qSwitch: Dynamical Off-Chip Bandwidth Allocation Between Local and Remote Accesses

Shaoming Chen, Lu Peng, Samuel Irving, Zhou Zhao, Weihua Zhang, and Ashok Srivastava, *Life Senior Member, IEEE*

***Abstract*—Multisocket computer systems are popular in workstations and servers. However, they suffer from the relatively low bandwidth of intersocket communication especially for massive parallel workloads that generate many intersocket requests for synchronizations and remote memory accesses. Intersocket traffic puts pressure on the underlying network connecting all processors with a limited bandwidth confined by pin resources. Given this constraint, we propose to dynamically increase the intersocket bandwidth by sacrificing off-chip memory bandwidth when systems have heavy intersocket communication but few off-chip memory accesses. Our design increases the physical bandwidth for intersocket communication via switching the function of pins from off-chip memory accesses to intersocket communication and can achieve an average performance speedup of 1.28 in geocentric mean for selected parallel multithreaded benchmarks.**

***Index Terms*—Intersocket traffic, multicore system.**

I. Introduction

MULTISOCKET systems are widely used to boost the throughput of massive parallel workloads that generate intensive local traffic, between processors and off-chip memory devices such as DRAM, and remote traffic for intersocket communication (we denote *local* accesses as communications between a processor and memory; *remote* accesses as intersocket communications. Off-chip bandwidth is used for both types of accesses). The limited local bandwidth of main memory bounds the performance of parallel workloads, since it can cause the serialization of parallel memory requests and offset the benefit of memory level parallelism, especially considering the ever-increasing memory footprint of workloads and the number of cores per die. Many architects address this problem by boosting the system throughput via advanced algorithms for off-chip memory requests [11], increasing the physical memory bandwidth at the cost of lowering core frequency [6], or reducing traffic via using a stacked DRAM, which has higher bandwidth than off-chip memory devices and a larger size than an SRAM-based cache [9]. These solutions relieve the memory bandwidth bottleneck, causing intersocket bandwidth to emerge as a new performance bottleneck for workloads with intensive intersocket communication.

Remote bandwidth bounds the performance of workloads that frequently fetch data from the cache of other processors or remotely from main memory. Inadequate remote bandwidth serializes memory requests and limits the benefits of memory level parallelism. The bottleneck of intersocket communications, like QuickPath Interconnect (QPI) [2], is hidden when remote main memory access is constrained by off-chip bandwidth, but is now revealed by the volume of requests directly to the DRAM cache that do not use off chip bandwidth. The QPI bandwidth becomes a greater concern than off chip bandwidth when data is more likely to be fetched from stacked DRAM, which has superior bandwidth compared to the remote bandwidth. This bottleneck is shown in Fig. 1 that breaks down the latencies of un-core requests in several-selected benchmarks on simulated platform. The un-core requests hit on local DRAM cache, local main memory, remote cache, remote DRAM cache, or remote main memory.

The qSwitch, which dynamically allocates off-chip bandwidth between local and remote accesses, is proposed to relieve the bottleneck constraining remote accesses. The total number of pins is a scarce resource [25] that power delivery networks and I/O compete for. Additionally, increasing the total number of signal pins is prohibitive since routing traces beneath processors is becoming very difficult. The qSwitch dynamically shifts a portion of local off-chip bandwidth, used for accessing main memory, into remote intersocket communication bandwidth when detecting a low number of local access activities without increasing the total number of signal pins. qSwitch improves the performance of workloads suffering from limited intersocket bandwidth, based on a vertical design from the circuit to architecture level. We list the following contributions of this paper.

1) We identify the latency of intersocket communication as the major bottleneck for massive parallel workloads that intensively share data across sockets.
2) We propose qSwitch for improving the performance of workloads on a multisocket system in which switching

Manuscript received August 22, 2016; revised January 4, 2017 and March 14, 2017; accepted April 20, 2017. Date of publication May 17, 2017; date of current version December 20, 2017. This work was supported by the U.S. National Science Foundation under Grant CCF-1017961, Grant CCF-1422408, and Grant CNS-1527318. This paper was recommended by Associate Editor X. Li. *(Corresponding authors: Lu Peng; Weihua Zhang.)*

S. Chen, L. Peng, S. Irving, Z. Zhao, and A. Srivastava are with the Division of Electrical and Computer Engineering, School of Electrical Engineering and Computer Science, Louisiana State University, Baton Rouge, LA 70803 USA (e-mail: schen26@lsu.edu; lpeng@lsu.edu; sirvin1@lsu.edu; eesriv@lsu.edu).

W. Zhang is with the Software School, Fudan University, Shanghai 201203, China (e-mail: zhangweihua@fudan.edu.cn).

Color versions of one or more of the figures in this paper are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/TCAD.2017.2705154

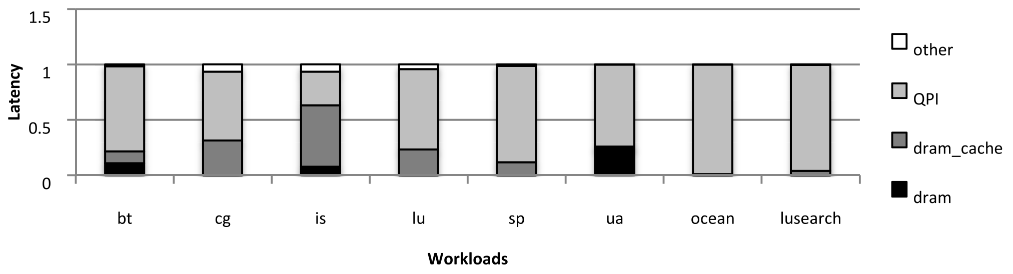


Fig. 1. Latency breakdown of un-core requests in the simulated system with two sockets.

agents turn on/off memory channels, QPI buses, and off-chip bus connections.

3) We evaluate the performance of qSwitch with the selected multithread workloads. We also investigate the runtime overhead and signal integrity for qSwitch.

II. Related Work

Many works are proposed such as increasing the throughput of main memory and the bandwidth of intersocket communication, since the long latencies of off-chip accesses has been identified as one of the bottlenecks for massive parallel workloads.

Researchers try to boost the throughput of main memory by modifying memory devices, the memory channels, and processors. For DRAM devices, row buffer misses reduce the utilization of the bandwidth since the program will incur a considerable overhead for turning on/off a row. The row buffer is proposed to break the inside of a bank into multiple subarrays, and thereby, reduce the row buffer miss rate, and have a lower overhead for switching the subarray instead of a whole row [12]. An asymmetric DRAM bank organization is proposed to improve the system performance via using larger rows for system throughput and smaller rows for lower overheads for turning on/off a row [21].

Several works improve the performance of main memory at the rank level. A conventional rank is broke down into mini-ranks that have a shorter data width, and can be operated individually for higher memory system throughput [26]. Increasing the bus frequency is proposed to improve the performance of memory channels via buffering data and commands in the DIMMs [1]. Splitting the data bus into several small buses is also proposed to boost the throughput of memory channels since each small data bus can work independently [23]. Dynamically increasing the bandwidth of the main memory is proposed but it has considerable parasitic capacity from power switches [6]. Our design only switches signal pins and does not have this issue.

From the processor side, methods are proposed to improve the performance via scheduling off-chip requests and using DRAM cache. A memory scheduler is proposed to boost system performance based on reinforcement learning that can understand program behaviors [8]. Another memory scheduler is designed to boost multithreaded performance by providing fair off-chip accesses for each thread [16], [17]. DRAM cache is proposed to reduce the number of off-chip accesses since it has superior bandwidth than main memory and larger size than SRAM-based cache [19]. Lowering the off-chip traffic and reducing the tag lookup latency further improve the performance of DRAM cache [9]. The works reduce the off-chip traffic between processors and main memory but do not affect the intersocket traffic.

Silicon photonics have been studied for a long time as a promising technology to replace the electrical off-chip buses and provide superior bandwidth with very low energy consumption [3]. It can boost the bandwidth of main memory while requires rearchitecting DRAM memory systems, and increase the bandwidth of interconnect [13]. A photonics interconnect has been developed [22] but is not widely used due to manufacturing costs and reliability issues [20]. The electrical chip-to-chip cost is 0.25$/Gbit, while the current parallel optic transceiver manufacturers state that perhaps $4/Gbit is achievable today. The reliability of silicon photonics interconnects is unclear since the integration of photonic emitters and receivers into the IC may cause some reliability issues. Our design is a cost-effective and reliable solution for intersocket traffic since it is based on conventional electrical interconnects.

III. Design Overview

We introduce two modes for a multisocket system: 1) the single-link mode in which the system has default bandwidth of off-chip memory and bandwidth of intersocket communication and 2) the multilink mode in which the system has multiplied bandwidth of intersocket communication at the cost of lower off-chip memory bandwidth. The two modes are shown in Fig. 2 as an example in which the system has two processors connected via a QPI bus with 20 lanes and the each processor has four memory channels. This example represents the typical case used in the following discussion easily extended for different system configurations. In this instance, the multilink mode multiplies the bandwidth of QPI by a factor of 3 and loses two memory channels since the number of pins for a memory channels is more than the number of pins for a QPI bus. This calculation is based on the fact that a memory channel requires 125 pins from a processor to access memory devices [1], while a QPI bus only demands 84 pins from processors [2].

This design needs a hardware unit to orchestrate mode switching by quickly detecting phases of intensive intersocket communication, as intensive phases may be sudden and short. The design introduces a switching agent for each socket to coordinate the increasing of intersocket communication and

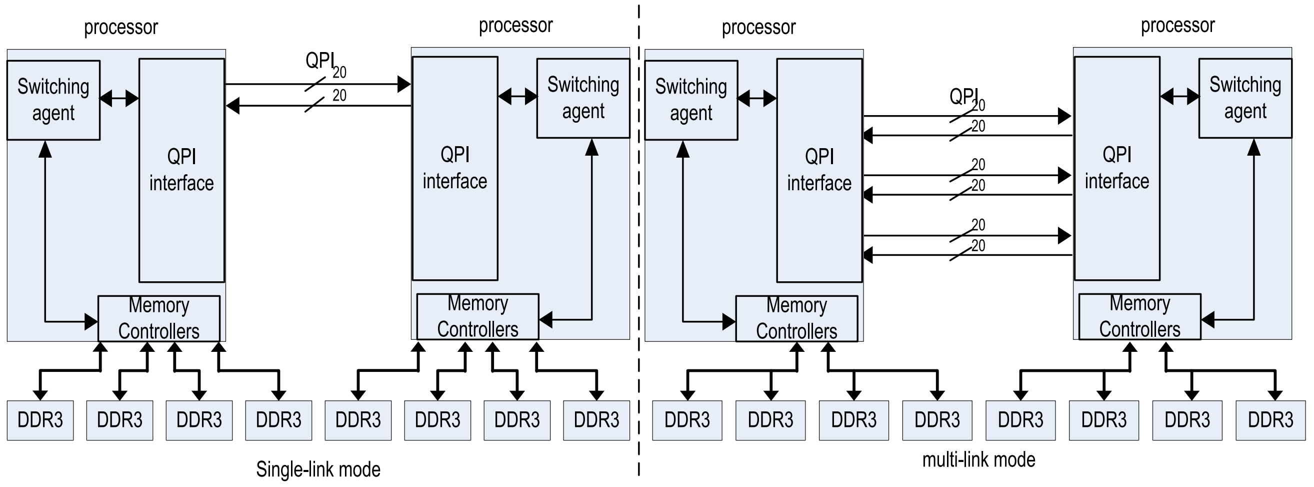


Fig. 2. Simulated system running in single-link and multilink modes.

the decreasing of off-chip memory bandwidth. This agent switches the function of switchable pins between accessing off-chip memory to communicating between sockets via signal switches sitting on the die and the motherboard. DRAM controllers are also manipulated so that they adapt to the less off-chip bandwidth and the QPI to utilize the extra bandwidth of intersocket communication. The switching agents from all processors have to reach an agreement that the system can increase its throughput via a switch, not just a subset of processors. With a bottom-up approach, we discuss the mechanism of switching off-chip bus connection as well as auxiliary circuits in Section III-A; the modifications of the DRAM controller and the QPI physical layer are addressed in Sections III-B and III-C; and the switching agents and the switching conditions are described in Sections III-D and III-E.

A. Off-Chip Connection

The modified off-chip connection is shown in Fig. 3 along with an auxiliary circuit called the "signal switch." We only show the related off-chip bus connection for a processor with a pair of QPI data lanes, since the simulated system is homogenous and the off-chip memory buses per socket are identical to each other.

In multilink mode, auxiliary circuits convert memory buses into additional QPI buses; processors can still read and write data from memory devices using the remaining memory buses. Disconnected memory devices connect to other memory buses as an extra rank in multilink mode, while they maintain dedicated buses in single-link mode. This design maintains the accessibility of data stored in memory devices though it requires extra circuits on the motherboard.

A signal switch is used to configure a pin for accessing off-chip memory devices or for intersocket communication, or to attach two memory channels to one another for increased data accessibility. The signal switch is a classic switch consisting of an *n*-type metal-oxide semiconductor (nMOS) and a *p*-type metal-oxide semiconductor (pMOS) each having a relatively low parasitic capacitance and propagation delay. This switch is ideal for high-speed signals that are sensitive to parasitic capacitance and signal delay.

With signal switches, we can increase the bandwidth of intersocket communication via switching the system from the single-link mode to the multilink mode. In single-link mode, pairs of signal switches (1) on the die connect pins to the memory controllers, and to a dedicated memory channel via pairs of signal switches (2) on the motherboard. The signal switches (3) detach the memory channel from another one and the system has four memory channels. The processor can write/read data from memory devices via the memory channels by turning on the signal switches in the corresponding direction. In multilink mode, the signal switches (1) and (2) connect the pins to the QPI buses instead of the memory channels, while the signal switches (3) attach the memory channel to another one and the system has two memory channels. The processor can access the memory devices via the two memory channels by turning on the signal switches (3) in the corresponding direction. The location of switches (2) and (3) on the motherboards are also vital to the signal integrity. The switches (2) should be placed close to the processors to reduce the signal reflection between the switches (1) and (2), while the switches (3) should be placed close to the DRAM devices for the same reason.

B. Memory Controllers

We modify the memory controllers to dynamically change the number of memory channels when the system switches between the single-link and multilink modes shown in Fig. 4. We turn on two memory controllers when the system switches from multilink mode to single-link mode and turn them off when switching back. The other two memory controllers handle all memory requests in single-link mode. Given a fixed address in mapping policy, this incurs a negligible area overhead to dispatch memory requests to the corresponding memory channels, and few extra pins to select the memory channel in single-link mode. The main challenge is that all memory

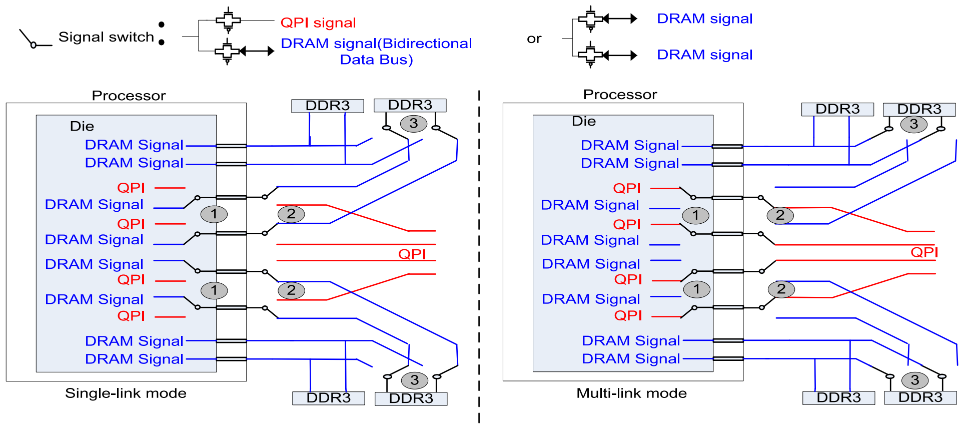


Fig. 3. Off-chip bus connection in single-link and multilink modes.

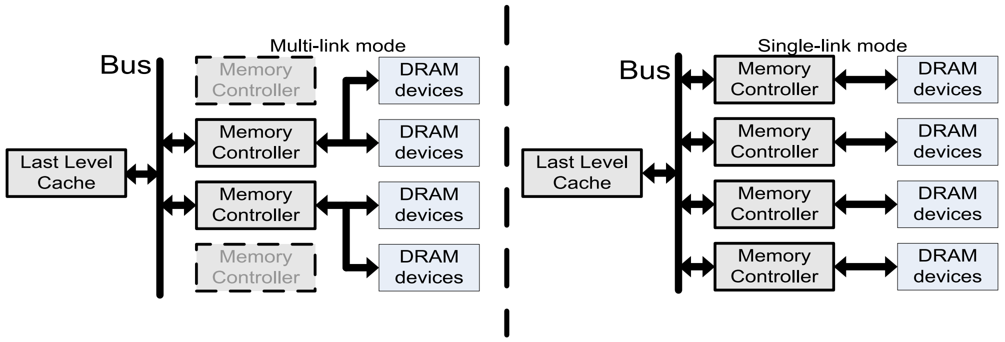


Fig. 4. Memory controller running in single-link and multilink modes.

requests have to be committed in memory controllers to switch the system between modes instead of migrating requests cross memory controllers. This overhead is discussed in the runtime overhead section.

The length of write and read request queues is halved when the system switches into multibus mode. This can potentially reduce the off-chip bandwidth for main memory. The slowdown is minimal due to low main memory access traffic in multibus mode. Additionally, we do consider the slowdown in our simulation.

Consolidating many memory channels could lead a channel to have too many ranks that exceed the standard, which may hurt the scalability of the design. The high speed of memory buses limits the maximal number of ranks in a memory channel. This constraint can be relaxed by lowering the frequency of the memory bus in multibus mode. This overhead could be negligible because traffic between the processors and main memory is low in multibus mode.

C. QPI Stack

QPI is a point-to-point processor interconnect with five layers including the physical layer, link layer, routing layer, transport layer, and protocol layer [2]. Each layer works independently from the other layers and we only discuss the physical and link layers that relate to our modification. We add the extra physical layers (PHY) that are fully connected with virtual networks in link layers to support more than

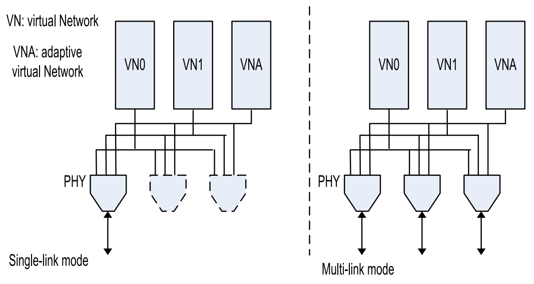


Fig. 5. Physical layers of QPI running in single-link and multilink modes.

one QPI bus shown in Fig. 5. The PHYs are powered off in the single-link mode, while they in the multilink mode can receive packages from other processors or send packages waiting in the buffers of virtual networks. The PHYs are buffer-less and thereby can be quickly turned on/off since the link layers control the traffic via credit/debit flow control. We employ switching agents to guarantee that there is no dropping package during the transitions between the multilink mode and the single-link mode. The switching agents enforce the PHYs can only send packages after the PHYs in the receiver side can accept the packages, when the system switches to the multilink mode. The switching agents also enforce the senders of the PHYs are turned off before the corresponding receivers of PHY are disabled, when the system switch to the single-link mode.

D. Switch Agents

To switch between the single-link mode and the multilink mode, we employ a switching agent inside each processor to coordinate the transitions in the two processors. One of the switching agents is preselected as the launcher, while all other switching agents are called as assistants. The launcher makes a switching decision and initializes the transition, then all assistants help processors complete the transition smoothly. All switching agents analyze the traffic of local off-chip memory access and that of intersocket communication via collecting hardware counters from the local memory controllers and the QPI controller, which consists of a sender and a receiver.

Based on this information, the switching agents take the following steps for a transition once they detect a phase in which the performance can be improved by switching the system to multilink mode.

1) The launcher detects the current phase and sends switching inquiries to other switching agents called assistants via the QPI buses. An assistant denies the switching inquiry by sending a disapproving response to the launcher if it does not detect this phase locally. Otherwise, the assistant accepts the inquiry by sending back an acknowledging response.
2) If the launcher receives a disapproving response, it immediately aborts this transition. Otherwise after it receives all acknowledging response, it turns off the two memory controllers, switches the off-chip connections, and turns on all the extra QPI receivers, while it initializes a transition by sending switching requests to all the assistants that also do the same thing locally once they receive the request.
3) After the switching completes, each switching agent sends responses to its neighboring switching agents.
4) After receiving a response from a neighbor, the switching agent turns on the QPI sender connecting to the neighbor. After all the extra QPI buses are connected, the transition concludes.

For switching the system to single-link mode, the switching agents take similar steps for the transition.

1) The launcher detects the phase and sends switching inquiries to other switching assistants via the QPI buses. The launcher will abort the transition if any switching assistants send back a disapproving response to the launcher.
2) After receiving acknowledging responses from all assistants, the launcher disables the extra QPI senders of all neighbors, while it sends switching requests to all assistants that take the same action. Each switching agent sends responses to its neighbors after the action is completed.
3) Once having received the response, a switching agent disables the corresponding QPI receivers. After it has received the responses from all neighbors, it switches the off-chip bus connection and then turns on the two memory controllers.
4) After every switching agent has taken this action, the transition completes.

Any switching agent can be selected as the launcher during system initialization. We ignore the overhead for leader election, since the election is only held once when all processors are powered on. Note that it is possible to switch a subset of processors into multilink mode while keeping others in single-link mode. A hybrid approach is not considered in this paper for to two reasons: most threads in workloads show similar phases and thereby most processors are likely to benefit from the same mode thus the benefit of a partial transition is minimal compared to that of a full transition.

Ideally, we can change the mode for each processor and its local memories individually. However, the additional benefit of switching processors individually is marginal since all processors stay in the identical mode. Most multithreaded workloads exhibit homogenous pattern of memory accesses, which means that the performance of all threads are bounded by local off-chip bandwidth or QPI bandwidth.

If multiprograms with same type of workloads such as memory intensive workloads or intersocket intensive workloads run together, they will tend to stay in the same mode. When they are different kind of workloads, switching to either mode improves the performance of some workloads but hurts the performance of other workloads due to less memory to processor bandwidth. So the key question becomes whether the performance improvement is worth at a cost of the slowdown of other workloads. The values of the performance improvement and the slowdown can be measured via switching the system into the opposite mode for a short interval. So whether the systems should switch to another mode depends on how to compare the two values based on the predefined fairness policy.

When the system has more than two processors, the system can pick up any two processors and increase the intersocket bandwidth for them, and then pick up another pair of processors after a certain duration. Utilizing the switches in this way is a simple and feasible solution to apply our design to a system with more than two processors.

E. Switching Condition

We use 0.1 ms as the minimum interval for a mode switch. The launcher will collect the number of un-core requests hitting locally and the number of un-core request hitting remotely. At the end of an interval, the launcher will initialize a switch from single-link mode to multilink mode if it observes that remote traffic is heavier than local traffic; or a switch from multi to single-link mode if it observes that local traffic is more intensive than remote traffic. The local traffic is gauged by consumed bandwidth and remote traffic is measured by outgoing bandwidth. The condition for switching the system into multilink mode is

$$(\mathrm{NQ}_{\mathrm{inbound}} + \mathrm{NQ}_{\mathrm{outbound}}) * \mathrm{SQ} > (\mathrm{NM}_{\mathrm{read}} + \mathrm{NM}_{\mathrm{write}}) * \mathrm{SM}$$

where $\mathrm{NQ}_{\mathrm{inbound}}$ and $\mathrm{NQ}_{\mathrm{outbound}}$ denote the numbers of QPI inbound and outbound packets, respectively, while SQ denotes the average packet size. $\mathrm{NM}_{\mathrm{read}}$ and $\mathrm{NM}_{\mathrm{write}}$ denote the numbers of reads and writes for local memory, respectively, and SM denotes the average size of memory requests.

When the condition is not satisfied, the system should switch back to the single-link mode.

To evaluate the performance of this dynamic switching, we introduce the baseline as a system that has no additional circuit and a configuration of the single-link mode. The baseline has an identical performance as the system that always stay in single-link mode. We also have static switching in which the system is permanently "switched" to multilink mode; and dynamic switching in which the system can dynamically switch between single-link and multilink mode.

F. Area Overhead & Propagation Delay

The area overheads of the design comes from the signal switches on the die, the modifications of QPI and the switching agent. The area of signal switches, consisting of a pair of large nMOS and pMOS, is negligible since the area of a signal switches is less than the area of 4000 transistors based on 45 nm technology. The extra QPI physical layers are bufferless and incur trivial area overhead. The switch agent for each processor also incurs a negligible area overhead since it uses a straightforward rule and only a few steps to coordinate the switches cross-processors, which are easy to implement in hardware.

The propagation delay caused by signal switches depends on the resistance and capacity of the load. We measure the propagation delays of the five cases shown in Fig. 6 by comparing the propagation delays with signal switches to the delays without them based on 45 nm technology using mentor graphic tools [15]. The longest propagation delays of the QPI bus and memory bus are 0.13 and 0.12 ns, respectively. The delay can be further reduced using better technology.

G. Runtime Overhead

We break down the runtime overhead of the transitions between the two modes into two parts: 1) the runtime overhead of turning on/off memory buses and 2) the runtime overhead of turning on/off QPI buses. The former mainly comes from restabilizing the signals on the memory buses and turning on/off the memory controllers. During the transitions, the memory devices are inaccessible and the processors do not send requests. We estimate this overhead mainly based on the runtime overhead of scaling DRAM frequency that is 512 memory cycles and 28 ns [7]. The overhead is estimated to be 0.67 μs given the 800 MHz memory frequency.

Additionally, we commit all the memory requests in the queue before turning off/on a memory channel. Given the read and write request queues in a memory channel have 32 total entries and each request takes 40 ns, the runtime overhead of turning on/off a memory channel can be estimated to be at most 1.28 μs which is still affordable. The total overhead of turning off/on memory buses is bounded by 1.95 μs.

The overhead of turning on/off QPI buses comes from restabilizing the signals on the QPI buses and turning on/off the QPI PHYs. Note that processors are not halted but cannot use the extra bandwidth of QPI buses during the transitions when the systems are switching to multilink mode. We conservatively estimate that switching QPI buses takes the same amount of time as switching memory buses. So the total runtime overhead is estimated to be 3.23 μs.

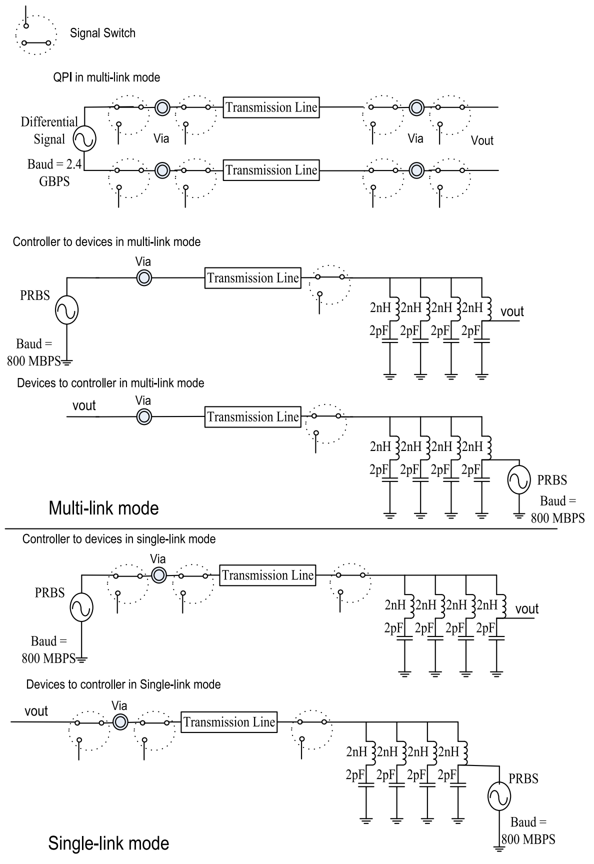
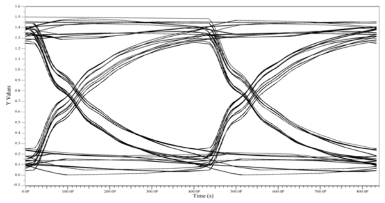

Signal Switch
QPI in multi-link mode
Differential Signal
Baud = 2.4 GBPS
Transmission Line
Via
Vout
Controller to devices in multi-link mode
PRBS
Baud = 800 MBPS
2nH
2pF
vout
Devices to controller in multi-link mode
Multi-link mode
Controller to devices in single-link mode
Devices to controller in Single-link mode
Single-link mode

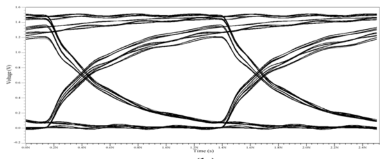
Fig. 6. SPICE models for QPI buses and memory buses in single-link and multilink modes.

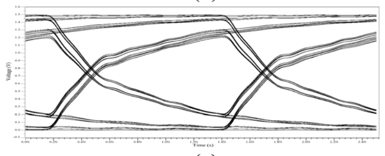
H. Signal Integrity

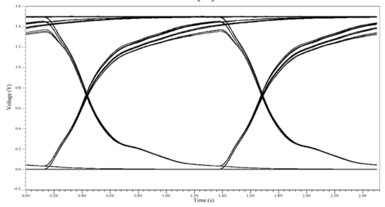
We setup up SPICE models in the two modes shown in Fig. 6, and test the signal integrity with Mentor Graphic tools [15] to prove that our design maintains signal integrity for the data path signals on memory buses and the data lane signals on the QPI bus. Memory bus signals are bidirectional with an 800 MHz frequency while the data lane on the QPI bus runs at a 2.4 GHz frequency.

For multilink mode, we show the eye diagram for the signal of a data lane on a QPI bus in Fig. 7(a). The eye diagram shows an open eye though it has some noise due to signal reflections. We also show the eye diagrams for a signal on the memory bus in Fig. 7(b) and (c). These diagrams also show open eyes though the signal in Fig. 7(b) suffers from signal reflections and the signal in Fig. 7(c) suffers from large capacity loads from memory devices. Additionally, we show the eye diagrams for single-link mode in Fig. 7(d) and (e) when data are read from memory devices or written into memory devices, each having clearer eyes compared to Fig. 7(b) and (c) due to fewer signal switches on the paths. These figures indicate that acceptable signal quality is retained in both scenarios.


(a)


(b)


(c)


(d)

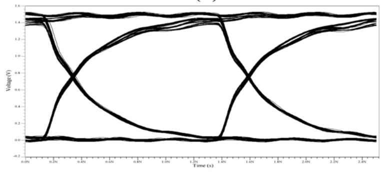

(e)

Fig. 7. Eye diagram of (a) QPI bus in multilink mode, (b) memory bus when reading data from devices in multilink mode, (c) memory bus when writing data to devices in multilink mode, (d) memory bus when reading data from devices in single-link mode, and (e) memory bus when writing data to devices in single-link mode.

IV. Experimental Setup

We set up our stimulated system with Sniper 6.1 [5] using the system configuration shown in Table I. The system has two processors with the configuration based on the Intel Xeon X5550. Each processor has four memory channels and one QPI bus while in single-link mode, and then has two memory channels and three QPI buses while in multilink mode. The energy consumption is estimated using the McPAT tool [14]. We also modified the simulator to include runtime overheads.

We also list the selected multithread workloads shown in Table II as well as the number of un-core requests per instruction, and the percentage of QPI latencies per the total un-core latencies. The workloads are selected from the NPB benchmarks [18], the Splash2 benchmarks [24], and the Decapo benchmarks [4]. Since the *lusearch* workload involves a considerable number of system calls, we run it in *jikes* research virtual machine [10] on the sniper simulator. Each workloads runs with eight threads and occupies all cores in the simulated system. We separate the workloads into those workloads exhibiting intensive intersocket communication and those workloads showing moderate or low intersocket communication. Most experiments are conducted using the communication-intensive workloads that reveal the benefits of multilink mode, while we use the nonintensive workloads to compare the performances of static and dynamic switching. We fast forward workloads into selected regions that show intensive intersocket traffic, and then warm up the cache for 1 billion instructions. We run a total of 800 million instructions for each workload since some threads, e.g., the garbage collector, run comparatively few instructions in

TABLE I
Configuration of the Simulated System

Component	**Parameters**
System	two processors
Processor	4 cores
Core	2.66 GHz, 4-way issue, 128-entry ROB hybrid local/global predictor
Cache Line Size	64B, LRU replacement
L1-I	32KB, 4 way, 4 cycle access time
L1-D	32KB, 8 way, 4 cycle access time
L2 cache	256 KB per core, 8 way, 8 cycle
L3 cache	shared 8 MB, 16 way, 30 cycle
Coherence protocol	MSI
DRAM	line-interleaved mapping, 34.1GB/s
DRAM cache	128 MB, 16 way, 512GB/s
QPI bus	20 link width, 3.2GHz, 25.6 GB

TABLE II
Selected Workloads

Workloads	**Benchmark Suite**	**Un-core Requests Per 1K Inst.**	**QPI Latency Percentage**
Workloads with intensive inter-socket traffic			
bt	NPB	3.70	77%
cg	NPB	20.44	62%
is	NPB	20.54	30%
lu	NPB	3.51	72%
sp	NPB	10.52	86%
ua	NPB	4.15	74%
ocean	Splash2	13.87	99%
lusearch	Decapo	8.92	95%
Workloads with low-to-moderate inter-socket traffic			
ft	NPB	6.89	61%
mg	NPB	15.36	30%
fmm	Splash2	0.15	27%
radiosity	Splash2	0.53	35%
raytrace	Splash2	0.95	25%

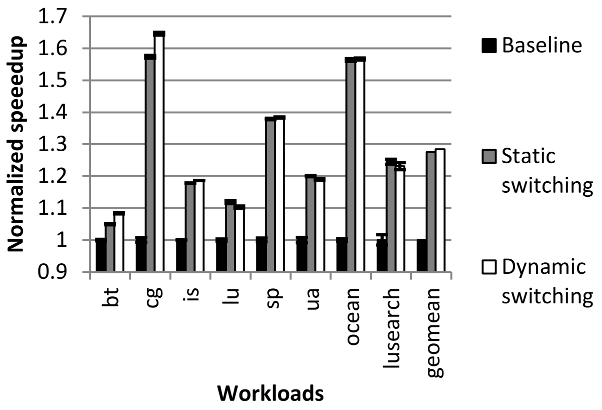


Fig. 8. Normalized speedup of static switching and dynamic switching normalized against the baseline.

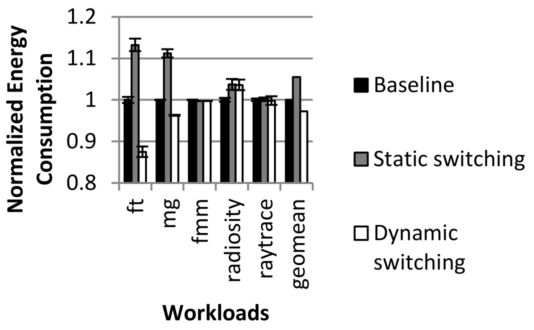


Fig. 9. Latency of un-core requests for static switching and dynamic switching normalized against that of the baseline.

TABLE III
INTERVALS IN THE MULTILINK MODE AND IN THE SINGLE-LINK MODE AS WELL AS THE TIMES OF SWITCHING TO THE MULTILINK MODE AND THE SINGLE-LINK MODE

	The multi-link mode	The Single-Link Mode	The switching to the multi-link mode	The switching to the Single-Link Mode
Workloads of intensive inter-socket traffic				
bt	235	32	2	1
cg	539	12	2	1
is	451	1	1	0
lu	335	7	2	1
sp	403	2	2	1
ua	354	9	4	3
ocean	465	2	2	1
lusearch	463	4	4	3
Workloads of moderate and low inter-socket traffic				
ft	209	83	3	2
mg	570	185	7	6
fmm	135	1	1	0
radiosity	524	27	18	17
raytrace	1577	19	18	17

the workload *lusearch*. We also run each workload five times and show the 95% confidence intervals for performance comparisons.

V. RESULTS

A. Performance of the Static Switching

We evaluate the performances of the baseline, in which the system runs in single-link mode; the system using static switching; and using dynamic switching. Fig. 8 shows the results of static switching and dynamic switching normalized against the results of the baseline for each workload. Static switching and dynamic switching gain a performance improvement of 28% and 29%, respectively, compared to the baseline. We also show the reduced latencies of un-core requests in static switching normalized against that in the baseline shown in Fig. 9. Note the latencies only account for the latencies incurred outside the cores. The workloads *cg* and *ocean* achieve speedups of 1.54 and 1.55, respectively, which are much more than other workloads, since they have intensive un-core traffic and their un-core latencies are significantly reduced by multilink mode. The other workloads gain moderate performance improvements. For example, *is* also has intensive un-core traffic but sees a smaller reduction of the latencies, while *lusearch* has a significant reduction of the latencies but moderate un-core traffic.

B. Performance and Energy Efficiency of the Dynamical Switching for Intensive Intersocket Traffic workloads

Dynamic switching can gain a similar performance improvement for the workloads in Fig. 8 since it can detect phases of intensive intersocket communication. For example, dynamic switching improves the performance of the *cg* workload from 1.57 to 1.64, since it finds a period of low intersocket traffic and guides the system switches back to the multilink mode. Dynamic switching provides approximately the same performance improvements as static switching for workloads that have only a few intervals of low intersocket traffic. We also list the number of the intervals that the system is in multilink mode or in single-link mode in Table III, as well as the number of times that the system switches to multilink mode or single-link mode. The extra benefits of dynamic switching for the *cg* workload come from the system switching back to single-link mode when it catches a consecutive series of 12 intervals in which the system exhibits low intersocket communication but moderate local traffic.

We investigate the energy consumptions of static and dynamic switching normalized against that of the baseline for each workload, respectively, shown in Fig. 10. Static switching and dynamic switching reduces the average energy consumption by 12% and 13% in geometric mean since it improves the system performance with minimal energy overhead. The more performance improvement that multilink mode has gained, the more energy consumption is reduced. For example, the workloads *cg* and *ocean* save 30% and 22% more energy than the others for dynamic switching, while they also achieve more performance benefits compared to others.

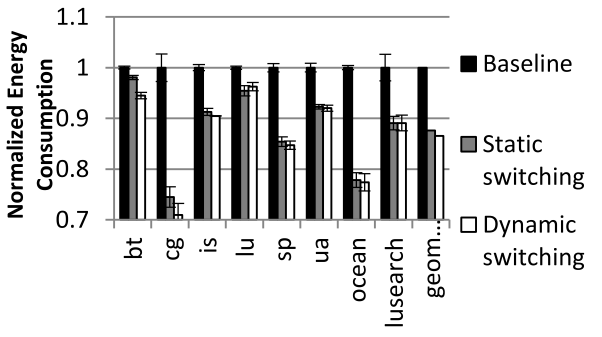


Fig. 10. Energy consumption while using static switching and dynamic switching normalized against the baseline.

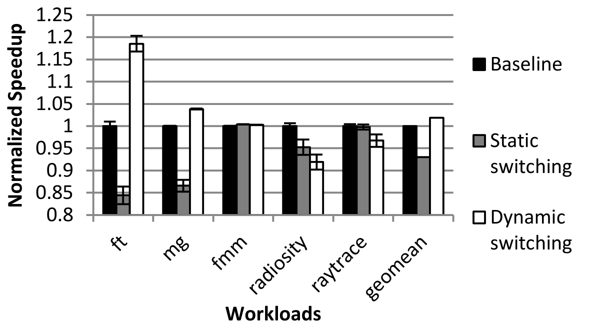


Fig. 11. Normalized speedup while using static switching and dynamic switching compared with the baseline for workloads with low-to-moderate intersocket traffic.

C. Performance and Energy Efficiency of the Dynamical Switching for Moderate and Low Intersocket Traffic Workloads

We also test the performance of dynamic switching compared with static switching for workloads exhibiting low-to-moderate intersocket traffic shown in Fig. 11. Static switching loses performance for these workloads, while dynamic switching keeps the performance or even gains a modest performance improvement. The results are normalized against the performance of the baseline for each workload, respectively. The dynamic switching gains are a normalized speedup of 1.18 and 1.04 for the *ft* and *mg* workloads, respectively, while static switching only achieves a normalized speedup of 0.84 and 0.87 individually. We also present the breakdown of normalized un-core latency in Fig. 12. All latencies are normalized against the total un-core latency of the baseline for each workload. Dynamic switching captures several stable periods in which performance can be improved via switching the system back to single-link mode. Dynamic switching reduces the QPI latency significantly for *ft* and *mg* while it increases DRAM latency slightly compared to static switching since these two workloads show long consecutive intensive intersocket traffic. Dynamic switching suffers from spikes of intensive local traffic that are hardly captured and thus only achieves a speedup of 0.91 for *radiosity*, which is close to the performance of static switching. Even though, the proposed dynamic switching can still achieve a geometric mean speedup of 1.02 for the five benchmarks with low-to-moderate intersocket traffic. This result implies that dynamic switching, combing static switching, and switch condition, does not impair the performance of applications with low-to-moderate intersocket traffic workloads.

We also present the energy consumptions of static and dynamic switching normalized against that of the baseline for each workload, as shown in Fig. 13. Static switching increases the average energy consumption by 4% while dynamic switching decreases it by 2% in geometric mean. The larger the performance improvement that dynamic switching has gained, the more the energy consumption is reduced. For example, dynamic switching reduces the power consumption of the workload *fg* by 8% and also boosts performance by more than other workloads.

D. Enhancement From Stride Prefetcher

We investigate the performances of dynamic switching and the baseline combined with a stride prefetcher shown in Fig. 14. The results are normalized against the baseline without a prefetcher and we show the performances with prefetchers that have a prefetch degree of 1 and 4, which denotes the number of prefetches issued on every memory reference. For the workloads with intensive intersocket traffic, the performances of the baseline with the prefetchers of degree 1 and 4 are 1.002 and 1.11, respectively, in the geometric mean; the performances of static switching with the prefetcher are 1.28 and 1.55; and the performance of dynamic switching are 1.30 and 1.56, respectively. For the workloads with low-to-moderate intersocket traffic, the performances of the baseline with prefetchers are 1.04 and 1.13 and the performances of dynamic switching are 1.05 and 1.14, respectively. The aggressive prefetchers shift the performance bottlenecks toward the QPI especially for the *is* workload. This can be verified in Fig. 15, which shows the percentage of QPI latencies for the baseline and the baseline with prefetchers of degree 1 and 4. This percentage of latency for the *is* workload is increased significantly when the prefetching degree is increased from 1 to 4, since the prefetcher now exploits the high bandwidth of the DRAM cache via increasing the memory level parallelism and thus reduces DRAM cache latencies but suffers from a limited QPI bandwidth, which offsets the benefit of memory level parallelism. Prefetchers boost the percentage of QPI latencies for the *is* workload since its un-core latencies in the DRAM cache are considerable even running on the baseline without a prefetcher compared with other workloads. Additionally, the aggressive prefetchers increase the latency of local DRAM access; though the latency of QPI is also increased, it is less important than the latency of the local DRAM. Dynamic switching actually keeps the system in the single-link mode.

We also evaluate the performance of dynamic switching using different configurations of the DRAM cache. This paper heavily relies on the DRAM cache's superior bandwidth, compared to off-chip main memory devices and the QPI, and thus, we want to verify the substantial benefit of dynamic switching in the broad design space of the DRAM cache.

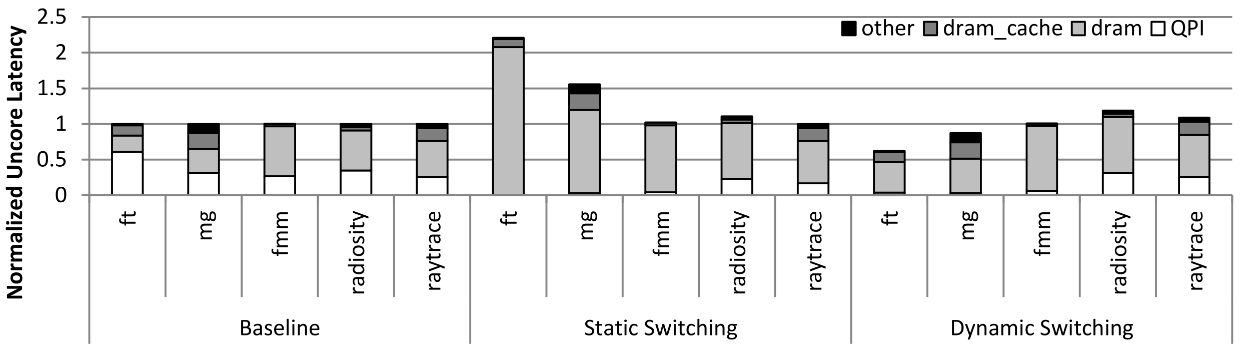


Fig. 12. Normalized un-core latency for the workloads with low-to-moderate intersocket traffic.

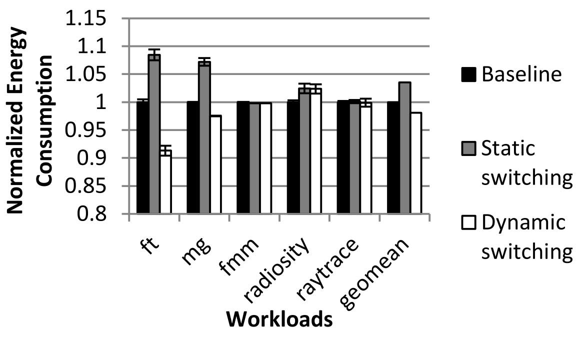


Fig. 13. Energy consumption while using static switching and dynamic switching normalized against the baseline for workloads with low-to-moderate intersocket traffic.

E. Bandwidth of the DRAM Cache

We investigate the performance improvement of dynamic switching with different bandwidths of DRAM caches as shown in Fig. 16. We vary the bandwidths from 128 to 512 Gb/s and compare the performances of dynamic switching and the baseline with the same DRAM cache bandwidth. Fig. 16 shows the performance of dynamic switching normalized against the performance of the baseline accordingly. The performance improvements are 1.27, 1.28, and 1.28 in the geometric mean for the DRAM cache bandwidths of 128, 256, and 512 Gb/s, respectively, for the workloads with intensive intersocket traffic, while the performances are all 1.02 in the geometric mean for the workloads with low-to-moderate intersocket traffic. The relatively stable improvements indicate that the benefit of dynamic switching is consistent as long as the bandwidth of cache DRAM is larger than that of the main memory.

F. Size of DRAM Cache

We also evaluate the performance improvement of dynamic switching using different DRAM cache sizes (32, 64, and 128 MB). When the size is increased, more un-core requests hit the DRAM cache which has much more bandwidth compared to off-chip memory devices. On the other hand, decreasing the DRAM cache's size will decrease its impact on the performance. Dynamic switching achieves an average of 20%, 21% and 28% performance improvement in geometric mean, which are normalized against the performance in the baseline with the same DRAM cache size, respectively, shown in Fig. 18 for the workloads with intensive intersocket traffic. Dynamic switching's relative performance is 1.01, 1.01, and 1.02 for the workloads with low-to-moderate intersocket traffic. The performance improvements for most workloads increase slightly as the size of DRAM cache is increased, while the performance improvement of the workload *ocean* increases quickly from -6% to 56% due to more remote requests hitting DRAM caches, which can be verified by the reduced latencies of un-core requests for *ocean* in Fig. 17. Fig. 17 also shows the latencies of un-core requests in multi-link mode normalized against the latencies in single-link mode. The figure shows most workloads slightly reduce the latency of un-core requests as the DRAM cache size is increased, while the latencies for *ocean* are reduced from 1.58 to 0.136. Decreasing the size of the DRAM cache from 128 to 32 MB causes the percentage of un-core latency caused by QPI latency to drop for the baseline running the *ocean* workload, while the percentage un-core latency caused by off-chip main memory latency increases from 0.002% to 86%. Additionally, varying the size of the DRAM cache shows only a slight impact on the performance and the latency of un-core requests, since a considerable portion of un-core requests hit the local DRAM instead of the DRAM cache.

G. Frequency of QPI Buses

We investigate the performance improvement of dynamic switching with different QPI bus frequencies (2.4, 3.2, and 4.8 GHz). 2.4 GHz is the lowest frequency of the QPI buses, while 4.8 GHz is first introduced on the Hashwell-E/EP platform. Boosting the frequency of QPI buses can gain more bandwidth but increases power consumption and poses more difficulties for routing QPI traces on the motherboard. Fig. 19 shows the performance improvement of dynamic switching normalized against the performance of the baseline with different QPI bus frequencies. Dynamic switching with QPI frequencies 2.4, 3.2, and 4.8 GHz achieves average speedups of 1.44, 1.28, and 1.15, respectively, in geometric mean for the workloads with intensive intersocket traffic. It reaches the average speedups of 1.5, 1.02, and 0.99 for the workloads with low-to-moderate intersocket traffic. Our design can gain a moderate performance improvement with

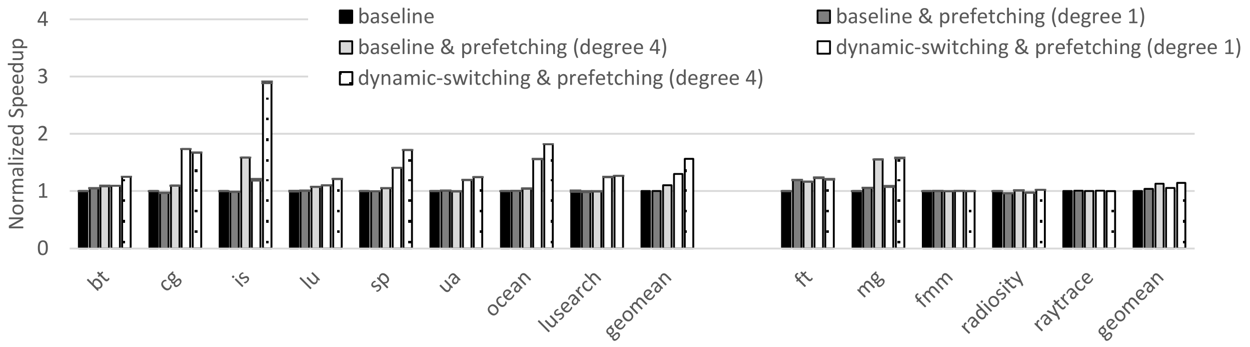


Fig. 14. Normalized speedup of dynamic switching compared to the baseline for various prefetching degrees.

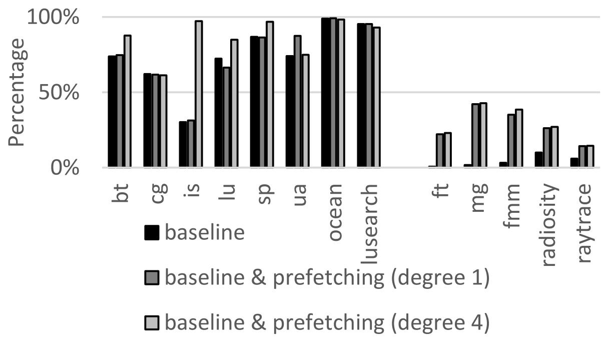


Fig. 15. Average percentage of un-core latencies caused by QPI latency for the baseline with various degrees of prefetching (average QPI latency/average un-core latency).

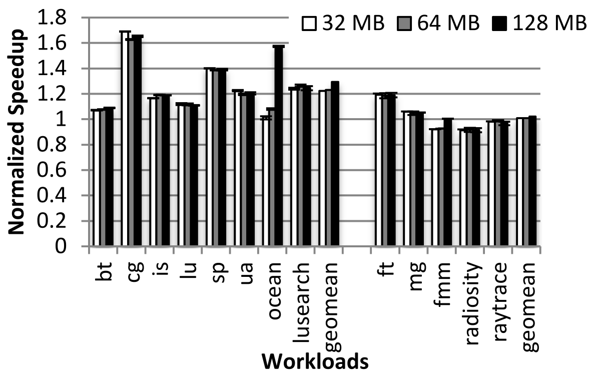


Fig. 18. Normalized speedup of dynamic switching for various DRAM cache sizes.

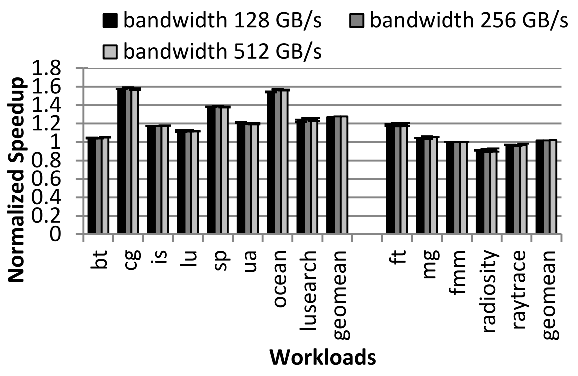


Fig. 16. Normalized speedup of dynamic switching for various DRAM cache bandwidths.

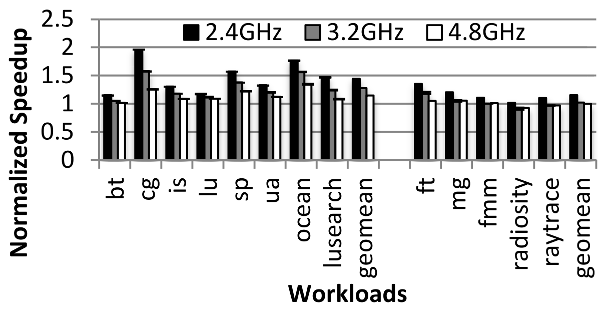


Fig. 19. Normalized speedup of dynamic switching for various QPI frequencies.

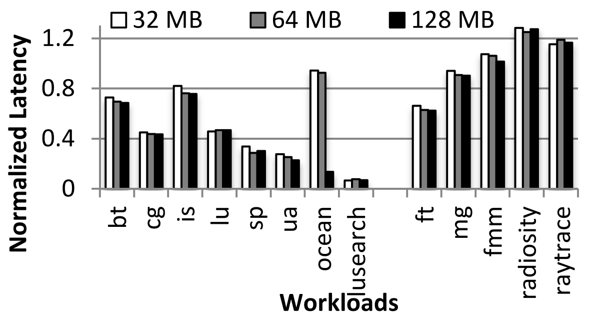


Fig. 17. Normalized latency of un-core requests using dynamic switching for various DRAM cache sizes.

the high frequency of 4.8 GHz, but sees a significant performance improvement with the low 2.4 GHz frequency for the workloads with intensive intersocket traffic. Fig. 20 shows the percentage of the un-core latencies in QPI in the total un-core latencies for the baseline with the different QPI bus frequencies. The ratio of QPI decreases as the frequency of QPI buses increases, which shortens the time of transferring data over the QPI buses and the waiting time of packets in the QPI. The ratio for the *cg* workload drops from 79% to 43% as the frequency of the QPI buses increase from 2.4 to 4.8 GHz, while the performance benefit of dynamic switching decreases from 1.96 to 1.25. The 4.8 GHz frequency of the QPI buses increases the percentage of the latencies from the DRAM cache with respect to the total un-core latency. For example, the frequency increases the percentage from 17% to 48% for the *cg* workload when the QPI bus frequency increases from 2.4 to 4.8 GHz. Additionally, dynamic switching gains a 15%

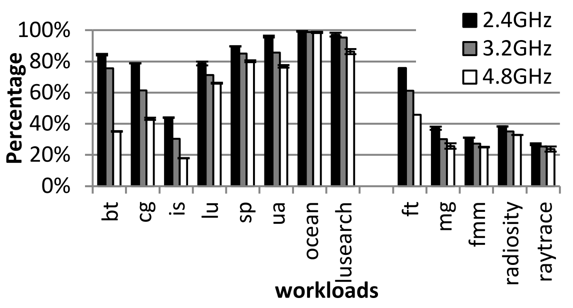


Fig. 20. Average percentage of un-core latency caused by QPI latency for the baseline for various QPI bus frequencies (average QPI latency/average un-core latency).

speedup when the QPI bus frequency is 2.4 GHz and maintains performance when it is 3.2 and 4.8 GHz.

VI. Conclusion

Multisocket systems are widely used for massive parallel workloads to improve throughput. The performance of multisocket system suffers from limited off-chip bandwidth confined by the scarce resource of processor pins. This problem can be relieved by the DRAM cache that is introduced to reduce the long latency of off-chip access via providing a large space to hold data and superior bandwidth to reduce queuing delay. The DRAM cache reduces the latencies of accessing main memory as the main contributor of the un-core latencies, while the latencies of intersocket communication emerge as a considerable bottleneck for workloads that frequently fetch data from remote memory.

The qSwitch design is proposed to reduce the intersocket latencies at the cost of local memory bandwidth, since the DRAM cache significantly reduces the number of off-chip local requests, and thereby, the local memory bandwidth becomes excessive in some cases. We design qSwitch from the off-chip bus connection to the switching agents in order to smoothly switch the system between two modes: 1) single-bus mode and 2) multibus mode. We investigate the signal integrity and discuss the design overhead to verify its feasibility. We also evaluation the performance benefits of qSwitch using different configurations of the DRAM cache and QPI to show the benefits exist in a broad design space.

This paper identifies the latency of intersocket communication as one of the performance bottlenecks in the era of DRAM cache for massive parallel workloads. Our results imply that the performance of the workloads can be improved via the optimization of intersocket communication such as wisely scheduling remote requests or reducing unnecessary remote requests. Furthermore, the limited bandwidth of intersocket communication could become increasingly painful as the number of cores on a die increases and more cores share bandwidth. Scaling the intersocket bandwidth with the number of cores is likely to be a challenge in the near future.

Acknowledgment

The authors appreciate the invaluable comments from the anonymous reviewers. They also acknowledge the computing resources provided by the Louisiana Optical Network Initiative HPC team.

References

[1] *4th Generation Core Family Desktop*. Accessed on May 23, 2017. [Online]. Available: http://www.intel.com/content/dam/www/public/us/en/documents/datasheets/4th-gen-core-family-desktop-vol-1-datasheet.pdf

[2] *An Introduction to the Intel® QuickPath Interconnect*, Intel, Santa Clara, CA, USA, 2009.

[3] S. Beamer *et al.*, "Re-architecting DRAM memory systems with monolithically integrated silicon photonics," in *Proc. ISCA*, Saint-Malo, France, 2010, pp. 129–140.

[4] S. M. Blackburn *et al.*, "The DaCapo benchmarks: Java benchmarking development and analysis," in *Proc. OOPSLA*, Portland, OR, USA, 2006, pp. 169–190.

[5] T. E. Carlson, W. Heirman, and L. Eeckhout, "Sniper: Exploring the level of abstraction for scalable and accurate parallel multi-core simulation," in *Proc. SC*, Seattle, WA, USA, 2011, Art. no. 52.

[6] S. Chen *et al.*, "Increasing off-chip bandwidth in multi-core processors with switchable pins," in *Proc. ISCA*, Minneapolis, MN, USA, 2014, pp. 385–396.

[7] Q. Deng, D. Meisner, L. Ramos, T. F. Wenisch, and R. Bianchini, "MemScale: Active low-power modes for main memory," in *Proc. ASPLOS*, Newport Beach, CA, USA, 2011, pp. 225–238.

[8] E. Ipek, O. Mutlu, J. F. Martinez, and R. Caruana, "Self-optimizing memory controllers: A reinforcement learning approach," in *Proc. ISCA*, 2008, pp. 39–50.

[9] D. Jevdjic, S. Volos, and B. Falsafi, "Die-stacked DRAM caches for servers: Hit ratio, latency, or bandwidth? Have it all with footprint cache," in *Proc. ISCA*, 2013, pp. 404–415.

[10] *Jikes RVM*. Accessed on May 23, 2017. [Online]. Available: http://www.jikesrvm.org

[11] Y. Kim, D. Han, O. Mutlu, and M. Harchol-Balter, "ATLAS: A scalable and high-performance scheduling algorithm for multiple memory controllers," in *Proc. HPCA*, 2010, pp. 37–48.

[12] Y. Kim, V. Seshadri, D. Lee, J. Liu, and O. Mutlu, "A case for exploiting subarray-level parallelism (SALP) in DRAM," *SIGARCH Comput. Architect. News*, vol. 40, no. 3, pp. 368–379, Jun. 2012.

[13] A. V. Krishnamoorthy *et al.*, "Computer systems based on silicon photonic interconnects," *Proc. IEEE*, vol. 97, no. 7, pp. 1337–1361, Jul. 2009.

[14] S. Li *et al.*, "McPAT: An integrated power, area, and timing modeling framework for multicore and manycore architectures," in *Proc. MICRO*, 2009, pp. 469–480.

[15] *Mentor Graphic*. Accessed on May 23, 2017. [Online]. Available: https://www.mentor.com

[16] O. Mutlu and T. Moscibroda, "Stall-time fair memory access scheduling for chip multiprocessors," in *Proc. MICRO*, Chicago, IL, USA, 2007, pp. 146–160.

[17] O. Mutlu and T. Moscibroda, "Parallelism-aware batch scheduling: Enhancing both performance and fairness of shared DRAM systems," in *Proc. ISCA*, 2008, pp. 63–74.

[18] *NAS Parallel Benchmarks*. Accessed on May 23, 2017. [Online]. Available: https://www.nas.nasa.gov/publications/npb.html

[19] M. K. Qureshi and G. H. Loh, "Fundamental latency trade-off in architecting DRAM caches: Outperforming impractical SRAM-tags with a simple and practical design," in *Proc. MICRO*, Vancouver, BC, Canada, 2012, pp. 235–246.

[20] W. S. Ring, "Silicon photonics: Challenges and future," Optoelectron. Ind. Develop. Assoc., Washington, DC, USA, OIDA Forum Tech. Rep., 2007.

[21] Y. H. Son, O. Seongil, Y. Ro, J. W. Lee, and J. H. Ahn, "Reducing memory access latency with asymmetric DRAM bank organizations," in *Proc. ISCA*, 2013, pp. 380–391.

[22] (2010). *The 50G Silicon Photonics Link Intel, Labs White Paper*. Accessed on May 23, 2017. [Online]. Available: http://download.intel.com/pressroom/pdf/photonics/Intel_SiliconPhotonics50gLink_WhitePaper.pdf?iid=pr_smrelease_vPro_materials2

[23] A. N. Udipi *et al.*, "Rethinking DRAM design and organization for energy-constrained multi-cores," in *Proc. ISCA*, Saint-Malo, France, 2010, pp. 175–186.

[24] S. C. Woo, M. Ohara, E. Torrie, J. P. Singh, and A. Gupta, "The SPLASH-2 programs: Characterization and methodological considerations," in *Proc. ISCA*, 1995, pp. 24–36.

[25] R. Zhang, K. Wang, B. H. Meyer, M. R. Stan, and K. Skadron, "Architecture implications of pads as a scarce resource," in *Proc. ISCA*, Minneapolis, MN, USA, 2014, pp. 373–384.

[26] H. Zheng *et al.*, "Mini-rank: Adaptive DRAM architecture for improving memory power efficiency," in *Proc. MICRO*, 2008, pp. 210–221.

Shaoming Chen received the bachelor's and master's degrees in electronics and information engineering from the Huazhong University of Science and Technology, Wuhan, China, in 2008 and 2011, respectively, and the Ph.D. degree in electrical and computer engineering from Louisiana State University, Baton Rouge, LA, USA, in 2016.

He is a Senior Designer with AMD, Austin, TX, USA. His current research interests include submemory system design and cost optimization of data centers.

Lu Peng received the bachelor's and master's degrees in computer science and engineering from Shanghai Jiao Tong University, Shanghai, China, and the Ph.D. degree in computer engineering from the University of Florida, Gainesville, FL, USA.

He is a Gerard L. "Jerry" Rispone Associate Professor with the Division of Electrical and Computer Engineering, Louisiana State University, Baton Rouge, LA, USA. His current research interests include memory hierarchy system, reliability, power efficiency, and other issues in processor design.

Dr. Peng was a recipient of the ORAU Ralph E. Power Junior Faculty Enhancement Awards in 2007 and the Best Paper Award (processor architecture track) from IEEE International Conference on Computer Design in 2001.

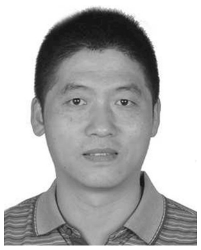
Samuel Irving received the bachelor's degrees in both computer science and electrical engineering from Louisiana State University (LSU), Baton Rouge, LA, USA, in 2011, where he is currently pursuing the Ph.D. degree in computer engineering.

His current research interests include machine learning, big data analytics, and heterogeneous architecture design.

Mr. Irving was a recipient of the Donald W. Clayton Ph.D. Assistantship at LSU.

Zhou Zhao received the B.S. degree in automation and the M.S. degree in circuit and system from the University of Electronic Science and Technology of China, Chengdu, China, in 2011 and 2014, respectively. He is currently pursuing the Ph.D. degree in electrical engineering with Louisiana State University, Baton Rouge, LA USA.

His current research interests include low power very large scale integration design, power management and signal integrity in chip multiprocessor, and logic gate design using emerging devices.



Weihua Zhang received the Ph.D. degree in computer science from Fudan University, Shanghai, China, in 2007.

He is currently an Associate Professor of Parallel Processing Institute, Fudan University. His current research interests include compilers, computer architecture, and parallelization and systems software.

Ashok Srivastava (LSM'15) received the M.Tech. and Ph.D. degrees in solid-state physics and semiconductor electronics from the Indian Institute of Technology Delhi, New Delhi, India, in 1970 and 1975, respectively.

He is a Wilbur D. and Camille V. Fugler, Jr., Professor of Electrical and Computer Engineering, Louisiana State University (LSU), Baton Rouge, LA USA. Before joining LSU, he was with several academic institutions and research and development laboratories, where his main areas of research were very large scale integration (VLSI) design and technology and emerging logic devices. While at LSU, he has held visiting appointments at several institutions across the globe. He has authored a book entitled *Carbon-Based Electronics: Transistors and Interconnects at Nanoscale* and a Co-Editor of two books entitled *Nano-CMOS and Post-CMOS Electronics* and *Devices and Modeling, Circuits and Design* (IET Press). He has authored and co-authored over 170 research papers in journals and conferences covering devices, circuits and systems, micro/nano-systems, holds 1 U.S. patent and supervised 41 graduate students. His current research interests include low-power VLSI circuit design and testability, nanoelectronics–nanoscale devices, circuits and integration, nonclassical device electronics with focus on carbon nanotube, graphene, and other 2-D material-based device electronics for emerging post-CMOS integrated circuit design.

Dr. Srivastava was a recipient the Prestigious 1979–1980 UNESCO Fellowship Award. He is a Life Senior Member of Electron Devices, Circuits and Systems, and Solid-State Circuits Societies, a Senior Member of SPIE, and a member of ASEE.