On-chip Interconnection Network for Accelerator-Rich Architectures

Jason Cong Michael Gill Yuchen Hao Glenn Reinman Bo Yuan
Center for Domain Specific Computing, UCLA
{cong, mgill, haoyc, reinman, boyuan}@cs.ucla.edu

ABSTRACT

Modern processors have included hardware accelerators to provide high computation capability and low energy consumption. With specific hardware implementation, accelerators can improve performance and reduce energy consumption by orders of magnitude compared to general purpose cores. However, hardware accelerators cannot tolerate memory and communication latency through extensive multithreading; this increases the demand for efficient memory interface and network-on-chip (NoC) designs.

In this paper we explore the global management of NoCs in accelerator-rich architectures to provide predictable performance and energy efficiency. Accelerator memory accesses exhibit predictable patterns, creating highly utilized network paths. Leveraging these observations we propose reserving NoC paths based on the timing information from the global manager. We further maximize the benefit of paths reservation by regularizing the communication traffic through TLB buffering and hybrid-switching. The combined effect of these optimizations reduces the total execution time by 11.3% over a packet-switched mesh NoC and 8.5% over the EVC [18] and a previous hybrid-switched NoC [29].

1. INTRODUCTION

Accelerators offer orders-of-magnitude improvement in performance and energy efficiency as compared to general-purpose cores. Driven by the need for energy efficiency, the industry has proposed incorporating accelerators other than general-purpose cores into a die. These on-chip accelerators are application-specific implementations of a particular functionality, and can range from simple tasks (i.e. a multiply-accumulate operation) to complex tasks (i.e. FFT, encryption/decryption). An example of existing accelerator architecture designs is the IBM wire-speed processor architecture [12], which features special-purpose dedicated accelerators optimized for security, XML parsing, compression, etc. We believe that future computing servers will improve their performance and power efficiency via extensive use of accelerators.

Unlike GPUs, accelerators cannot hide memory latency through extensive multithreading. The throughput of an accelerator is often bounded by the rate at which the accelerator is able to interact with the memory system. For this reason, an open question is: How should the accelerators, cores, buffers, L2 caches and memory controllers should be interconnected for maximum efficiency? Existing interconnect designs for accelerator architectures use partial crossbar switches [12, 10], bus [24] and packet-switched networks [3, 19] to transfer data between memory and accelerators. In Larrabee [24], a wide bi-directional ring bus is used to limit the communication latency. Bakhoda et al. [3] propose using a packet-switched mesh interconnect to connect accelerators and memory interfaces.

Unfortunately, as the number of on-chip accelerators increases, crossbar switches will consume intolerable area overhead due to the inability to scale. The ring bus solution is not sufficient to avoid latency that reaches problematic levels when the number of nodes increases beyond a certain point. Even though the modern packet-switched network enables high bandwidth by sharing channels to multiple packet flows, it comes with long per-hop delays and a high energy overhead.

To combat this problem, circuit-switched fabrics for CMPs are explored [13, 27, 28]. Compared to packet-switched networks, circuit-switched networks can significantly lower the communication latency, but suffer from the long setup and teardown latency. In order to amortize the setup overhead, hybrid-switched networks are proposed, where packet-switched flits and circuit-switched flits are interleaved on the same physical channels [16].

Our investigation shows that accelerator memory accesses exhibit pairwise bulk transfer and streaming, creating highly utilized but frequently changing paths between the accelerator and memory interfaces. Moreover, hardware accelerators feature well-defined I/O patterns following fixed timing, allowing for better resource scheduling. Previous circuit-switched networks do not work well because the setup decisions are made locally and are unaware of the long-term communication patterns or the characteristics of accelerators; this results in unnecessary setups/teardowns and infrequent use of circuit paths. In the meantime, frequent setup and teardown requests hurt the overall performance.

In this paper we propose the *Hybrid network with Predictive Reservation (HPR)* which globally manages NoC resources and reserves paths based on the timing of accelerator memory accesses. The global accelerator manager identifies the location and timing of accelerator memory accesses, reserves paths and resolves conflicts in advance to improve circuit utilization. Meanwhile, the circuit-switched paths are reserved at the granularity of NoC cycles, maximizing the benefit of circuit-switched paths while minimizing the interference caused to packet-switched traffic.

DAC'15, June 07-11, 2015, San Francisco, CA, USA

ACM 978-1-4503-3520-1/15/06$15.00
http://dx.doi.org/10.1145/2744769.2744879.

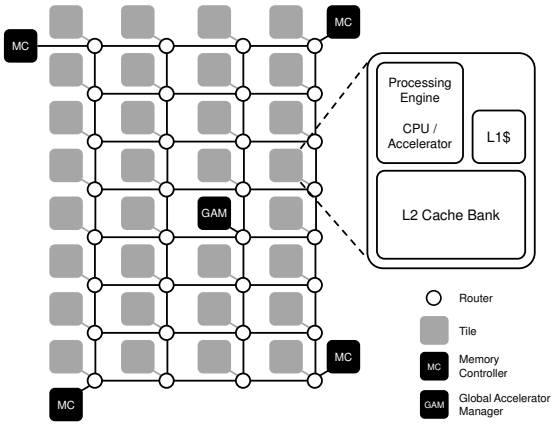


Figure 1: The overview of the accelerator-rich architecture

The key contributions of this work are:

• We analyze the common characteristics of accelerator communication patterns, which suggests opportunities and challenges in shared resource management.

• We extend the global accelerator manager to identify exact locations and precise timing periods of accelerator memory accesses, and predictively reserve a series of circuit-switched paths before the accelerator starts to execute.

• To support the global circuit-switching decisions, we propose a new hybrid network design that provides predictable performance for circuit-switched traffic, while minimizing the interference caused to packet-switched traffic.

2. BASELINE ARCHITECTURE

In this section we describe our baseline accelerator architecture and on-chip interconnect. Fig. 1 illustrates the overall chip layout of the accelerator-rich architecture [7]. This platform consists of cores (with private L1 caches), accelerators, shared L2 banks, memory controllers, the global accelerator manager, and the on-chip interconnect. We assume a 2D mesh topology with memory controllers placed in the corners, similar to the topology used in the Tilera TILE64 [26] and Intel's 80-core design [25], since it provides a simple and scalable network.

Our on-chip accelerator architecture features three key elements: 1) on-chip accelerators that implement complex specialized fixed functionalities, 2) buffer in NUCA [9], a modified cache that enables allocation of buffers in the shared last-level cache, 3) a global accelerator manager that assists with dynamic load balancing and task scheduling.

Loosely Coupled Accelerators: The on-chip accelerators that we consider in this work are specialized hardware computational units that are shared on an as-needed basis among multiple cores. These accelerators can significantly improve performance and save energy. However, this massive increase in performance typically comes with an increase in memory demand that is significantly larger than general-purpose cores. Fig. 2 shows the block diagram of loosely coupled accelerators featuring a dedicated DMA-controller and scratch-pad memory for local storage. The DMA engine is responsible for transferring data between the SPM and shared L2 caches, and between SPMs in the scenario of accelerator chaining.

Buffer in NUCA: Accelerators are designed to work with private buffers. These buffers are introduced to meet two goals: 1) bounding and reducing the fluctuation in latency between memory accesses, and 2) taking advantage of reuse of data within the buffer. As the number of accelerators in a system grows, the amount of dedicated buffer space devoted to these accelerators grows as well. In order to make more efficient use of chip resources, previous work provides a mechanism to allocate these buffers in cache space [11, 9, 19]. When not used as buffers, this memory would instead be used as regular cache. While there are a number of mechanisms for allocating buffers in cache, we chose the Buffer in NUCA [9] scheme due to the consideration of spatial locality and the distributed nature of banked caches found in many-core systems.

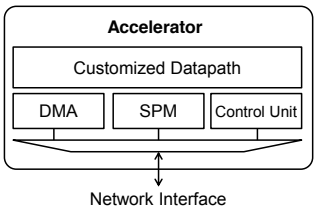


Figure 2: The microarchitecture of the on-chip accelerator

The Global Accelerator Manager (GAM): The GAM fulfills two primary roles: 1) it presents a single interface for software to invoke accelerators, and 2) it offers shared resource management including buffer allocation and dynamic load balancing among multiple accelerators to allow collaboration on a single large task. To invoke an accelerator, a program would first write a description of the work to be performed to a region of shared memory. This description includes location of arguments, data layout, computation to be performed, and the order in which necessary operations to be performed. By evaluating the task description, the GAM splits the requested computation into a number of fixed-size data chunks to enable efficient parallelism, prescreens common TLB misses using a shared TLB, and then dispatches tasks to available accelerators [8].

3. NETWORK CHARACTERIZATION

Due to the pipelined hardware implementation, accelerators often exhibit predictable memory access patterns such as bulk transfers and streaming. In this section we set out to characterize the common accelerator memory access patterns that motivate our proposed NoC design.

We collect the number of streaming flits between node pairs and show the network traffic breakdown in Fig. 3. Streaming flits are defined as consecutive flits traveling between the same source/destination pair. We can observe that the data streams account for a considerable fraction of total on-chip data traffic – more than 70% of total flits transmitted – which suggests that effective optimizations will have a significant impact on performance and efficiency.

We further investigate the memory access patterns of certain benchmarks to explore the potential optimization opportunities. Fig. 4(a) shows the L2 cache bank access traces for `Deblur` from the medical imaging suite [6]. The `Deblur` accelerator takes in two 3D arrays, performs the Rician deconvolution and then outputs the 3D deconvoluted array. The results are collected to show the input stage of one accelerator execution using our simulation platform. Memory data is mapped to cache lines at the memory page granularity to allow for previous optimization techniques, such as hardware prefetching and page coloring, to be easily adapted. As we can see from this figure, the accelerator gen-

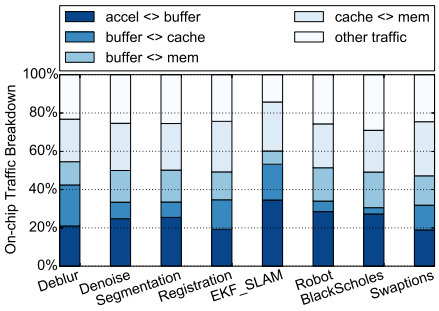


Figure 3: Network Traffic Breakdown. Results are collected on the baseline architecture described in Section 2. Details about the benchmarks and setups of the simulation platform can be found in Section 5.

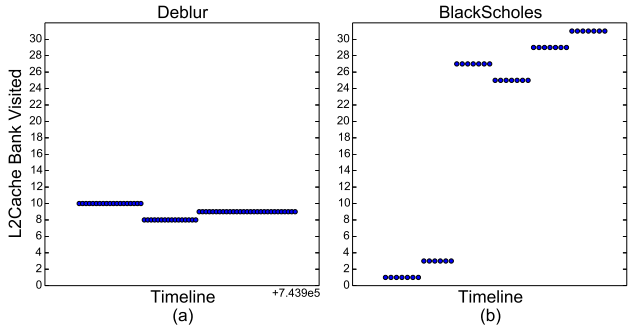


Figure 4: L2 cache bank access trace from the read phase of `Deblur` and `BlackScholes`. A read to a cache bank is depicted as a dot in the figure.

erates consecutive memory requests toward three L2 cache banks. Since these requests have ideal spatial and temporal locality, setting up a circuit-switched path would greatly improve the throughput. Based on our observations, however, this is not always the case for accelerators targeting different domains.

Fig. 4(b) shows the access traces for `BlackScholes` from the financial analytics domain [4]. The figure clearly shows that the accelerator reads from six different banks that correspond to six input arrays defined in the kernel. Unlike `Deblur`, the `BlackScholes` accelerator only requests a small amount of data from multiple cache banks. As a consequence, setting up circuit-switched paths for all destinations will not help to improve the overall performance since each path is not extensively used to amortize the setup overhead.

Although `Deblur` and `BlackScholes` have different memory access patterns, both cases demonstrate that requests are generated following predictable timings. Prior work also proved that accelerator data streams have constant timing intervals between consecutive requests [15].

Based on the above observations, we summarize that data streams introduced by hardware accelerators often 1) account for a considerable portion of the total on-chip traffic, 2) exhibit uninterrupted streams with different lengths towards multiple destinations, and 3) generate memory requests following constant timing. Previous optimizations on packet-switched and hybrid-switched NoCs either add significant per-hop delay to support streaming or fail to identify the pattern to improve circuit utilization. In light of this, we propose global management combined with hybrid-switching to deliver efficient and predictable NoC performance to accelerator-rich architectures.

4. MECHANISM: HPR

We propose the hybrid network with predictive reservation (HPR) to exploit the accelerator memory access characteristics to improve the network performance. The goal is to maximize the benefit of circuit-switching data streams while reducing the setup overhead and the interference with packet-switched traffic. To achieve this goal, we propose the global management to effectively identify the timing of setup and teardown of circuit-switching paths, and the time-division multiplexing (TDM) based hybrid-switching to provide efficient transmission with low overhead.

4.1 Global Management of Circuit-Switching

As described in Section 2, the global accelerator manager (GAM) provides shared resource management and dynamic load balancing. We extend the GAM to perform predictive reservation (PR) of NoC resources for accelerators to improve the throughput of the network.

The GAM is able to obtain information on the accelerator type and the read/write sets from the task description sent by a core before assigning the task to the accelerator. Therefore, the latency of the accelerator can be easily obtained and the possible read/write locations can be extracted by performing address translations. Based on this information, the communication pairs and the timing of communication are known to the GAM without actually executing the task.

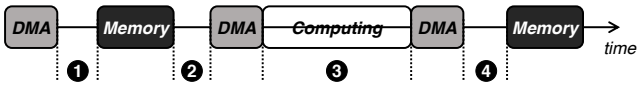


Figure 5: The service timeline of one accelerator execution

Taking advantage of this knowledge, the GAM is capable of scheduling a series of circuit-switched paths at the granularity of time slot for the accelerator. As Fig. 5 shows, at the beginning of processing the task, the DMA engine issues requests to memory interfaces, according to the read set, to fetch the data to the buffer. After these requests arrive at the memory interface, responses will be generated after the access latency, which can be estimated for cache and memory. Once all data requests are fulfilled, the accelerator will start to read from the input buffer and write to the output buffer following the fixed latency. Then results will be written back to the memory. As we can tell from the above process, the latency of each stage is either constant or can be estimated, which suggests that by identifying the timing of the four stages shown in Fig. 5, and in turn providing predictable circuit-switching transmission to traffic in each stage, unpredictable factors in the accelerator execution can be largely reduced.

To prevent conflicts in path reserving, the GAM tracks the current reservation status of each router in the network using a global reservation table. It searches the table for routes and conflicts before sending out the setup message. The XY-routing scheme is adopted for simplicity. If a conflict is found in the table, the GAM will delay the targeted time slots to the earliest available slots. Accordingly, future transactions will be delayed as an aftermath of the conflict. As we try to make reserved circuit-switched paths *as transient as possible*, the probability of conflicts is minimized, and the penalty of delayed reservation is also negligible. With the global management of circuit-switching, the result of setups can be guaranteed beforehand. Thus, no `ACK` message is required, thereby reducing network traffic and setup latency

of circuit-switched paths.

In the event that a TLB miss occurs during the read/write session of an accelerator, the reserved circuit-switched paths must be voided since they can no longer match the communication period, leading to performance degradation. To combat this problem, we propose to translate the address from the read/write set at the GAM, buffer corresponding TLB entries and then send those entries to the accelerator alongside the task description. As a result, the address translation requests at the accelerator side will always hit in the local TLB so that the accelerator is able to execute to completion without TLB misses. In other words, the TLB buffering mechanism eliminates the uncertainty in the accelerator execution, providing the GAM with better estimations of communication timings.

Example 1 shows a case for circuit-switched paths reservation to summarize the proposed global management scheme.

Example 1 Circuit-switched Paths Reservation

```
The GAM receives a task description from a core:
Perform buffer allocation
Extract memory addresses from read/write sets
for all physical address obtained from the read set do
    Locate the L2 $ bank and the Memory controller
    Try to reserve circuit-switched paths using XY-routing between the buffer, L2 cache bank and memory controller
    if conflicts found then
        Delay the time window
    end if
    Update local reserved route record
end for
Reserve circuit-switching paths for both read and write session between the accelerator and the buffer
Do step 4 for the write set
Send the task description to the accelerator
```

4.2 Hybrid Network

We adopt a hybrid-switched router architecture similar to [16, 29], as is shown in Fig. 6 and 7. To support the 2-stage hybrid-switched pipeline, the conventional 4-stage virtual-channeled wormhole router is extended with circuit-switched latches, a reservation table, and demultiplexers.

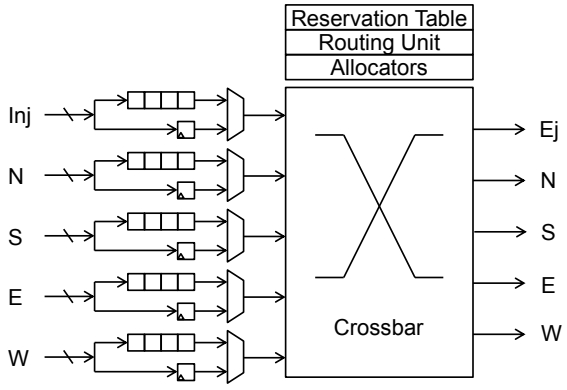


Figure 6: The hybrid-switched router architecture

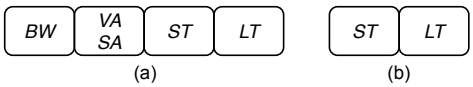


Figure 7: The packet-switched and circuit-switched pipeline

Once the setup decision is made by the GAM, a setup flit carrying the reservation information will traverse the planned path. This setup flit reserves consecutive time slots for multiple future data streams sharing the same path (shown in Fig. 8). To avoid multiple table entries occupied by the same data stream, the reservation table is organized as each entry corresponds to a future data stream – where a start time, an end time, an input port and an output port are recorded. The router uses the reservation table to configure the switch in preparation for circuit-switching. By the completion of one circuit-switched session, the router recovers to packet-switching mode, and the old table entry is freed up to allow for new reservations.

StartTime	EndTime	InPort	OutPort
200	208	1	3

→

StartTime	EndTime	InPort	OutPort
201	209	1	3

Figure 8: An example of the reservation tables from two neighboring routers

By the start of a reserved circuit-switched session, the router needs to check the circuit field of the incoming flit to determine if a data stream is arriving. Once a data stream is confirmed, the router will forward incoming flits directly to the switch that was already configured according to the reservation. The packet-switched flits in the buffer are not allowed to perform virtual channel allocation (VA) and switch allocation (SA) until the end of the session. If the circuit field is zero, this is a packet-switched flit which means no data stream matches the reservation. The reason for a missed reservation can be speculative schedules of circuit paths (i.e., miss-predictions of the data locations). As a result, the current circuit-switched session will be aborted and packet-switched flits are released to proceed to regular pipeline stages.

In order for data streams to *catch* the reserved session, the source and destination network interfaces are notified of the reserved window as well. The source node will not inject streaming flits to the NoC until the reserved session has taken place. This is essential for memory interfaces with variable latency (due to contention, scheduling policies, etc.) because by doing so, a *deadline* is set at the source node to bound the unpredictable memory latency. As a consequence, uninterrupted streams can be generated to fully use the reserved circuit-switched paths.

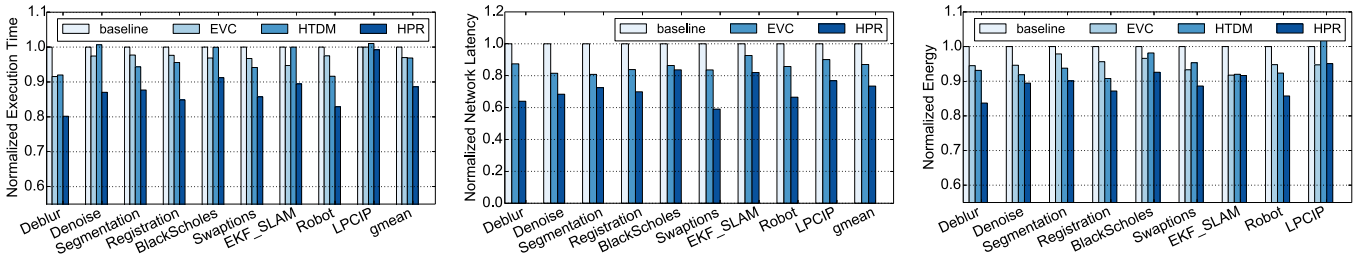
In summary, we propose HPR to effectively identify the timing of data streams and speculatively reserve circuit-switched paths to improve the throughput of the network. Meanwhile, we design a hybrid network that allow the reserved circuit-switched paths to tolerate variable memory latency and provide predictable performance with minimal interference to the packet-switched traffic.

5. SIMULATION RESULTS

5.1 Experimental Methodologies

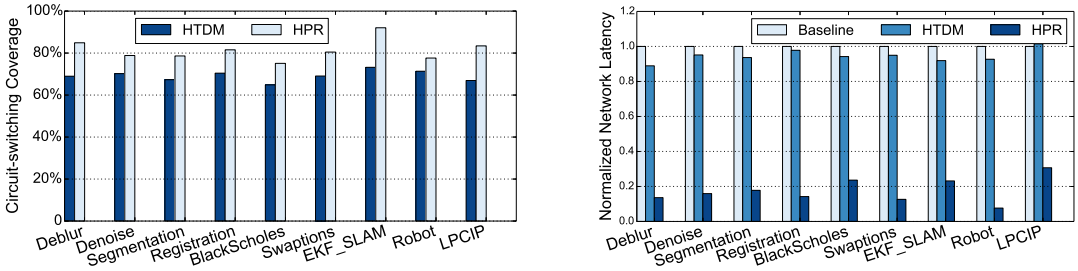
We extended the full-system cycle-accurate Simics [20] and GEMS [21] simulation platform and modified GARNET [2] to model the baseline architecture described in Section 2 and the proposed HPR scheme illustrated in Section 4. We use Orion 2.0 [17] to estimate the power and energy consumption of the NoC. We consider a 32-node mesh topology, with one core, 30 on-chip accelerators and one GAM, with parameters shown in Table 1.

The benchmarks used in our study are four benchmarks from the medical imaging domain [6], two from the financial



(a) Normalized execution time (b) Normalized network latency (c) Normalized energy

Figure 9: Overall simulation results



(a) Fraction of flits covered by circuit-switched paths (b) Network latency of control messages

Figure 10: Network traffic results

analytics domain [4], and three from the robotics domain [1]. We extract the computation-intensive kernel functions from these benchmarks and implement them as on-chip accelerators. We used the high-level synthesis tool Vivado HLS from Xilinx to generate the RTL of these computation-intensive kernel functions, and used the Synopsys design compiler to obtain ASIC characteristics of the accelerators.

Table 1: System parameters for the baseline architecture

NoC	4×8 mesh topology, XY routing, wormhole switching, 4-stage router pipeline
Core	4 Ultra-SPARC III-i cores @ 2GHz
Coherence protocol	MESI directory coherence
L1 data & inst cache	32KB for each core, 4-way set-associative, 64B cache block, 3 cycle access latency, pseudo-LRU
L2 cache (static-NUCA)	2MB, 32 banks, each bank is 64 KB, 8-way set-associative, 64Byte cache block, 6-cycle access latency, pseudo-LRU
Memory	80GB/s bandwidth, 280-cycle access latency

We compare the HPR scheme against a baseline packet-switched NoC with four virtual channels, the express virtual channel (EVC) [18] and a previously proposed TDM-based hybrid-switched NoC for heterogeneous platforms [29] (denoted as HTDM below).

5.2 Experimental Results

Fig. 9(a) shows the comparison of normalized total execution time for each benchmarks against the baseline. HPR reduced the total execution time by 11.3% over the baseline, and 8.5% on average over EVC and HTDM. The largest reduction was seen for `Deblur` – about 20%. This is mainly attributable to high traffic levels between the accelerator and a small number of locations in the benchmark, leading to high circuit utilization. `LPCIP`, on the other hand, represents a relatively small read/write set with dynamic memory accesses defined during the execution. With GAM completely oblivious to them, the dynamic accesses cannot be covered by circuit-switched paths, and reserved paths also experience infrequent use due to unpredictable delay. As can be seen from this figure, none of the evaluated schemes significantly improve the performance.

Fig. 9(b) shows the normalized network latency for EVC and HPR. EVC provides a consistent reduction in network latency of about 15% since the mechanism aims to improve the performance for all on-chip traffic and is oblivious to the heterogeneity in network traffic. Meanwhile, the goal of HPR is to provide efficient and predictable transmission for accelerator data streams. As a consequence, HPR reduces latency for the most critical part of the traffic, resulting in lower overall average network latency.

Fig. 9(c) shows the normalized network energy for the different benchmarks. Again, HPR shows energy reductions over the compared schemes – a reduction of around 11% on average going up to 16% for `Deblur`. In terms of `LPCIP`, whereas HTDM suffers from underutilized circuit-switched paths due to *short* dynamic accesses, HPR saves setup energy on dynamic accesses and still gains from predictively reserved circuit-switched paths.

In trying to study this gain further, we analyze the circuit-switching coverage of streaming flits with respect to the total number of on-chip streaming flits (shown in Fig. 10(a)). As shown in this figure, both HTDM and HPR provide relatively high coverage of streaming flit. The reason that our scheme achieves better performance is essentially twofold. First, in setup of the same circuit-switched path, HTDM requires the confirmation of an `ACK` message before transmission, whereas our scheme eliminates the overhead by introducing global management. Second, our scheme also aims to identify and circuit-switch control messages (e.g., memory requests) which account for a small portion of flits, but are critical to the overall latency of memory access. In contrast, previous schemes only target data movements. Fig. 10(b) illustrates this effect by showing the normalized average network latency for control messages. The largest reductions were seen for `Deblur`, `Swaptions` and `Robot`, which receive the largest reductions in execution time as well. This is because these reductions in network latency will in turn lead

to faster memory responses to the accelerator.

6. RELATED WORK

Accelerator Architectures: We classify on-chip accelerators into two classes: 1) tightly coupled accelerators where the accelerator is a functional unit that is attached to a particular core - Garp [14], UltraSPARC T2 [22], Intel's Larrabee [24] are examples of this, and 2) loosely coupled accelerators where the accelerator is a distinct entity attached to the NoC and can be shared among multiple cores.

Our paper focuses on loosely coupled accelerators in a way where accelerators can be shared between multiple cores. VEAL [5] uses an architecture template for a loop accelerator and proposes a hybrid static-dynamic approach to map a given loop on that architecture. Polymorphic pipeline array [23] uses an array of PEs which can be reconfigured and programmed. Hou et al. [15] study the common characteristics of the data streams introduced by on-chip accelerators. By evaluating two extensions to the existing chip architecture, the authors attempt to identify opportunities and challenges to initiate future research in this area.

Interconnection Networks: There is a large amount of work focusing on improving the overall throughput. In conventional packet-switched networks, deeper router pipelines and buffers account for a significant portion of per-hop delay and energy consumption. Kumar et al. [18] proposed express virtual channels (EVC) in which intermediate nodes are bypassed in order to remove the delay in the buffer write, virtual channel arbitration and switch arbitration. However, EVC is limited to small-hop data communication and incurs significant overhead due to credit management.

Recent research proposes several interconnect designs using circuit-switched networks for on-chip data transfer. In [27], circuit-switching paths are set up based on network transaction handling. All data packets will not be transferred until the circuit-switched path is set up. Wolkotte et al. [28] propose a reconfigurable circuit-switched network for heterogeneous SoCs. Jerger et al. [16] explore hybrid-switched networks where circuit-switched and packet-switched flits share the same fabric. However, the reconfiguration of circuit-switched paths is determined a setup network which adds an additional plane to the network. [29] propose a time-division multiplexing hybrid-switching NoC for CPU-GPU heterogeneous platforms. Fine-grained time slot reservation is implemented and paths sharing is introduced to improve energy efficiency. However, due to the lack of global decision, the performance gains very little and even degrades in some cases.

7. CONCLUSION

In this work, we propose the hybrid network with predictive reservation for accelerator-rich architectures to take advantage of common characteristics of accelerator communication patterns. The global management of NoC resources is proposed to predictively reserve circuit-switched paths for accelerators based on the knowledge of location and precise time period. The hybrid network is designed to provide efficient and predictable transmission during the reserved time period while reducing the interference caused to the packet-switched traffic. The evaluations show that HPR can achieve an average of 11.3% reduction in execution time compared to the baseline packet-switched network and 8.5% compared to EVC and a previous TDM-based hybrid-switched network.

Acknowledgments

We thank the anonymous reviewers for their feedback. This work was supported in part by C-FAR, one of the six SRC STARnet Centers, sponsored by MARCO and DARPA, and by NSF/Intel Innovation Transition (InTrans) Grant awarded to the Center for Domain-Specific Computing (CDSC).

8. REFERENCES

[1] "The mobile robot programming toolkit." [Online]. Available: http://www.mrpt.org/

[2] N. Agarwal *et al.*, "Garnet: A detailed on-chip network model inside a full-system simulator," in *ISPASS*, April, pp. 33–42.

[3] A. Bakhoda *et al.*, "Throughput-effective on-chip networks for manycore accelerators," in *MICRO*, 2010.

[4] C. Bienia, "Benchmarking modern multiprocessors," Ph.D. dissertation, Princeton University, January 2011.

[5] N. Clark *et al.*, "Veal: Virtualized execution accelerator for loops," in *ISCA*, 2008.

[6] J. Cong *et al.*, "Customizable domain-specific computing," *Design Test of Computers, IEEE*, pp. 6–15, March 2011.

[7] J. Cong *et al.*, "Architecture support for accelerator-rich cmps," in *DAC*, 2012.

[8] J. Cong *et al.*, "Architecture support for domain-specific accelerator-rich cmps," *ACM TECS*, vol. 13, no. 4s, pp. 131:1–131:26, 2014.

[9] J. Cong *et al.*, "Bin: a buffer-in-nuca scheme for accelerator-rich cmps," in *ISLPED*, 2012.

[10] J. Cong *et al.*, "Optimization of interconnects between accelerators and shared memories in dark silicon," in *ICCAD*, 2013.

[11] C. F. Fajardo *et al.*, "Buffer-integrated-cache: a cost-effective sram architecture for handheld and embedded platforms," in *DAC*, 2011.

[12] H. Franke *et al.*, "Introduction to the wire-speed processor and architecture," *IBM Journal of Research and Development*, vol. 54, no. 1, pp. 3–1, 2010.

[13] K. Goossens *et al.*, "Æthereal network on chip: concepts, architectures, and implementations," *Design Test of Computers, IEEE*, vol. 22, no. 5, pp. 414–421, 2005.

[14] J. R. Hauser *et al.*, "Garp: A mips processor with a reconfigurable coprocessor," in *FPT*, 1997.

[15] R. Hou *et al.*, "Efficient data streaming with on-chip accelerators: Opportunities and challenges," in *HPCA*, 2011.

[16] N. D. E. Jerger *et al.*, "Circuit-switched coherence," in *NOCS*, 2008.

[17] A. B. Kahng *et al.*, "Orion 2.0: A fast and accurate noc power and area model for early-stage design space exploration," in *DATE*, 2009.

[18] A. Kumar *et al.*, "Express virtual channels: Towards the ideal interconnection fabric," in *ISCA*, 2007.

[19] M. J. Lyons *et al.*, "The accelerator store: a shared memory framework for accelerator-based systems," *TACO*, vol. 8, no. 4, p. 48, 2012.

[20] P. Magnusson *et al.*, "Simics: A full system simulation platform," *Computer*, vol. 35, no. 2, pp. 50–58, Feb.

[21] M. M. K. Martin *et al.*, "Multifacet's general execution-driven multiprocessor simulator (gems) toolset," *SIGARCH Computer Architecture News*, 2005.

[22] U. Nawathe *et al.*, "An 8-core, 64-thread, 64-bit, power efficient sparc soc (niagara 2)," *ISSCC*, 2007.

[23] H. Park *et al.*, "Polymorphic pipeline array: a flexible multicore accelerator with virtualized execution for mobile multimedia applications," in *MICRO*, 2009.

[24] L. Seiler *et al.*, "Larrabee: a many-core x86 architecture for visual computing," *ACM Transactions on Graphics (TOG)*, vol. 27, no. 3, p. 18, 2008.

[25] S. R. Vangal *et al.*, "An 80-tile sub-100-w teraflops processor in 65-nm cmos," *Solid-State Circuits*, vol. 43, no. 1, pp. 29–41, 2008.

[26] D. Wentzlaff *et al.*, "On-chip interconnection architecture of the tile processor," *Micro, IEEE*, pp. 15–31, 2007.

[27] D. Wiklund *et al.*, "Socbus: Switched network on chip for hard real time embedded systems," in *IPDPS*, 2003.

[28] P. T. Wolkotte *et al.*, "An energy-efficient reconfigurable circuit-switched network-on-chip," in *IPDPS*, 2005.

[29] J. Yin *et al.*, "Energy-efficient time-division multiplexed hybrid-switched noc for heterogeneous multicore systems," in *IPDPS*, 2014.