traffic is spread over VCs more than in the mesh, and therefore it is less likely to chain with the same input VC. At saturation, 14.5% of the packets chain with a packet from another input, 2% chain with a packet from the same input and VC, and 2% chain using the same input but another VC. Because the FBFly is symmetric, chaining probabilities are comparable among all FBFly routers.

4.7 Starvation and Priorities

Section 2.5 describes two starvation control mechanisms. In this section, we evaluate the mechanism which releases connections after a predetermined number of cycles. This provides weaker fairness guarantees but also avoids increasing the number of priority classes the PC allocator needs to support. In our simulations, this mechanism was adequate to prevent starvation.

Starvation control has a minimal effect on throughput and latency. For single-flit packets, a starvation threshold of eight cycles provides a marginal (1.5%) throughput increase due to improving fairness, while for eight-flit packets it has no effect. However, setting a starvation threshold smaller than the packet length reduces performance gains because starvation control releases connections before packets can be fully transferred. Therefore, packets wait for a switch allocator grant while having reserved an output VC. For instance, using a starvation threshold of four cycles with eight-flit packets drops maximum throughput by an average of 3%, compared to not using starvation control. This illustrates that starvation control can negate packet chaining gains if it releases connections too early. As discussed in Section 4.3, starvation control increases throughput in certain traffic patterns. In the cases where performance drops with starvation control, packet chaining never performs worse than iSLIP-1 (no packet chaining). Starvation control has a more significant effect with packet chaining schemes with very few chaining candidates, such as considering only the same input and VC. That is because more inputs and VCs risk being starved because they are not considered for chaining. When considering all inputs and VCs, the iSLIP-1 PC allocator already performs round-robin selection of inputs. Selecting a PC allocator with some inherent fairness properties assists with providing adequate fairness and starvation control.

Latency distributions are similar for networks with and without starvation control. Throughput results presented in this paper are the minimum throughput among all sources for each simulation (worst-case throughput). Therefore, worst-case throughput is also similar for networks with and without starvation control. This shows that in all our simulations, connections were released before noticeable starvation or unfairness arose. Under low loads, connections were usually released due to input VCs becoming empty. Under high loads, connections were usually released due to output VCs without credits. Therefore, the starvation mechanisms we proposed should have a threshold which does not degrade performance in the common case, but also adequately prevents fairness issues and starvation under adversarial traffic.

Disabling priority-handling in the PC allocator reduces throughput by 6.5% for uniform random traffic and 4.5% by average across traffic patterns and with single-flit packets. That is because PC allocator requests which are more likely to be cancelled due to unfavorable switch allocator decisions may no longer be placed in the lower priority class, as explained in Section 2.4.

4.8 Application Performance

Table 1: Packet chaining versus iSLIP-1 using benchmarks.

Benchmark	IPC increase	Benchmark	IPC increase
Blackscholes	46%	Canneal	1%
Dedup	6%	FFT	9%
Fluidanimate	3%	Swaptions	29%
Average	16%		

Table 1 presents our application results. Packet chaining increases IPC, but the gains depend on the load and traffic pattern created by each application. Applications with an increased network load, bursty traffic or shorter packets receive higher benefits from packet chaining. Applications with working sets large enough to not fit into L1 caches cause a high load on the network. Under high load, the network may operate past saturation and thus benefit from reduced tree saturation due to packet chaining. For instance, Blackscholes has the largest IPC reduction because it creates more network traffic both in small periods of time (bursty traffic) and by average. Other applications either create less traffic or are more latency-insensitive. Increased load and short packets may be caused by the interaction of the application with the cache system. This is particularly true for systems with small cache lines and more communication-heavy coherence protocols.

Short packets are critical in a typical cache-coherent CMP. They can affect execution time significantly because they transfer time-critical control messages that are often on the application's critical path. Of note, 53% of the packets are single-flit by average across applications in our simulations. In addition to increased throughput, a crucial factor for the IPC increase is reducing packet latency due to packet chaining, especially at times of heavy network load. Maximum packet latency is reduced by an average of 20%. Average latency is only 7% less with packet chaining, because many packets are sent under low network load, and thus have low latency with and without packet chaining. Reducing maximum latency is important under high network load.

Most applications are not affected by the network for most of their execution time, because many parallel algorithms consist of computation phases with no barriers and with working sets that fit into L1 caches. In these cases, gains from network optimizations are limited. Finally, applications may or may not benefit from starvation control, depending on their traffic pattern. For instance, performance for Canneal and FFT degrades without starvation control.

In addition, packet chaining is comparable (provides a 0.5% lower IPC) by average across applications compared to wavefront, which has significant timing and cost overheads as explained in Section 4.9. This clearly illustrates that packet chaining offers performance comparable to more complex allocators without the associated overhead.

Previous work has observed that many applications in certain CMP configurations make light use of the network and thus are not affected by techniques improving throughput, like packet chaining [23]. In these cases, networks can reduce their cost, for example by narrowing their datapath, such that their average load increases and thus they become throughput-limited. Packet chaining makes this option more attractive by increasing maximum throughput and reducing latency past very low injection rates. To illustrate this point, packet chaining increases IPC by an average of 16%

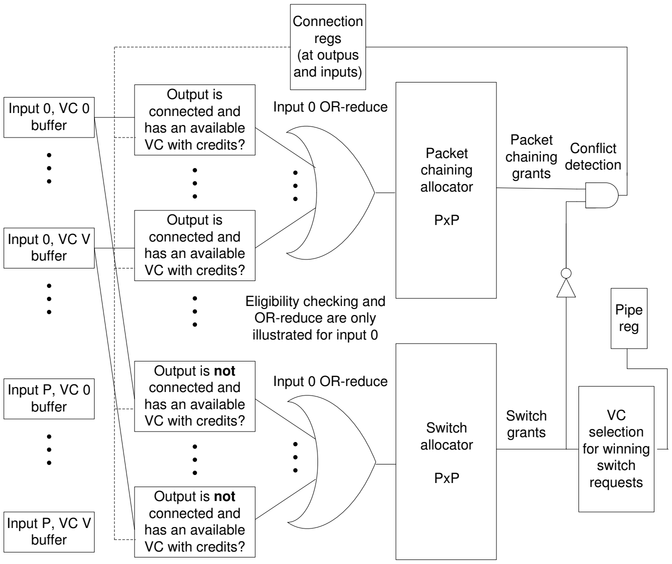

Connection regs (at outpus and inputs)
Input 0, VC 0 buffer
Input 0, VC V buffer
Input P, VC 0 buffer
Input P, VC V buffer
Output is connected and has an available VC with credits?
Output is connected and has an available VC with credits?
Output is **not** connected and has an available VC with credits?
Output is **not** connected and has an available VC with credits?
Input 0 OR-reduce
Input 0 OR-reduce
Eligibility checking and OR-reduce are only illustrated for input 0
Packet chaining allocator
PxP
Switch allocator
PxP
Packet chaining grants
Conflict detection
Switch grants
Pipe reg
VC selection for winning switch requests

Figure 12: Block diagram of the PC and SA stages with the PC and switch allocators in parallel.

compared to iSLIP-1 when both networks have a datapath width of 32 bits. While the average IPC increase across applications remains the same as with a 64-bit datapath, the maximum IPC increase is reduced to 37% for a 32-bit datapath and occurs for Swaptions. These results also show that packet chaining does not increase application performance solely for single-flit packets because with a 32-bit datapath the minimum packet length is two flits.

4.9 Packet Chaining Cost

Packet chaining, when considering all inputs and VCs, requires an extra PC allocator similar to the switch allocator. The PC allocator is placed in parallel to the switch allocator as illustrated in Figure 12. Flits in the PC and SA stages described in Section 2.4 are physically located in the input buffers. Advancing to the SA stage from the PC stage is accomplished by updating the active connections instead of moving flits. Flits depart the buffers when ready to traverse the switch. Similar to the combined switch allocator [13], requests for the PC allocator are OR-reduced to a P×P set of requests. Each input and output port maintains a register to store which other input or output port it is connected to.

Allocators only consider eligible requests as described in Section 2.4. Moreover, the PC allocator must have knowledge of tail flits in the SA stage. That check is part of the eligibility checking—it simply requires an extra input to the AND gate responsible for deasserting ineligible PC allocator requests. All data for eligibility checking resides in the input buffers and state registers, and therefore is available at the beginning of the cycle. This is similar to allocation without packet chaining because that also requires output port information for eligibility checking which resides in the input buffers. At the end of the allocation pipeline stage, a simple logic gate performs conflict detection by deasserting any PC allocator grants that conflict with switch allocator grants. Then, the state registers which keep a record of the active connections are updated, which is part of the allocation timing path with incremental allocation as well. The results of the PC allocator affect switch allocator request eligibility in the next cycle by setting the connection registers.

Note that the logic at the end of the pipeline stage requires a few more logic gates if lower-priority requests to the PC allocator are generated for chaining requests which depend on switch allocator grants, as explained in Section 2.4. However, this check can be performed by a single logic gate at each output which takes as input the switch allocator grant for that output and the desired switch allocator result to make the PC allocator grant for that output valid. This desired result can be computed early in the cycle. However, the conflict detection and lower-priority PC request handling operate in parallel with assigning VCs to winning switch requests which is more complex and also occurs at the end of the pipeline stage for the combined switch allocator [13]. If the switch allocator is not combined but there is a separate VC allocator, the one or two gates per PC allocator output described above prolong the allocation timing path only marginally compared to the rest of the timing path. If speculative VC-switch allocation is used, the logic after the switch allocator to handle speculative requests is similarly complex as, and in parallel with, PC conflict detection.

The cost and timing overhead of packet chaining described above should be compared to wavefront because wavefront provides performance comparable to or lower than packet chaining. In a mesh, wavefront requires up to 3× the power, 2.5× the area and 20% more delay than separable allocators [2]. In high-radix routers such as the FBFly, wavefront occupies 2.7× the area, consumes 6× the power and has an increased delay by 36% [2]. In contrast, adding the PC allocator doubles the area for allocation. Power doubles in the worst case, which we assume for our calculations, but in the average case the switch allocator's activity factor will be reduced, reducing its dynamic power. Furthermore, as explained above, PC allocation does not prolong the allocation timing path with a combined separable switch allocator. Therefore, compared to packet chaining, wavefront requires 1.5× more power, 1.25× more area and 20% more delay in the mesh, as well as 3× more power, 1.35× more area and 36% more delay in the FBFly. Also, compared to packet chaining, a two-iteration separable switch allocator has the same area but twice the delay and worst-case power because it performs two iterations in a single cycle. Finally, augmenting paths allocators are even more complex than wavefront and thus are too costly for single-cycle implementations [10].

Considering only VCs from the same input significantly simplifies packet chaining because an arbiter per input is required instead of a complete allocator. This scheme still offers comparable performance to wavefront in numerous cases with only a small fraction of the cost for the PC allocator and no delay overhead.

5. RELATED WORK

Pseudo-circuits [1] operate on the same principle as packet chaining but only consider consecutive packets in the same input VC. Flits in pseudo-circuits can skip router pipeline stages. Pseudo-circuits are released when another input VC requests the connected output in order to prioritize latency, whereas packet chaining maintains the connection in order to improve allocation efficiency under load. Newly-arriving flits using a connection do not skip the switch allocation stage with packet chaining because in our latency-optimized two-cycle router, doing so would place look-ahead routing [8] in the critical path. It would also require a separate VC allocator which would reduce the number of free VCs available for chaining compared to our combined allocator.

Further research has been performed on allocation. Speculative VC allocation parallelizes VC and switch allocation [21]. Requests can be propagated in advance of flits in frequently-used paths [22] or decisions can be precomputed [21]. Finally, express VCs [15] and token flow control [14] allow flits to bypass the router pipeline based on prior knowledge or established paths. Packet chaining does not rely on pre-established paths and is applicable to such techniques.

6. CONCLUSIONS

Packet chaining is a simple and effective method for increasing allocator matching efficiency without extending allocation time, focusing on short packets. It extends the benefits of incremental allocation to packets of any length. Compared to iSLIP-1 with incremental allocation, which has comparable allocation delay, packet chaining offers a 15% increased throughput at maximum injection rate. Packet chaining increases throughput compared to multi-iteration iSLIP allocators and wavefront allocators by 10% and 6% respectively under maximum injection rate, and gives comparable (1% higher) throughput to an augmenting paths allocator for single-flit packets. For long packets, packet chaining still offers comparable or slightly increased throughput compared to these allocators. Packet chaining achieves this without the delay or cost of these more complex allocators, especially in high-radix routers where the overhead of these allocators increases and can reach up to 6× more power and 37% more delay for a wavefront allocator compared to a separable allocator in a FBFly [2]. Cache-coherent CMPs benefit from packet chaining because short messages are critical and often dominate traffic. In our simulations using application benchmarks, packet chaining increases IPC by up to 46% (16% average). Packet chaining is beneficial to a wide range of systems and provides a simple way to increase allocation efficiency with minimal impact on the allocation timing path and without the area and power overheads of more complex allocators.

Acknowledgments

This work was supported in part by the National Science Foundation under Grant CCF-0702341, in part by the National Security Agency under Contract H98230-08-C-0272-P007 and in part by the Robert Bosch, Prof. Michael Farmwald and Prof. Michael J. Flynn Stanford Graduate Fellowships.

7. REFERENCES

[1] M. Ahn and E. J. Kim. Pseudo-circuit: Accelerating communication for on-chip interconnection networks. In *Proceedings of the 43rd Annual IEEE/ACM International Symposium on Microarchitecture*, 2010.

[2] D. U. Becker and W. J. Dally. Allocator implementations for network-on-chip routers. In *Proceedings of the 2009 ACM/IEEE Conference on Supercomputing*, 2009.

[3] C. Bienia. *Benchmarking Modern Multiprocessors*. PhD thesis, Princeton University, January 2011.

[4] W. J. Dally. Virtual-channel flow control. *IEEE Transactions on Parallel and Distributed Systems*, 3(2), 1992.

[5] W. J. Dally and B. Towles. Route packets, not wires: On-chip interconnection networks. In *Proceedings of the 38th annual Design Automation Conference*, 2001.

[6] W. J. Dally and B. Towles. *Principles and Practices of Interconnection Networks*. Morgan Kaufmann Publishers Inc., 2003.

[7] L. R. Ford and D. R. Fulkerson. Maximal flow through a network. *Canadian Journal of Mathematics*, 8(3), 1956.

[8] M. Galles. Spider: A high-speed network interconnect. *IEEE Micro*, 17(1):34–39, 1997.

[9] P. Gupta and N. McKeown. Designing and implementing a fast crossbar scheduler. *IEEE Micro*, 19:20–28, 1999.

[10] R. R. Hoare, Z. Ding, and A. K. Jones. A near-optimal real-time hardware scheduler for large cardinality crossbar switches. In *Proceedings of the 2006 ACM/IEEE Conference on Supercomputing*, 2006.

[11] J. Kim, W. J. Dally, and D. Abts. Flattened butterfly: a cost-efficient topology for high-radix networks. In *Proceedings of the 34th annual International Symposium on Computer Architecture*, 2007.

[12] C. P. Kruskal and M. Snir. The performance of multistage interconnection networks for multiprocessors. *IEEE Transanctions on Computers*, pages 1091–1098, December 1983.

[13] A. Kumar, P. Kundu, A. Singh, L.-S. Peh, and N. Jhay. A 4.6Tbits/s 3.6GHz single-cycle NoC router with a novel switch allocator in 65nm CMOS. In *Proceedings of the 25th International Conference on Computer Design*, 2007.

[14] A. Kumar, L.-S. Peh, and N. K. Jha. Token flow control. In *Proceedings of the 41st annual IEEE/ACM International Symposium on Microarchitecture*, 2008.

[15] A. Kumar, L.-S. Peh, P. Kundu, and N. K. Jha. Express virtual channels: towards the ideal interconnection fabric. In *Proceedings of the 34th annual international symposium on Computer architecture*, 2007.

[16] R. Kumar, V. Zyuban, and D. Tullsen. Interconnections in multi-core architectures: Understanding mechanisms, overheads and scaling. In *Proceedings of the 32nd annual international symposium on Computer architecture*, 2005.

[17] C.-K. Luk, R. Cohn, R. Muth, H. Patil, A. Klauser, G. Lowney, S. Wallace, V. J. Reddi, and K. Hazelwood. Pin: building customized program analysis tools with dynamic instrumentation. In *Proceedings of the 2005 ACM SIGPLAN conference on Programming language design and implementation*, 2005.

[18] N. McKeown. The iSLIP scheduling algorithm for input-queued switches. *IEEE/ACM Transanctions on Networking*, 7:188–201, 1999.

[19] G. Michelogiannakis, N. Jiang, D. U. Becker, and W. J. Dally. Packet chaining: Efficient single-cycle allocation for on-chip networks. *IEEE Computer Architecture Letters*, 2011.

[20] S. S. Mukherjee, F. Silla, P. Bannon, J. Emer, S. Lang, and D. Webb. A comparative study of arbitration algorithms for the alpha 21364 pipelined router. *SIGARCH Computer Architecture News*, 30:223–234, 2002.

[21] R. Mullins, A. West, and S. Moore. Low-latency virtual-channel routers for on-chip networks. In *Proceedings of the 31st annual International Symposium on Computer Architecture*, 2004.

[22] D. Park, R. Das, C. Nicopoulos, J. Kim, N. Vijaykrishnan, R. K. Iyer, and C. R. Das. Design of a dynamic priority-based fast path architecture for on-chip interconnects. In *Proceedings of the 15th Symposium on High Performance Interconnects*, 2007.

[23] D. Sanchez, G. Michelogiannakis, and C. Kozyrakis. An analysis of interconnection networks for large scale chip-multiprocessors. *ACM Transactions on Architecture and Code Optimization*, 7(1):4:1–4:28, 2010.

[24] A. Singh. *Load-Balanced Routing in Interconnection Networks*. PhD in electrical engineering, Stanford University, 2005.

[25] Y. Tamir and H. C. Chi. Symmetric crossbar arbiters for VLSI communication switches. *IEEE Transactions on Parallel and Distributed Systems*, 4:13–27, 1993.