10-28-2013

apSLIP: A High-performance Adaptive-Effort Pipelined Switch Allocator

Syed Ali Raza Jafri
Purdue University, sjafri@purdue.edu

Hamza Bin Sohail
Purdue University, hsohail@purdue.edu

Mithuna Thottethodi
Purdue University, mithuna@purdue.edu

T.N. Vijaykumar
Purdue University, vijay@purdue.edu

apSLIP: A High-performance Adaptive-Effort Pipelined Switch Allocator

Syed Ali Raza Jafri

Hamza Bin Sohail

Mithuna Thottethodi

T.N. Vijaykumar

TR-ECE-13-13

October 28, 2013

Purdue University

School of Electrical and Computer Engineering

465 Northwestern Avenue

West Lafayette, IN 47907-1285

apSLIP: A High-performance Adaptive-Effort Pipelined Switch Allocator

Syed Ali Raza Jafri, Hamza Bin Sohail, Mithuna Thottethodi, and T.N. Vijaykumar

School of Electrical Engineering

Purdue University

West Lafayette, IN, USA

Email: {sjafri, hsohail, mithuna, vijay}@purdue.edu

Abstract

Switch allocation and queuing discipline has a first-order impact on network performance and hence overall system performance. Unfortunately, there is a fundamental tension between quality of switch allocation and clock-speed. On one hand, sophisticated switch allocators such as iSLIP include dependencies that make pipelining hard. On the other hand, simpler allocators which are pipelineable (and hence amenable to fast clocks) degrade throughput.

This paper proposes apSLIP which uses three novel ideas to adaptively pipeline iSLIP at fast clocks. To address the dependence between the grant and request stages in iSLIP, we allow superfluous requests to occur and leverage the VOQ architecture which naturally enables easy availing of the corresponding grants. To address the dependence between the reading and updating of priority counters in iSLIP, we use stale priority values and solve the resulting double booking by privatizing the priority counters and separating the arbitration into odd and even stream. Further, we observe that while iSLIP can exploit multiple iterations to improve its matching strength, such additional iterations deepen the pipeline and add to the network latency. The improved matching strength helps high-load scenarios whereas the increased latency hurts low-load cases. Therefore, we propose an adaptive-effort pipelined iSLIP – apSLIP – which adapts between one iteration (shallow-pipeline) at low loads and two iterations (deep pipeline) at high loads. Simulations reveal that compared to an aggressive 2-cycle router apSLIP improves, on average, end-to-end packet latency in an 8x8 network by 43% and high-load application performance in a 3x3 network by 19% without affecting the low-load benchmarks.

1 INTRODUCTION

As the microprocessor industry moves towards higher on-chip core counts, the adoption of multi-hop networks as the interconnection fabric is inevitable because neither buses nor crossbars scale adequately. The queuing discipline employed in the on-chip network router has a first order impact on both latency and throughput of the network. Routers can queue flits either at the input ports or the output ports. However, input-queued routers suffer from head-of-line (HOL) blocking which significantly degrades performance [1]. In contrast, output-queued routers are free of HOL blocking but naïve implementations require write bandwidth to the output queues to scale with the number of input ports for the cases where flits from multiple input ports are destined to a single output port. This "speed up" of the output queues is hard even for a few input ports [2]. To address this issue, Karol et al. in [3] propose the virtual output queuing (VOQ) architecture for routers. VOQ creates as many queues at each input port as there are output ports. Because each queue corresponds to a single output port, VOQ completely eliminates head-of-line blocking without the need for speedup of the switching fabric.

To be effective, however, the VOQ scheme requires a sophisticated switch allocation algorithm which can support high network throughput. A low throughput switch would throttle the network and render the VOQ scheme useless. McKeown proposes the iSLIP switch allocation algorithm in [4] which approaches close to a 100% network throughput. VOQ routers along with the iSLIP switch allocation algorithm have been used extensively in Internet routers. Internet routers can exploit VOQ/iSLIP because they do not need flow control and can drop packets upon congestion. In contrast, on-chip network routers cannot gain from the iSLIP algorithm which necessitates a slow clock. Clock speeds are more critical than Internet router clock speeds where router delay is a small fraction of the long end-to-end delay (e.g., 40 ms). Pipelining iSLIP to achieve fast clock is challenging due to dependencies which is the main problem we address in this paper.

An alternative to pipelining is to adapt per-packet switch allocation which reduces the importance of fast allocation by decreasing the frequency of allocation from per-flit to per-packet. In per-packet allocation, a packet holds the

allocated switch port until all the packet's flits are transmitted. Such allocation enables the use of sophisticated (and slow) switch allocators, as employed in Internet routers, where slow clocks are acceptable. However, there are two key disadvantages for on-chip networks. First, per-packet allocation requires either full-packet buffering (which can add significant area/power overheads) or reservation of unused links when packets are spread over multiple routers (which can exacerbate tree-saturation and hence hurt performance). Second, because on-chip networks have a large number of small, single-flit control packets, per-packet switch allocation is no better than per-flit switch allocation. Packet chaining [5] ameliorates this problem by chaining multiple small packets together whenever possible; but at the cost of additional hardware complexity to detect chaining opportunity and duplicate allocators to exploit the opportunity.

Due to the above problems with VOQ and iSLIP, current on-chip network routers employ input queuing implemented via virtual channels (VCs) to alleviate HOL blocking along with simple switch allocation algorithms which are pipelined for throughput. However, the simple algorithms (e.g., SPAA [6]) offer no theoretical guarantees that they can achieve full (100%) network throughput, unlike iSLIP.

We propose *apSLIP* which combines VOQ and *adaptive-effort, pipelined iSLIP* to achieve higher network throughput than the current combination of input queuing and simple switch allocation algorithms. While apSLIP can work with per-flit or per-packet allocation, we focus on per-flit allocation due to its lower hardware overhead.

To provide flow control with VOQ, we observe that in traditional networks, the source router allocates the VC at the destination router and tracks the VC's occupancy for flow control. In VOQ, however**,** the destination virtual output queue is determined at the destination router, unknown to the source router. To address this problem, we utilize look-ahead routing [2] where the destination's output port and therefore the virtual output queue are known at the source router. Alternatives to flow control, such as dropping or deflecting flits, perform worse at high network loads [7, 8]. In addition to flow control, VCs can also provide deadlock freedom for which we use the well-known alternative of dimension-ordered routing (DOR).

To address the main problem of pipelining iSLIP, we propose three novel ideas. Pipelining iSLIP is challenging due to two dependencies amongst its three phases (natural pipeline stages), which cause RAW hazards. The first hazard involves resending requests for flits before the outcome (grant/no-grant) of the previous request for the same flits is known. Such re-sent requests would be superfluous if the earlier request is granted and the corresponding flit dispatched. Such superfluous requests may then receive output grants which constitute lost opportunity for other contending flits. Our first idea is based on the key observation that with VOQ and at high network loads, each virtual output queue will have more than one flit in the common case. Therefore, there will almost always be other flits waiting in the same queue to avail a grant for a superfluous request. We emphasize this VOQ-iSLIP synergy that the grant can be availed easily only in VOQ where all the flits in the queue are destined for the granted output which is not the case in input queuing where finding a flit in an input queue for the granted output is hard. Therefore, combining iSLIP with input queuing instead of VOQ would not achieve the same effect.

The second hazard is a RAW hazard that arises because priority-counters used for round-robin arbitration are written in stage 3 but read in stage 2. Because the priority counters hold metadata and not program data, we ignore the RAW hazard and use stale metadata without violating program dependencies. However, such a strategy does cause performance degradation because of double-booking of resources. We overcome this double booking by separating the arbitrations into odd and even streams which amounts to privatizing the priority counters (a separate set of counters for each stream instead of one-set of counters for all arbitrations).

Pipelining iSLIP fundamentally enables another optimization in the switch allocator by exploiting a key feature of iSLIP. iSLIP is one of the maximal-matching allocators that can achieve higher-quality matching at higher effort via more iterations of the matching algorithms. Unpipelined, multi-iterative iSLIP implementations are worse than single-iteration implementations when it comes to clock speed. However, our pipelining can achieve a 2-iteration, 6-stage pipelined implementation at a fast clock. While the second iSLIP iteration is useful at high network loads (where the increased bandwidth helps reduce queuing latency), the extra latency hurts performance at low loads (where there is no increase in throughput). To address this issue, we propose our third idea of an adaptive-effort allocator that adapts the pipeline depth between one and two iterations depending on the injection rate to achieve low latency at low loads and high bandwidth at high loads.

In summary, the paper's contributions are:

- We pipeline iSLIP by addressing two key hazards:
  - For superfluous requests, we leverage the VOQ architecture which naturally enables easy availing of the corresponding grants
  - For priority-counter hazard, we use stale priority values and avoid the resulting double booking by

privatizing the priority counters and separating the arbitration into odd and even streams.

- We propose apSLIP, an adaptive-effort pipelined iSLIP which adapts between low-effort, low-latency matching at low loads (i.e., one iteration in three stages) and high-effort, high-bandwidth matching at high loads (i.e., two iterations in six stages).

Comparisons with several switch allocators using a trace-driven network simulator and a full-system simulator running commercial and scientific workloads show that apSLIP improves, on average, end-to-end packet latency in an 8x8 network and high-load application performance in a 3x3 network without affecting the low-load benchmarks by 43% and 19%, respectively, over an aggressive 2-cycle router, and 20% and 9%, respectively, over idealized packet-chaining (with per-packet allocation) while using smaller buffers and avoiding duplicate allocators.

The rest of the paper is organized as follows. Section 2 discusses related work. Section 3 provides a brief background on router queuing disciplines and iSLIP. Section 4 describes apSLIP's details. Section 5 describes our experimental methodology and Section 6 presents experimental results. Finally, Section 7 concludes the paper.

2 RELATED WORK

Alternatives to pipelining iSLIP are: (1) bypass the router, (2) reduce router latency to 1 cycle, (3) make switch allocation unimportant, and (4) improve switch allocation algorithm. Proposals for the first option speculatively exploit the lack of resource contention at low and near-zero loads [9] [10] to allow flits to bypass most of the router and incur only wire-delays. The SMART router extends this further to achieve multi-router traversal with only wire-delays [11]. In general, such speculative techniques degenerate to full router latency at modest and high loads. We find that memory-intensive commercial and scientific workloads incur high cache miss rates and thereby high network load so that such speculative techniques do not work well in practice. As such, apSLIP significantly outperforms the techniques (Section 6.1).

The second option includes many shallow-pipelined or even single-cycle router proposals [14] [7]. There are two ways in which the entire router can fit within a single cycle. First, the critical path through the router is truly reduced by eliminating key dependencies and enhancing circuit-level parallelism. In general, modern router designs do not have superfluous dependencies that may be non-speculatively eliminated. Alternately, the second possibility is that even though the critical path is unchanged, the clock happens to be slow enough to accommodate the entire critical path. Such a design offers a marginal latency advantage over a pipelined alternative because of latch over heads in the pipelined design. The latency advantage comes at the cost of reduced bandwidth and is limited only to low loads. At high loads the low bandwidth significantly degrades performance compared to a pipelined alternative with a faster clock. Additionally at low loads, there is not much communication and hence little opportunity to impact overall performance so that the latency advantage does not matter much. At high loads, however, queuing delays dominate router delays, which implies the pipelined design will achieve both better latency and better bandwidth. Not surprisingly, our comparison with an ideal, single-cycle router shows that apSLIP significantly outperforms the router (Section 6.1).

As discussed in Section 1, per-packet switch allocation (e.g., packet chaining [5]) reduces the importance of fast allocation – the third option – but requires full packet buffering to avoid severe performance degradation. This requirement can lead to large buffers and area/power overheads. For example, assuming 7 ports (4 network ports + 3 local ports), a coherence protocol that uses 3-5 virtual networks, 128-bit flits, 5-flit packets (assuming 64-byte cache blocks), and 8 VCs per virtual network, a per-packet design requires between 13-22 KB buffers per router. In contrast, per-flit switch allocation may use fewer flit buffers (say 2-3 flits/queue) thus reducing buffer requirements by 1.67X-2.5X (5.2-13.2 KB per router).

For the fourth option, TS-router [15] proactively avoids scheduling conflicts by using knowledge of future (conflicting) flits. Input ports where flits are expected in the future are prioritized for switch allocation to evacuate older flits before the scheduling conflict occurs (on the arrival of the future flit). This *anticipatory evacuation* policy is effective only at low loads when input queue occupancy is low and thus evacuation is feasible. At medium/high loads, when there are higher numbers of flits, it is impossible to evacuate all flits in time to avoid scheduling conflicts. Consequently, apSLIP significantly outperforms even a 1-cycle TS-router (Section 6.1).

3 BACKGROUND

We discuss queuing discipline in routers, and iSLIP and its variants.

3.1 Input Queuing

Karol et al. [1] showed that the throughput of an NxN port input-queued switch with FIFO queues, under certain conditions, will be limited to just $(2-\sqrt{2}) = 58.6\%$. The underlying cause of this limitation is HOL blocking, where flits are delayed by other flits ahead in line destined for a different output port. The HOL-blocking observed in modern systems is not as bad as suggested by the limit in [1] whose conditions (e.g., all ports equally likely to be taken, single FIFO queuing) are not always true. One of the most

prevalent techniques for reducing HOL blocking is virtual channel flow control proposed by Dally et al. [16]. As shown in Fig 1, a virtual channel (VC) is associated with a buffer which can hold flits of a single packet and other state information. Multiple VCs share the bandwidth of a single physical channel. Hence VCs act like multiple FIFO queues at each input of the router. If flits of one packet (hence one VC) are blocked, the input port can transfer flits from another packet (another VC) hence mitigating HOL blocking. When the packet is fully transferred, the router can allocate the VC to another incoming packet. While VCs can ameliorate HOL blocking (because packets in different VCs do not block one another), they cannot completely eliminate HOL blocking (because packets within VCs cannot bypass blocked packets).

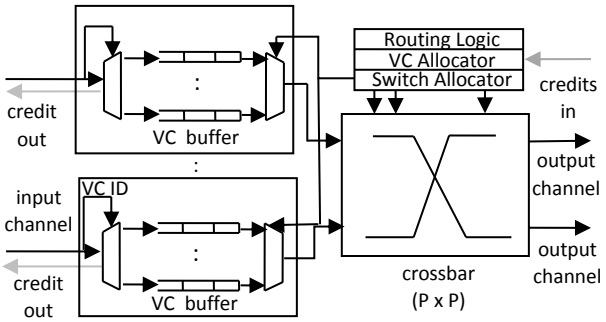

Fig 1: VC Router Architecture

Mukherjee et al. [6] perform a comparison of various switch allocation algorithms for VC based flow control. They propose the Simple Pipelined Arbitration Algorithm (SPAA) and showed its superiority to unpipelined iSLIP and unpipelined Wave Front Algorithm (WFA) [17]. While both iSLIP and WFA can reach higher throughput than SPAA, they are not pipelined and cannot compare in performance with pipelined SPAA at a fast clock. However, SPAA sacrifices powerful matching of input to output ports in favor of pipelineability.

3.2 iSLIP Operation and Pipeline Hazards

Proposed by McKeown in [4], iSLIP is an allocation algorithm that provides lower latency as compared to parallel iterative matching in general and can theoretically reach a 100% network throughput. We enumerate the key steps of iSLIP below:

1. Request (RQ) stage: Each input port sends requests to every output port for which it has a flit.
2. Output Arbitration (OA) stage: Each output port selects on request based on a private counter and informs the corresponding input port. Note the counter is not incremented at this stage.
3. Input Arbitration/Counter Update (IA/CU) stage: In an input port receives grants from multiple output ports, it selects on based on a private round robin counter. The input and output port both increment their counters.

Fig 2 illustrates the unpipelined operation of the iSLIP allocator for two flits. There are two cases of dependencies. First, the RQ stage for subsequent allocation attempts uses information on successful matches from the previous allocation to ensure that successfully matched flits do not continue to assert requests (solid arrow in Fig 2). Second, the priority counters used for round-robin arbitration are written in stage three and read in the OA stage of subsequent allocations (dashed arrow in Fig 2). Pipelining iSLIP reveals that each of these two dependencies translate to RAW (read-after-write) hazards (Fig 3).

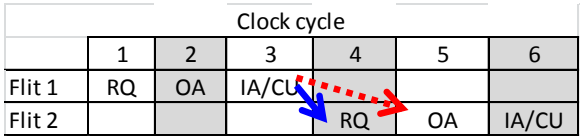

Fig 2: Value communication in unpipelined iSLIP

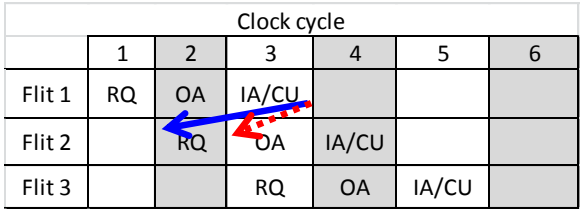

Fig 3: Hazards exposed by pipelining iSLIP

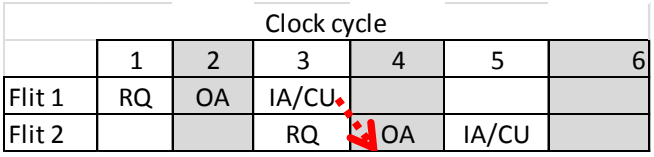

Fig 4: Inter-iteration pipelining in Tiny Tera

3.3 VOQ and variants

In contrast to VCs which map input FIFO queues to packets, VOQs map FIFO queues to the output ports of the router thus completely eliminating HOL blocking (see Fig 5).

As we mention in Section 1, while implementing virtual output queuing is non-trivial in a flow-controlled network, VOQs have been widely adopted in Internet routers where flow control is not required. Researchers have proposed several variants of the powerful multi-iterative iSLIP algorithm to provide high-throughput switch allocation in virtual output queued internet routers. Nick McKeown proposes pipelining across different iterations of iSLIP in the Tiny Tera Internet router to reduce the latency of a single round of multi-iterative iSLIP allocation. The Tiny

Tera switch allocator leverages the fact that an input port which receives at least one output grant in the OA stage is guaranteed to transfer flits and hence should be excluded from resending requests to subsequent allocations to later iterations of iSLIP. Thus, the Tiny Tera switch allocator can start the RQ stage of the next iteration without waiting for the IA/CU stage of the previous iteration to complete. Fig 4 shows the IA/CU-to-RQ hazard being omitted (solid arrow in Fig 3) so that two iterations of one round complete in 5 cycles. In general, Tiny Tera can start a new round of iSLIP arbitration every $2i + 1$ cycles rather than every $3i$ cycles in the unpipelined case where i is the number of iterations per round assuming each stage of iSLIP takes 1 cycle. In contrast, our approach can start a new round every cycle.

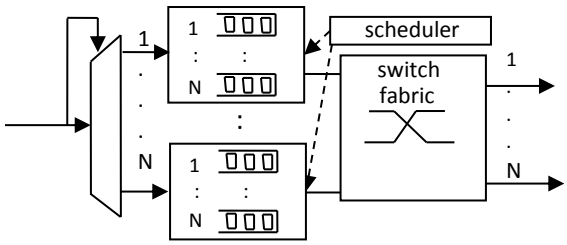

Fig 5: VOQ Router

Kim *et al*. propose using buffered crossbars in high-radix on-chip routers [18]. The buffers in the crossbar act like limited VOQs further reducing HOL blocking. The performance of their switch allocator is bounded by that of the SPAA allocator (with VOQs) because of their use of input arbitration followed by output arbitration.

4 apSLIP

Recall from Section 1 that apSLIP employs VOQ to eliminate HOL blocking combined with our two innovations (1) high-throughput pipelined iSLIP switch allocation, and (2) adaptive-effort switch allocation.

4.1 Virtual Output Queuing in On-chip Networks

As mentioned in Section 1, VOQ has one fundamental operational difference vis-à-vis flow-controlled (i.e., backpressured) networks that use VCs. Essentially, VOQ requires the destination router to determine the home queue of an incoming flit because flits are placed in a virtual queue corresponding to the flit's output port. VC-based flow-controlled networks, on the other hand, require the source router to determine the home queue of the flit on the destination router. The source router allocates a VC on the destination router and tracks the occupancy of this VC when sending a flit to ensure that the destination router does not drop/overwrite any incoming flits.

apSLIP provides VOQ in a flow-controlled network by determining the virtual queue in which the incoming flit will reside at the source router instead of the destination router, using the well-known idea of look-ahead routing. Thus, look-ahead routing enables the use of VOQ in a backpressured network. The apSLIP router provides virtual queues at each input port for each output port of the router. The source router tracks the occupancy of these virtual queues through credits just like in flow-controlled networks with VCs. When sending a flit the source router uses look-ahead routing to determine the output port, and consequently the virtual queue, for which the flit is destined at the destination router. The source router then sends the flit when there is space available in the virtual queue.

The implications of using VOQ as opposed to VCs are many. Aside from eliminating HOL blocking, VOQ also simplifies the apSLIP router by removing VCs and the VCA stage from the pipeline. The primary goal of VCs is to prevent intermingling of flits of different packets. A VC allocated to a packet serves as an input queue which bids for the crossbar in the switch allocation stage. Hence a VC cannot have flits of multiple packets which may be headed in different directions. VOQ, on the other hand, guarantees that all flits in a queue, whether from single or multiple packets,, are headed in the same direction. Therefore there is no need to keep flits of different packets in a virtual queue separate. Note that while flits of different packets could intermingle in a virtual queue, the relative ordering of flits of a single packet is still maintained. Removing VCs lets us shorten the router pipeline by removing the VCA stage. The router determines the VOQ for an outgoing flit in the look-ahead routing stage as a function of the next hop address.

While VOQ improves performance, removing VCs from the network creates challenges which we address next. First, because we allow intermingling of flits from different packets, per-flit switch allocation requires that each flit must now carry address and virtual network information, which results in slightly wider links, router buffers and crossbars (e.g., 8 bits per flit for an 8x8 network with up to four virtual networks). While such per-flit address information is unnecessary for per-packet switch allocation and apSLIP can employ either per-flit or per-packet allocation, we assume the former because the latter imposes high overhead of larger buffers. While VCs with per-packet allocation avoid this overhead, they incur other overheads such as the header flit and per-flit VC number neither of which are needed if each flit carries the address and virtual network. While VOQ's per-flit address incurs area and power overhead (e.g., 8/128 = 6.7% for 128-bit flits in a 8x8 network), VC's header flit incurs power overhead (e.g., 1 header flit per 4 128-bit flits for 64-byte cache block payload = 20%) and the per-flit VC number incurs area overhead (5 bits for 8 VCs x 3 virtual networks for 5/128 =

3.9%). Thus VOQ's overheads are comparable to or better than VC's. Further, while VOQ requires quadratically many buffers compared to VC requiring only linear buffer counts, this difference matters in practice only for high-radix routers which are uncommon in on-chip networks. In fact, our experiments use fewer buffers for VOQ than for VC (Section 5).

Second, because of the absence of VCs, one may think that our design may not use routing algorithms that use VCs for deadlock avoidance. (Note, this concern applies only to network deadlocks. We still use multiple VOQs to avoid coherence deadlocks via virtual networks.) In general, routing algorithms that prevent deadlock by restricting turns can be used without any additional safeguards even in the absence of VCs. Further, using multiple VOQs via virtual networks to break cyclic dependencies is also possible. One possible side-effect of using VOQs with a deadlock prevention strategy may be unbalanced usage of virtual queues. For example, in a 2D mesh using XY dimension-ordered routing (DOR) packets in the Y+/Y- input ports will utilize only Y+/Y- output queues leaving the X+/X- queues unutilized. We counter this imbalance by using non-uniform static queue sizes. One may also use dynamic queue sizes as proposed in [19] [20]. The fact that VCs and VOQs are typically implemented as partitions of a single SRAM array simplifies expanding the highly-utilized queues at the expense of the under-utilized queues.

4.2 VOQ Synergy with Pipelined Switch Allocation

Recall from Section 3.2 that there are two key RAW hazards that prevent naïve pipelining of iSLIP. We start our discussion on pipelining iSLIP with a high-level observation that the RAW hazards are meta-data hazards (hazards that affect request vectors and priority counters; not program data) that affect allocation performance and allocation fairness. As such, ignoring the RAW hazard and using stale information does not violate any program dependencies. However, using stale meta-information (such as priority counters) naively can degrade performance significantly and in the worst case, even cause starvation. We outline the solutions for each of the two hazards. The first hazard is relatively easy to handle and the second hazard is a little more complicated.

Consider the first hazard between IA/CU (stage 3) and RQ (stage 1) in Fig 3, which is common to all pipelined switch allocators. The key challenge is that requests for subsequent allocations must be finalized *before* the outcome (i.e., grants) of prior allocations are determined. Using stale information (i.e., continuing to make switch requests for all outstanding flits), leads to potentially wasted allocations wherein an input port receives redundant grants for flits which have previously been dispatched.

VOQ and iSLIP can synergistically mitigate this hazard's effects. At high loads and consequentially high VOQ occupancy, flits are available in the queue to avail a request grant from an output port. Even though the flit that caused that request is no longer in the queue, the VOQ organization makes it easy to find other flits that are destined for the same output port. At low loads, some switch grants may go unutilized. However, the switch allocator is not a bottleneck at low loads; thus, any wasted grants do not hurt performance as we show in our results.

One may think that the above idea can be applied to VCs as well in one of two ways: (1) The redundant grants could be availed when the input port happens to have another flit in the same VC that can use that switch allocation. However, such availing is more successful in VOQs than VCs because the former holds flits to the same output port in one queue while the latter does not. We include this optimization in our baseline VC implementation and show that apSLIP is significantly better. (2) A more aggressive optimization is to search other VCs of the same input port for flits that could use the redundant grant. Such an optimization is similar to packet-chaining [5], which we show is outperformed by apSLIP.

4.3 Privatization of priority counters

The second hazard occurs between OA stage (stage 2) and the IA/CU stage (stage 3) on the per-port output port counters. Recall from Section 3.2 that an output port, whose grant is accepted by an input port, updates its private counter in IA/CU (stage 3). The output port then reads its private counter to send grants in the following OA (stage 2) (see Fig 3).

	Clock cycle						
	1	2	3	4	5	6	7
Flit 1	RQ	OA	IA/CU				
Flit 2		RQ	Stall	OA	IA/CU		
Flit 3			RQ	Stall	Stall	OA	IA/CU

Fig 6: Stalling to avoid pipeline RAW hazard

A naïve solution would be to stall the pipeline stages to eliminate the hazard as shown in Fig 6. Unfortunately, such a solution would halve the throughput as it achieves matches only every other cycle. Instead of stalling, another naïve solution would be to use stale metadata which results in the output port in stage 2 reading an outdated counter value as shown in Fig 3. Unfortunately, this choice causes serious performance pathology. Specifically, reading the outdated counter value results in the output port nominating the same input port twice (i.e., two reads of the same counter value before an update) In the meantime, the input port accepts grants based on its up-to-date private counter which is read and then updated in stage 3 (Section 3.2). The slow-moving output port counter when coupled with the input port

counter (moving at the regular rate) results in unfairness and significantly degraded performance (8% less saturation throughput than our baseline SPAA router).

The key to iSLIP's successful matching is keeping the private counters of input ports and output ports desynchronized with respect to those of the other input and output ports, respectively. Consider a scenario where two output ports keep sending grants to the same input ports because their private counters keep synchronizing. The input grants can choose only one output port and hence the other would be wasted. Instead, by moving at the correct rate, the counters stay desynchronized.

We need to resolve the hazard between stage 3 (of flit 1) and stage 2 (of flit 2) while ensuring that both input port and output port counters move at the correct rate. Our solution ignores the RAW hazard and uses stale information. To prevent the resulting counter synchronization, we propose duplicating the counters (say counter set 0 and counter set 1), effectively *privatizing* them for odd/even cycles, as shown using subscripts in Fig 7. This counter privatization is similar to compiler variable privatization.

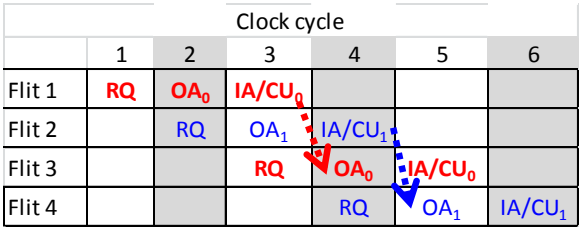

Fig 7: Privatization with duplicate counters

At a high-level, our solution is equivalent to operating two independent, hazard-free allocators, each of which guarantees fairness. At a low-level, we do not actually duplicate the allocators. Rather, we *privatize* the per-port priority counters for odd and even cycles. Because of such privatization, we completely eliminate the hazard between the IA/CU and OA stages of consecutive allocations. Effectively, each allocation uses stale information from two allocations ago. Consecutive writes and reads to the same set of output port counters are now separated by two cycles (see flit 1 and flit 3 in Fig 7) which ensures that the updated counters are available at the end of cycle 3 before they are read in cycle 4. Further, because of the absence of races, the corresponding input and output counters are incremented at the correct rate (i.e., exactly one read per update). We have empirically examined non-uniform arrival patterns other than the example in Fig 7 and confirmed that our privatization is equivalent to unpipelined iSLIP.

Our discussion so far has focused on the apSLIP allocator pipeline. Fig 8 shows the full router pipeline where the look-ahead routing occurs in parallel with apSLIP's RQ.

Router Stage 1	Router Stage 2	Router Stage 3	Router Stage 4	Link
LAR + RQ	OA	IA/CU	Switch Traversal	

Fig 8: apSLIP router pipeline

In [21], the authors target pipelining instruction issue in out-of-order processors which also poses a RAW hazard problem of issuing dependent instructions back-to-back. The authors propose to have grand-parent instructions wakeup grand-child instructions instead of parent instructions waking up child instructions. Our odd-even counters are similar in spirit. However, they do not prevent overbooking whereas our counter-privatization does.

4.4 Multi Iterative and Adaptive pSLIP

We extend the iSLIP pipeline to create a multi-iterative iSLIP switch allocator with increased matching capability. Recall from Section 1 that iSLIP is an iterative, maximal-matching allocator that can achieve better matching by expending additional matching effort in the form of additional iterations. While high-load applications would benefit from the resultant higher throughput, low-load applications would lose performance due to the deeper pipeline's increased latency and higher chances of redundant -grant mis-speculation.

To increase throughput at high loads without hurting latency at low loads, we propose an adaptive-effort iSLIP. Every router employs a six-stage, two-iteration switch allocator pipeline. However, each router independently determines whether to run a single or dual-iterative switch allocator based on network load determined by queue occupancy. At low queue occupancy, the switch allocator provides its matches at the end of the first iteration yielding a three-stage pipeline. When queue occupancy crosses a threshold, the switch allocator runs two iterations corresponding to six stages (we found experimentally that a threshold in the range of 35-60% occupancy works and we use 50% for our results). We revert from two to one iteration only after the queues are empty (and not when the occupancy dips below the threshold), thus providing hysteresis to avoid frequent changes to the pipeline depth. Because we switch from two iterations to one iteration only when the queues (and therefore the request vectors) are empty, we avoid any potential grant-conflicts between an earlier two-iteration allocation and a later one-iteration allocation. While changing the depth of most pipelines at runtime is usually near impossible, the iSLIP pipeline is unique in that the

iterations are identical. Therefore, the pipeline may be terminated at the end of any iteration. We do not examine implementing more than two iterations as we saw diminishing returns from more iterations.

5 Methodology

We use two simulators: a trace-driven network-only simulator using Garnet for an 8x8 network (64 nodes) and the full-system GEMS [22] with Garnet [23] on top of Simics [24] for a 3x3 network (9 nodes) using out-of-order-issue, SMT cores and detailed memory system models (larger systems proved infeasible due to long out-of-order simulation times).While the former can cover larger systems, the latter shows the feedback effect of the network on execution time, not shown by the former, albeit for smaller systems.

Table 1: Workload parameters

Name	Run	Input	Warmup	Inj Rate
PARSEC Benchmarks				
Fluidanimate	200 million cycle traces	Simmedium		0.03
Blackscholes		Simmedium		0.08
Dedup		Simmedium		0.09
Streamcluster		1 mil. 128-D points, 5000 centers		0.10
Canneal		4 native		0.16
Commercial Workloads				
Apache	600 *tx*	20,000 files 45,000 clients 25 ms think time	20,000 *tx cache 2 million system*	0.32
SPECjbb	3000 *tx*	90 warehouses	50,000 *tx cache 1 million system*	0.30
Online Transaction Processing (OLTP)	50 *tx*	25000 warehouses 300 connections	5000 *tx cache 0.1 million system*	0.28
SPLASH-2 Workloads				
Ocean	Full	34x34 grid		0.02
Barnes	Full	512 particles		0.01
Water-nsquared	1 time step	64 molecules		0.02

We compare apSLIP with VoQ with several switch allocators (Section 2): speculative designs [10] [9], SMART [11], TS-router [15], and per-packet with and without packet chaining [5]. To cover the various speculative designs which reduce router latency, we assume an idealized pipelined SPAA-based two-cycle router at all loads which uses virtual channels (our 2-cycle baseline). The first stage overlaps look-ahead routing with ideal single-cycle VC allocation using perfect speculation. The second stage performs both the local and global arbitration phase of the SPAA switch allocator. In this allocator, any redundant grant is availed if the input port has another flit in the same VC (Section 4.2). We also show an idealized, one-cycle SPAA-based baseline router at all loads though the speculative designs achieve low latency only at low loads and incur full latency at high loads (e.g., 4 cycles). The baseline routers use DOR and 8 VCs per input port in each of the three virtual networks and each VC has 5-flit buffers where each flit is 128 bits. Our flit width is in line with recent real products' widths of 16-64 bits [12] [13]. We note that while buses and rings (e.g., Intel Xeon Phi) need to be wider for bandwidth, they can afford to be as they do not have as many links as a typical network. Adding more VCs did not yield any significant improvement. The baselines and all other schemes run at 2.8 GHz (same as core frequency) and have a 1-cycle wire delay in the links beyond the router latency.

We build SMART on the 1-cycle baseline and the other previous schemes, TS-router, per-packet, and packet chaining, on the 2-cycle baseline. We include several idealizations for each of these schemes. For SMART, we assume that the router latency and router set-up are zero cycles for route segments without any contention so that the only latencies are 1-cycle inter-router wire delay and 1-cycle router delay with contention. While the SMART paper assumes 9 routers traversed in one 1-GHz cycle under zero contention, SMART can remove only router delays and not reduce wire delays; the use of asynchronous repeaters in SMART to reduce wire delay is a well-known technique equally applicable to all networks. Therefore we assume 1-cycle/hop wire delay for SMART. For TS-router, per-packet, and packet chaining, we assume an ideal one-cycle switch allocation (2 cycles total router latency). The per-packet switch allocator uses large, packet-sized 18-flit buffers while packet chaining uses the large buffers and an idealized, collision-free, duplicate allocator that finds the best chaining candidate among all the packets in the switch.

Our apSLIP implementation models unevenly partitioned VOQs at each input port, sharing a pool of 64-flit buffers (fewer than the baseline VC's 8x3x5 = 120-flit buffers per input port). Each input port has as many VOQs as there are output ports. apSLIP's adaptive pipeline varies between four and seven stages (total router latency). The first stage includes look-ahead routing (Fig 8) followed by one or two iterations of the iSLIP algorithm at low and high loads, respectively (Section 4.3). apSLIP uses 8 extra bits per flit for address (Section 4.1).

Table 2: System Configuration

System	1 chip 64 cores
Network	8x8 mesh, each tile has a core+private L1, an L2 cache bank and a directory slice connected to a router (three injection/ejection ports); flit width is 128 bit, 2 control virtual networks and 1 data

	network (8 + 8 + 8= 24 VCs) with 8-flit-deep buffers; 1-cycle link latency
Cores	Equivalent to 2-way SMT, 4-issue out-of-order with 40-entry instruction window
Private L1 Caches	Split I & D, each 32KB, 2-way set associative, 64-byte blocks, 2-cycle latency, 16 MSHRs
Shared L2 Cache	Unified 64-MB with 64 banks, 16-way set associative with LRU, 12-cycle bank access latency, 16 MSHRs
Memory	8 GB DRAM, 250-cycle off-chip access time, 16 memory controllers (4 per side of mesh), 2 DIMMs per channel, 2 ranks per DIMM, 8 banks per rank, 32 bank queue entries

Trace-Driven Network-only Simulation: We gather traces from full-system simulations of 64 in-order-issue 2-way SMT cores to run on an 8x8 2-D mesh network simulator with 64 nodes; Table 2 summarizes the system's key parameters. We scale the execution rate of each individual trace to match the per-thread instructions per cycle of an out-of-order-issue 2-way SMT core. We then apply a constant scaling factor to the scale down the execution rate of all traces to compensate for the smaller bisection bandwidth of the 8x8 network to avoid network saturation for our baseline. Note that such downscaling is conservative as it helps the baseline avoid latency explosion associated with network saturation. We measure average end-to-end packet latency (i.e., latency from packet insertion in to the source queue till packet drain at destination) after adequate network warm-up.

Full-System Simulation: The simulated system has 9 out-of-order-issue, 4-way SMT cores. Each tile is similar to that in Table 2 ; but the number of tiles is scaled down to 9. To maintain similar per-node bisection bandwidth for the 3x3 network as the 8x8 network, we scale down the link widths to 32 bits.

Benchmarks: We run three sets of multi-threaded benchmarks: five medium-/low-load scientific applications from the PARSEC suite [25], three high-load commercial applications, and three low-load scientific applications from the SPLASH-II suite [26]. Table 1 column "Inj Rate" gives the load in terms of per-cycle, per-core injection rate for the 8x8 network. We run the commercial benchmarks for a fixed number of transactions after adequate cache warm-up. We scale scientific workloads from SPLASH-II to run to completion. We run the PARSEC and commercial benchmarks for our trace-driven simulations; and commercial and SPLASH-II benchmarks for our full-system simulations (PARSEC takes too long to run to completion due to larger data set sizes). Finally, we also run trace-driven, open-loop simulations using synthetic workloads with data (5 flits) and control (1 flits) packets on the 8x8 2-D mesh network with unbounded source queues.

6 Results

We start with the main comparison of apSLIP+VOQ with several previous switch allocators using commercial and scientific workloads. We then isolate the impact of VOQ and adaptive allocation. Next, we analyze apSLIP's performance using synthetic workloads. Finally, we present circuit-level analysis of the apSLIP switch allocator.

6.1 Performance

Fig 9 plots end-to-end packet latency improvement for the 8x8 network in our trace-driven network simulator. The Y axis shows end-to-end packet latency improvement over our 2-cycle SPAA-based baseline for 1-cycle SPAA-based baseline, idealized SMART on top of the 1-cycle baseline, idealized TS router on top of the 2-cycle baseline, per-packet switch allocation without and with packet chaining on top of the 2-cycle baseline. The X axis shows, in the order of decreasing load (Table 1), our commercial applications with average (Mean Hi), and PARSEC benchmarks with average (Mean Lo), and overall average (Mean).

For the high-load commercial applications, the 1-cycle baseline, SMART, and TS router do not show significant improvement over the 2-cycle baseline. The 1-cycle baseline and SMART are limited by SPAA's poor matching power. The SMART paper shows higher speedups at low loads, while higher loads make uncommon SMART's favorable case of contention free route segments. Recall from Section 2 that TS router's anticipatory evacuation is unable to adequately evacuate the input port queues at high loads and is thus unable to improve scheduling. The TS-router paper shows around 2% improvement over packet chaining which we do not see because the TS-router paper uses buffers that are smaller than a packet which impedes packet chaining (we use packet-sized buffers for packet chaining). Per-packet allocation without and with packet chaining fare better than the previous schemes at high loads; packet chaining achieves significant end-to-end latency improvement (22% Mean), which is in line with the packet chaining paper. Note that our implementation of packet chaining has ideal choice of chaining candidates from all packets in the switch. Still, per-packet allocation is limited by control packets and the HOL blocking in VCs; and packet chaining alleviates but does not eliminate the control packet problem nor the HOL blocking in VCs. Finally, apSLIP improves performance significantly by employing VOQ to remove HOL blocking and iSLIP to achieve high-quality allocation. apSLIP improves the high-load commercial workloads significantly (63% Mean-Hi). Recall that while packet chaining needs large buffers and duplicate allocators and that our implementation uses an idealized, collision-free packet chaining allocator (Section 5), apSLIP performs better without incurring such overheads.

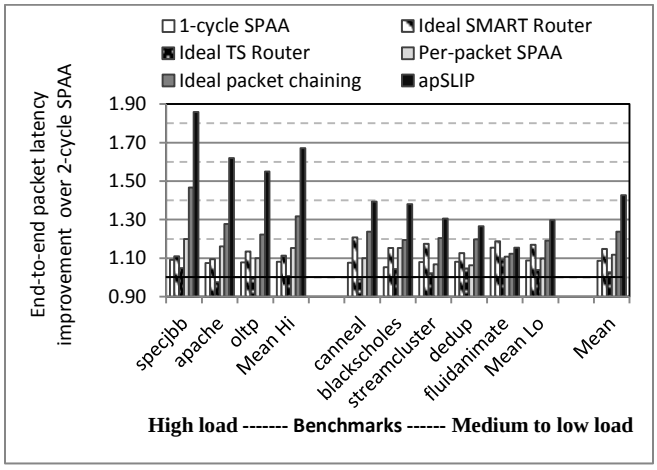

Fig 9: End-to-end packet latency

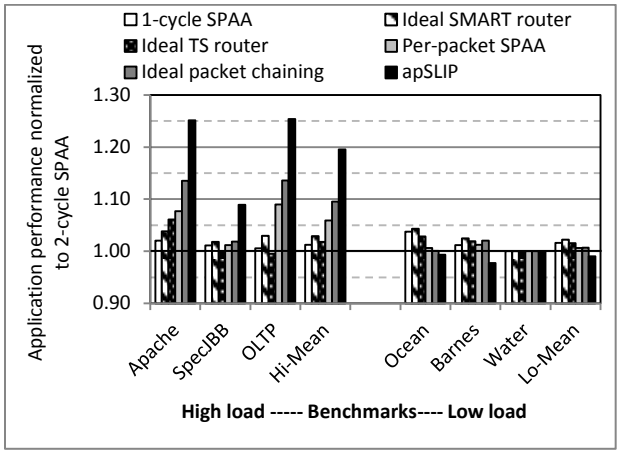

Fig 10: Application performance

For the medium-to-low-load PARSEC applications, the 1-cycle baseline, SMART and TS router provide some end-to-end latency improvements (8-19%). apSLIP with VOQ outperforms all the other schemes for all benchmarks but *fluidanimate* where the ultra-low load gives an edge to the 1-cycle baseline and ideal SMART. apSLIP's longer router latency hurts packet latency while its higher throughput is not needed. Overall, apSLIP improves packet latency by 43% (Mean), performing significantly better than the others.

Fig 10 plots application performance (1/execution time) for a 3x3 network connecting 9 out-of-order-issue 4-way SMT cores in our full-system simulator. The Y axis shows performance for our 1-cycle SPAA-based baseline, idealized SMART, idealized TS router, per-packet switch allocation without and with packet normalized to that of our 2-cycle SPAA-based baseline. The X axis shows our commercial (high load) and SPLASH-II (low load) benchmarks in the order of decreasing load (Table 1).

For the high-load commercial applications, the trends from the end-to-end latency measurements hold, though the two graphs plot different metrics and benchmarks, and should not be compared directly. apSLIP improves performance of the high-load applications by 19%. For the low-load SPLASH-II applications, there is little opportunity to avail of lower latency (1-cycle or ideal SMART) or higher throughput (apSLIP). apSLIP loses a little due to its longer latency but is within 5% of the other idealized schemes.

6.2 Performance breakdown

The apSLIP scheme combines VOQ, pipelined iSLIP, and adaptive pipelining (3- or 6-stage pipeline based on network load). We now isolate the impact of these components. We isolate the impact of VOQ's elimination of HOL blocking from apSLIP in Fig 11 and of apSLIP's adaptive pipelining in Fig 12. We do not isolate the pipelining part of apSLIP because unpipelined iSLIP can generate a match only once every three cycles as opposed to every cycle which would incur severe performance loss. Further, we use full-system simulations in Fig 12 to evaluate impact of adaptivity under real injection rates.

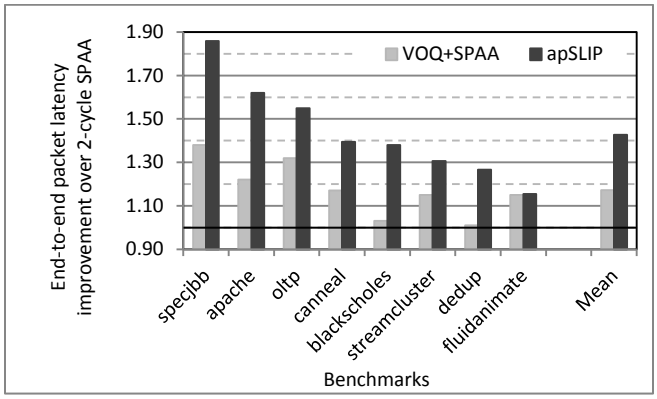

Fig 11: Impact of VOQ on end-to-end packet latency

In Fig 11, we isolate VOQ's impact by comparing SPAA with VOQ and apSLIP with VOQ (i.e., our full scheme) in an 8x8 network running our commercial and PARSEC benchmarks. The Y axis shows end-to-end packet latency improvement over our 2-cycle baseline (SPAA with VC) for SPAA combined with VOQ (2-cycle router latency like the baseline) and apSLIP (4- or 7-cycle router latency). VOQ with SPAA improves over VC with SPAA (our baseline) by 17% due to VOQ's removal of HOL blocking. apSLIP adds another 26% to VOQ's improvements for a total of 43%

(i.e., both components of apSLIP give good benefits). Some benchmarks (*dedup* and *blackscholes*) do not benefit from VOQ as they do not incur HOL blocking. *fluidanimate*'s ultra-low load makes switch allocation unimportant so that VOQ accounts for all of apSLIP's benefits.

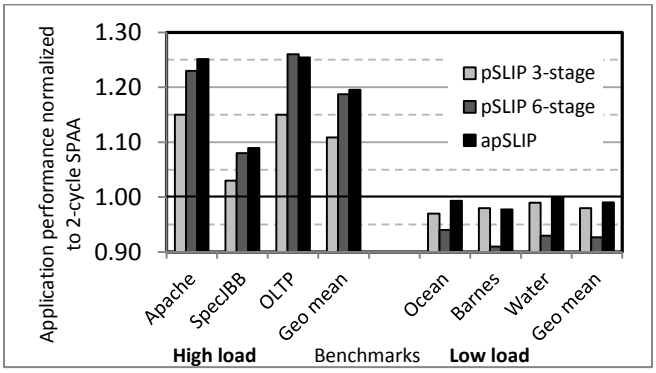

Fig 12: Impact of adaptivity on application performance

In Fig 12, we isolate apSLIP's adaptivity by comparing apSLIP against pipelined iSLIP with static 3 and 6 stages in a 3x3 network running our commercial and SPLASH-II workloads. The Y axis shows application performance for 3-stage, 6-stage, and adaptive pipelined iSLIP normalized to that of our 2-cycle baseline. While the high-load commercial workloads prefer the 6-stage pipeline's higher bandwidth over the 3-stage pipeline's lower latency, the low-load scientific workloads reverse this preference. Being adaptive, apSLIP performs better than or close to the better of the two static pipelines across all loads.

6.3 Synthetic Workloads

To better understand apSLIP's performance, we run synthetic workloads of traffic patterns, namely uniform random, transpose, and bit complement on an 8x8 network (Table 1). In Fig 12(a-c), we plot the end-to-end packet latency (Y axis) versus the injection rate in flits per node per cycle (X axis) for SPAA with per-packet switch allocation, SPAA with packet-chaining, and apSLIP. The uniform random pattern creates contention without hot spots and therefore emphasizes switch allocation. In this pattern (Fig 12(a)), per-packet saturates first followed by packet chaining and apSLIP which performs best. In the bit complement and transpose patterns (Fig 12(b, c)), which stress the network bisection, all three schemes saturate near injection rate of 0.26. These results show that apSLIP is robust across traffic patterns and performs better when switch allocation matters.

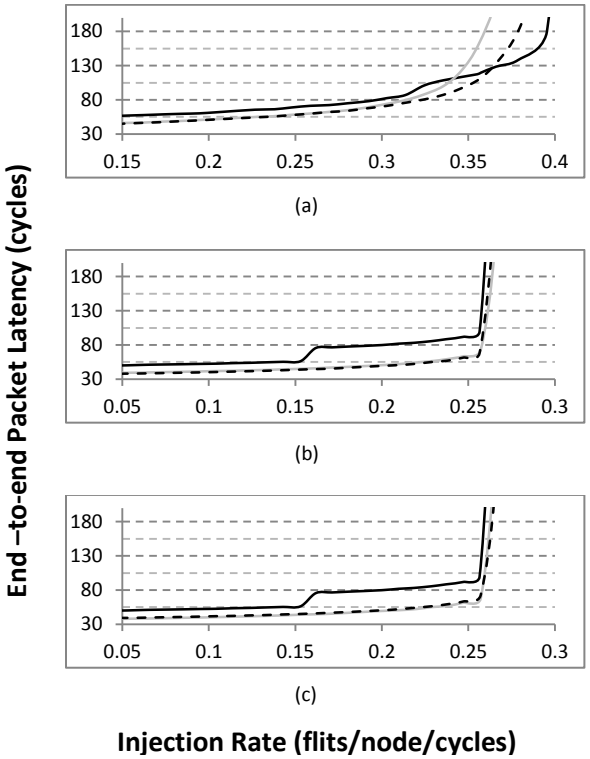

Fig 12: End-to-end packet latency for synthetic workloads: (a) uniform random (b) bit complement (c) transpose

6.4 Circuit Analysis

To analyze apSLIP's circuit delays, we model the two key stages (OA and IA/CU) of apSLIP and SPAA in Verilog, verify the functionality using Mentor Graphics's ModelSim, and synthesize the model in 45 nm technology using Synopsys's Design compiler. We do not model the third lightweight stage which includes only wire delays for forwarding requests. In terms of delays, the input and output arbitrations are qualitatively identical in SPAA and apSLIP. While only the counter updates differ, they are off the critical path. Accordingly, the synthesized models show little difference in clock speeds between SPAA and apSLIP with SPAA allocator at 7.6 FO4 (243 ps) and apSLIP allocator at 7.8 FO4 (249 ps) whereas the input buffer write is the critical path at 8.7 FO4 (278 ps).

For area and power, there are two parts: the switch allocator and the router datapath. Switch allocators account for only 1-2% of router area and 2-4% of router power [10]. Therefore, though apSLIP allocator incurs 72% and 95% overhead in area and power, respectively, over SPAA, apSLIP's actual overheads are small. Recall from Section 4.1 that the datapath area and power overheads of apSLIP using VOQ are comparable to or better than those of SPAA using VC.

7 Conclusion

Switch allocation and queuing discipline has a first-order impact on network performance; and hence on overall system performance. Quality of switch allocation and clock-speed impose opposing constraints: Dependencies in sophisticated switch allocation algorithms such as iSLIP make pipelining at fast clocks hard. On the other hand, simpler, pipelineable algorithms which are amenable to fast clocks degrade throughput.

This paper proposes apSLIP, a high-performance, adaptive-effort, pipelined switch allocator. apSLIP uses three novel ideas to pipeline iSLIP. First, we break the request-grant RAW hazard by leveraging VOQ which easily allows another flit to avail a redundant grant. Second, we untangle double-booking problem arising from priority counter RAW hazard by privatizing priority counters. Finally, we use adaptive effort switch allocation to achieve high-bandwidth at high loads (via a deeper pipeline) and low latency at low loads (via a shallower pipeline). Simulations reveal that compared to an aggressive 2-cycle router apSLIP improves, on average, end-to-end packet latency in an 8x8 network by 43% and high-load application performance in a 3x3 network by 19% without affecting the low-load benchmarks. apSLIP's high bandwidth and low latency are important for on-chip networks to keep up with the ever-growing core counts of multicores.

REFERENCES

[1] M. Karol, M. Hluchyj and S. Morgan, "Input versus output queueing on a space division switch," IEEE Trans. Communications, vol. 35, no. 12, pp. 1347-1356, 1987.

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

[3] M. J. Karol, K. Y. Eng and H. Obara, "Improving the performance of input-queued ATM packet switches," in Proc. of INFOCOM, 1992.

[4] N. McKeown, "The iSLIP scheduling algorithm for input-queued switches," IEEE/ACM Trans. Networks, vol. 7, no. 2, pp. 188-201, 1999.

[5] George Michelogiannakis, Nan Jiang, Daniel Becker, and William Dally, "Packet chaining: efficient single-cycle allocation for on-chip networks," in Proc. of MICRO-44 , 2011.

[6] Shubhendu S. Mukherjee, Federico Silla, Peter Bannon, Joel Emer, Steve Lang, and David Webb, "A comparative study of arbitration algorithms for the Alpha 21364 pipelined router," in Proc. of ASPLOS-X, 2002.

[7] M. Hayenga, N. E. Jerger and M. Lipasti, "SCARAB: a single cycle adaptive routing and bufferless network," in Proc. of MICRO 42, 2009.

[8] T. Moscibroda and O. Mutlu, "A case for bufferless routing in on-chip networks," in Proc. of the 36th ISCA '09, 2009.

[9] Amit Kumar, Li-Shiuan Peh, Partha Kundu, and Niraj K. Jha, "Express virtual channels: towards the ideal interconnection fabric," in Proc. of the 34th ISCA, 2007.

[10] Amit Kumar, Partha Kundu, Arvind P. Singh, Li-Shiuan Peh, and Niraj K. Jha, "A 4.6Tbits/s 3.6GHz single-cycle NoC router with a novel switch allocator in 65nm CMOS," in Proc. of the 25th ICCD 2007..

[11] Tushar Krishna, Chia-Hsin Owen Chen, Woo Cheol Kwon, and Li-Shiuan Peh, "Breaking the On-Chip Latency Barrier Using SMART," in Proc. of 19th HPCA, 2013.

[12] Intel Corporation, "An Introduction to the Intel® QuickPath Interconnect," Intel Corporation, 2009.

[13] Yatin Hoskote, Sriram Vangal, Arvind Singh, Nitin Borkar, and Shekhar Borkar, "A 5-GHz Mesh Interconnect for a Teraflops Processor," IEEE Micro , pp. 51-61, September 2007.

[14] R. Mullins, A. West, and S. Moore, "Low-Latency Virtual-Channel Routers for On-Chip Networks," in Proc. of the 31st ISCA, 2004.

[15] Yuan-Ying Chang, Yoshi Shih-Chieh Huang, Matthew Poremba, Vijaykrishnan Narayanan, Yuan Xie, and Chung-Ta King, "TS-Router: On Maximizing the Quality-of-Allocation in the On-Chip Network," in Proc. of the 19th HPCA, 2013.

[16] W. J. Dally, "Virtual-channel flow control," in Proc. of the 17th ISCA, 1990.

[17] Y. Tamir and H. C. Chi, "Symmetric Crossbar Arbiters for VLSI Communication Switches," IEEE Trans. Parallel. Distrib. Syst., vol. 4, no. 1, pp. 13-27, 1993.

[18] J. Kim, W. J. Dally, B. Towles and A. K. Gupta, "Microarchitecture of a High-Radix Router," in Proc. of the 32nd ISCA, 2005.

[19] Y. Tamir and G. L. Frazier, ""High-performance multiqueue buffers for VLSI communication switches," in Proc. of the 15thISCA, 1988..

[20] Chrysostomos A. Nicopoulos, Dongkook Park, Jongman Kim, N. Vijaykrishnan, Mazin S. Yousif, and Chita R. Das, "ViChaR: A Dynamic Virtual Channel Regulator for Network-on-Chip Routers," in Proc. of the 39th MICRO,, 2006.

[21] J. Stark, M. D. Brown and Y. N. Patt, "On pipelining dynamic instruction scheduling logic," in Proc. of the 33rd MICRO 33, 2000.

[22] Milo M. K. Martin, Daniel J. Sorin, Bradford M. Beckmann, Michael R. Marty, Min Xu, Alaa R. Alameldeen, Kevin E. Moore, Mark D. Hill, and David A. Wood, "Multifacet's general execution-driven multiprocessor simulator (GEMS) toolset.," SIGARCH Comput. Archit. News, vol. 33, no. 4, pp. 92-99, 2005.

[23] Niket Agarwal , Tushar Krishna , Li-shiuan Peh , Niraj K. Jha, "Garnet: A detailed on-chip network model inside a full-system simulator," in Proc. of the ISPASS, 2009.

[24] Peter S. Magnusson, Magnus Christensson, Jesper Eskilson, Daniel Forsgren, Gustav Hållberg, Johan Högberg, Fredrik Larsson, Andreas Moestedt, and Bengt Werner, "Simics: A full system simulation platform," Computer, vol. 35, no. 2, pp. 50-58, 2002.

[25] Steven Cameron Woo, Moriyoshi Ohara, Evan Torrie, Jaswinder Pal Singh, and Anoop Gupta, "The SPLASH-2 programs: characterization and methodological considerations," in Proc. of the 22nd ISCA, 1995.

[26] C. Bienia, S. Kumar, J. P. Singh and K. Li, "The PARSEC benchmark suite: characterization and architectural implications," in Proc. of the 17th PACT, 2008.

[27] Thomas E. Anderson, Susan S. Owicki, James B. Saxe, and Charles P. Thacker, "High-speed switch-scheduling for local area networks," ACM Trans. Comput. Syst., vol. 11, no. 4, pp. 319-352, 1993.

[28] P. Gupta and N. McKeown, "Designing and Implementing a Fast Crossbar Scheduler," IEEE Micro, vol. 19, no. 1, pp. 20-28, 1999.

[29] S. Vangal, J. Howard, G. Ruhl, S. Dighe, H. Wilson, J. Tschanz, D. Finan, P. Iyer, A. Singh, T. Jacob, S. Jain, S. Venkataraman, Y. Hoskote, N. Borkar, "An 80-tile 1.28TFLOPS network-on-chip in 65nm CMOS," in Proc. of the ISSCC, 2007.