Alloy: Parallel-Serial Memory Channel Architecture for Single-Chip Heterogeneous Processor Systems

Hao Wang[1], Chang-Jae Park[2], Gyung-su Byun[3], Jung Ho Ahn[4], Nam Sung Kim[1]

[1]The University of Wisconsin-Madison
{hwang223, nskim3}@wisc.edu

[2]Samsung Electronics
changjae012.park@samsung.com

[3]Southern Methodist University
gsbyun@smu.edu

[4]Seoul National University
gajh@snu.ac.kr

Abstract

A single-chip heterogeneous processor integrates both CPU and GPU on the same chip, demanding higher memory bandwidth. However, the current parallel interface (e.g., DDR3) can increase neither the number of (memory) channels nor the bit rate of the channels without paying high package and power costs. In contrast, the high-speed serial interface (HSI) can offer much higher bandwidth for the same number of pins and lower power consumption for the same bandwidth than the parallel interface. This allows us to integrate more channels under a pin and/or package power constraint but at the cost of longer latency for memory accesses and higher static energy consumption in particular for idle channels. In this paper, we first provide a deep understanding of recent HSI exhibiting very distinct characteristics from past serial interfaces in terms of bit rate, latency, energy per bit transfer, and static power consumption. To overcome the limitation of using only parallel or serial interfaces, we second propose a hybrid memory channel architecture–Alloy consisting of low-latency parallel and high-bandwidth serial channels. Alloy is assisted by our two proposed techniques: (i), a memory channel partitioning technique adaptively maps physical (memory) pages of latency-sensitive (CPU) and bandwidth-consuming (GPU) applications to parallel and serial channels, respectively, and (ii) a power management technique reduces the static energy consumption of idle serial channels. On average, Alloy provides 21% and 32% higher performance for CPU and GPU, respectively, while consuming total memory interface energy comparable to the baseline parallel channel architecture for diverse mixes of co-running CPU and GPU applications.

Keywords

Memory architecture; Serial memory interface; Heterogeneous processors

1. Introduction

The off-chip (main) memory bandwidth has been a critical shared resource for chip multiprocessors (CMPs), but it has increased at a far lower rate than the number of cores per chip over technology generations. In particular, this problem is exacerbated for single-chip heterogeneous processors (SCHPs) integrating both CPU and GPU on the same chip [1,2], because GPU applications typically demand considerably higher bandwidth than CPU applications.

High-bandwidth Memory: 3D-stacked DRAM can provide substantially higher bandwidth for the memory system [3]. Yet, its capacity does not scale with the increasing capacity demand due to thermal and mechanical reliability issues; 3D-stacked DRAM should be jointly used with the conventional off-chip DRAM for the memory system [4]. 2.5D-stacked DRAM using the silicon interposer technology alleviates the thermal reliability issue of the 3D-stacked DRAM [5]. However, such 2.5D-stacked DRAM is more expensive than 3D-stacked DRAM because of higher manufacturing complexity and thus lower yield [6], limiting its use to the memory system for a niche high-end market. Hence, the off-chip memory still remains as a valuable option if its bandwidth can be increased cost-effectively.

Limitations of Parallel Interfaces: Most memory channel architectures use parallel interfaces (e.g., 64 bits for DDR3), where each bus line is bit-wise synchronous to a clock signal. Over the past decades, we have increased the memory bandwidth by either increasing the number of channels or the bit rate of channels. Nowadays, we face the following fundamental limitations due to the nature of the parallel interface. First, each channel requires a large number of pins (over 130 I/O pins for DDR3 [7]), but the processor package is often pin-constrained. Second, increasing the bit rate degrades signal integrity super-linearly due to intensified crosstalk between lines and signal attenuation, requiring larger and more power-hungry I/O circuits [8]. Finally, the parallel interface requires bit-wise synchronization, fundamentally limiting the bit rate and the number of lanes due to clock-to-data (C2D) and data-to-data (D2D) skews across lanes.

Advantages and Disadvantages of HSI: Alternatively, HSI (or simply serial interface) transmits serialized bit streams at an extremely high rate. Figure 1 plots energy per bit transfer (pJ/bit) versus bit rate per pin (Gbps/pin) of recent parallel and serial interfaces [9,10,11,12,13,14,15,16,17,18]. The serial interface can provide much higher Gbps/pin at lower pJ/bit than the parallel interface, consuming lower I/O power (Gbps × pJ/bit). The lower I/O power is more desirable, because the I/O power begins to contribute to a notable fraction of total

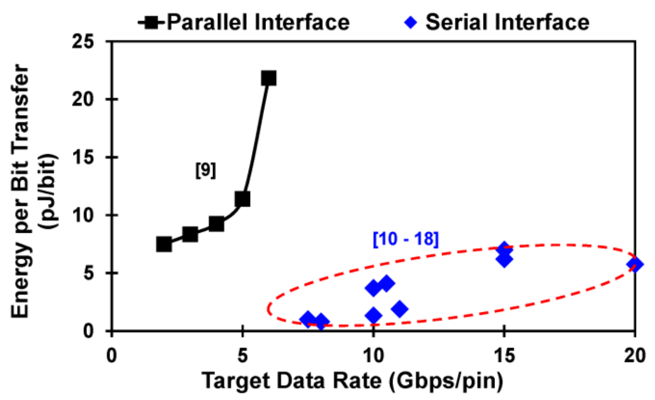

Figure 1: Energy-efficiency (pJ/bit) versus bit rate (Gbps/pin) comparison.

package power due to increasing bandwidth demand [8]. Nonetheless, the serial interface has its own drawbacks; it exhibits longer data transfer latency and consumes higher static energy when it is idle. First, the latency is increased for serializing, packetizing, and encoding data as a price for a high bit rate. Our detailed modeling shows that the one-way latency of HSI can be 10-30*ns* longer than that of DDR3. This can be notable to be used for the memory system for CPUs considering that the CAS latency of DDR3-1600 DRAM is 13.75*ns* [7]. Second, DDR3 consumes very low power when it does not transfer data. In contrast, HSI typically consumes almost the same power whether or not it transfers data. Furthermore, the past HSI exhibits long latency when it exits from a power-down state, forcing HSI to mostly stay in active state. Therefore, the overall energy consumption of HSI can be higher than that of DDR3 when the channel utilization is low. These are the key reasons that the memory system still adopts the parallel interface.

Key Contributions: In this paper, we first provide a deep understanding of recent HSI including its distinct characteristics compared to the past serial interfaces used for FBDIMM [19] and as such [20,21,22] (Section 2), how it works as a memory interface and how it impacts the main memory system design (Section 3).

Second, tackling the bandwidth-wall challenge, we propose hybrid memory channel architecture – Alloy comprised of parallel and serial channels, exploiting that GPU often demands high bandwidth but it can usually tolerate long latency [23] while CPU typically does not demand as high bandwidth as GPU but it often cannot tolerate long latency in SCHPs (Section 4.1). Compared to the baseline dual-parallel memory channel architecture, we replace one of the two parallel channels with multiple serial channels, with each of them providing the same bandwidth as a DDR3-based parallel channel but using fewer pins. Therefore, Alloy can provide lower overall latency than using only serial channels and higher bandwidth than using only parallel channels, under a package pin constraint.

Third, considering the aforementioned characteristics of CPU and GPU, we propose a memory channel partitioning technique that adaptively maps physical pages of latency-sensitive (CPU) and bandwidth-sensitive (GPU) applications to parallel and serial channels, respectively (Section 4.2). Alloy adopting this technique offers 21% and 33% higher performance when SCHP co-runs diverse CPU and GPU applications, respectively.

Finally, we provide a detailed analysis on the peak power and energy consumption of memory interfaces and demonstrate how recent HSI's short wake-up latency, which is enabled by its advanced clocking scheme, can be architecturally exploited to reduce static energy consumption (Section 4.3). This technique reduces the energy consumption of four serial channels to a level similar to two parallel channels while reducing the GPU performance improvement by only 1%.

2. High-speed Serial Interface

HSI transmits the bit stream of serialized data at a very high bit rate. To achieve such a high bit rate, low-voltage differential signaling (LVDS) and equalization techniques are used to overcome inter-symbol interference and signal attenuation in channels [10]. Since HSI also uses embedded clocking, each lane can operate independently. This allows the serial interface to shun C2D and D2D skews across lanes and thus further increases the bit rate. Overall, HSI provides much better pJ/bit scalability than the parallel interface over 10Gbps (Figure 1). As the parallel interface is approaching its fundamental limit, increasing its bit rate over a certain point super-linearly increases pJ/bit [8].

Recent HSI versus Past Serial Interfaces: HSI is distinct from the past interfaces used for FBDIMM [19] and as such [20,21,22]; they employ differential signaling and serialization like HSI, but they still share some fundamental limitations of the parallel interface because they are in fact a narrow-width parallel interface. Consequently, such interfaces suffered from very high (dynamic and static) power consumption and long wake-up latency, inheriting the limitations of both parallel and serial interfaces. In contrast, recent HSI can offer much lower power consumption and far higher bandwidth while notably reducing wake-up latency using advanced clocking. This allows us to apply more aggressive static power management. However, longer latency, which comes as a price for a high bit rate, becomes notable for recent HSI whereas it was not paid much attention for past serial interfaces.

Transmitting and Receiving: Figure 2 depicts the architecture of a transmitter (Tx) and receiver (Rx) pair. Tx and Rx operate in separate symbol clock domains (typically hundreds of MHz). On the Tx side, (i) the payload (either address/command or data) is stored in the asynchronous FIFO buffer after crossing from the (DRAM or processor) internal I/O bus clock domain to the symbol clock domain; (ii) it is packetized with a header and a tail; (iii) the packetized payload is distributed across lanes at byte granularity; (iv) the distributed packet is 8b/10b encoded at each lane; (v) each 10-bit symbol of the encoded packet is serialized; and (vi) the bit stream of each serialized 10-bit symbol, where the clock signal is embedded to avoid C2D

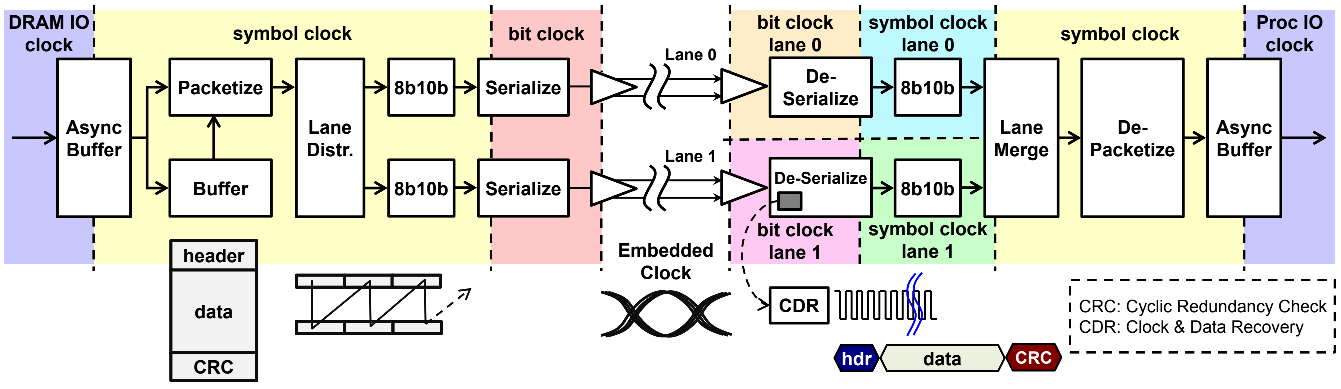

Figure 2: Architecture of an HSI transmitter-receiver pair where different clock domains are separated by dotted lines.

skews, is transmitted through each lane. On the Rx side, step (i) through (iv) are reversed after the clock and data recovery (CDR) unit recovers the clock from the incoming bit stream.

The latency of HSI can be represented as follows:

$$T_{LAT} = T_{TxSER} + T_{RxDES} + T_{LINK} + T_{VarCDC} \quad (1)$$

where T_{TxSER} is symbol clock cycles for step (i) through (vi); T_{RxDES} is symbol clock cycles for reversing step (i) through (vi); T_{LINK} is the latency of transmitting a packet through a link (i.e., packet size / link bandwidth); T_{VarCDC} is variable cycles incurred by Clock-Domain Crossing (CDC) due to uncertain phase difference between different clock domains.

Packetizing and Encoding: HSI encapsulates data with a header and a tail to form a packet. At the end of each packet, 16-bit cyclic redundancy check (CRC) code is augmented to detect possible transmission errors. It is reported that 40Gpbs Rx exhibits a bit error rate (BER) of 10^{-12} (i.e., one bit error may occur every 25s) [12]. The re-transmission of a packet every 25s will have a negligible impact on performance. 8b/10b encoding transforms 8-bit symbols into 10-bit ones for symbol and packet boundary identifications.

Signaling: The parallel interface mostly uses single-ended signaling. However, as the bit rate increases, signals are significantly distorted by crosstalk between lanes and attenuation in channels. Consequently, Tx needs to drive the signals with more power to maintain the signal integrity at Rx. In contrast, the serial interface uses LVDS to reduce crosstalk and power-supply noise, allowing robust transmissions at a much higher rate and lower power consumption.

Clocking: At a relatively low bandwidth target, forward clocking, which sends a separate clock signal along with the data links, requires simpler Tx/Rx circuits and thus consumes lower power than embedded clocking. However, it poses a fundamental limitation on lane speed and scalability due to C2D and D2D skews across lanes for the parallel interface. Thus, HSI uses embedded clocking, where Rx recovers the clock signal from the incoming bit stream. Since the recovered clock is coherent to its own data for each lane, inter-lane skews can be tolerated up to several symbol intervals. Further, Rx can gather bytes from multiple lanes in asynchrony although Tx distributes bytes across multiple lanes in synchrony (Figure 2). This further eliminates the bit rate constraint placed by D2D skews. Therefore, embedded clocking is one of the key reasons for HSI to achieve a much higher bit rate at lower power consumption than the past interface technology such as FBDIMM.

3. HSI-based Memory Channel Architecture

3.1 Architecture

There are various ways to employ HSI between a processor and a memory module, such as using a logic layer stacked to memory layers similar to hybrid memory cube (HMC) [24,25] and an on-board buffer [26] . In this section, we describe one possible channel architecture implementation that can work with the commodity DDR3-based DIMMs. However, the techniques described in the rest of this paper are orthogonal to the channel architecture implementation presented in this section and thus can be applied to other memory modules employing HSI such as HMC.

Interface: To interface an HSI-based memory channel with the standard DIMMs, we place an on-board bridge chip similar to on-board buffer chips that have been widely used to improve the signal integrity for high-end systems such as IBM Power 795 [20]. Our bridge chip is equipped with (i) a buffer relaying commands and data between the memory controller (MC) and the DIMM and (ii) a timing unit regulating the latency varied by CDC (i.e., T_{VarCDC}) such that various DRAM timing constraints are satisfied.

Link: ALNK in Figure 3 is a forward link and consists

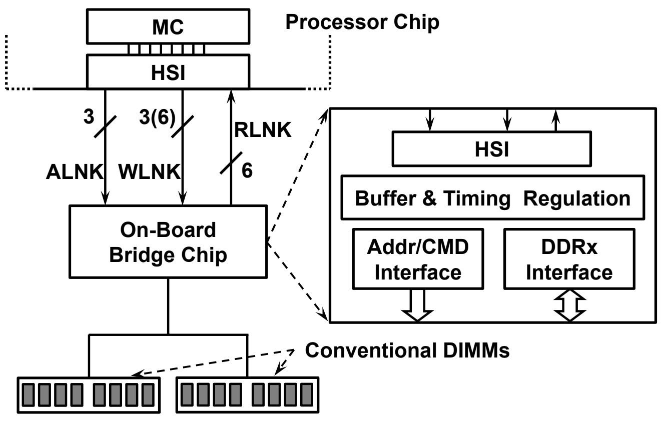

Figure 3: HSI-based memory channel architecture.

Table 1: Specification of serial and parallel interfaces aimed for DDR3-1600; the numbers in the parentheses are for 3-lane WLNK.

	Serial	Parallel
Data Block Size (byte)	64	
ECC size (byte)	8	
Payload (byte)	72	
Packet Size (byte)	76	-
Encoding Overhead	25%	0%
Total size (bit)	760	576
Bit rate per lane (Gbps)	25.3	1.6
# of data lanes	6 (3)	64 + 8=72
Total bit rate	151.8 (75.9)	115.2
Time for a block (ns)	5 (10)	5
# of total pins	30 (24)	131

of 3 lanes for transporting address/command packets. A DDR3-based channel needs 35 bits for its address/command bus: (i) 16-bit address; (ii) 3-bit bank select; (iii) 2-bit row/column strobe; (iv) 1-bit write enable; (v) 2-bit rank select; (vi) 9-bit data mask; and (vii) 2-bit clock enable signals [7]; HSI using embedded clocking does not need to send separate clock signals to DIMMs. Atop these 35 bits, 5 dummy, 32 packetization, and 18 8b/10b-encoding overhead bits constitute an address/command packet (i.e., total 90 bits). For 3-lane ALNK to provide the same bandwidth as a DDR3 address/command bus transporting an address/command over 1 DIMM clock cycle (1.25*ns* for DDR3), the bit rate of each lane should be 24Gbps.

RLNK and WLNK in Figure 3 are *uni-directional* backward and forward links, each of which is comprised of 6 lanes, for transferring read and write data packets, respectively. Although HSI can support a *bi-directional* link, it exhibits prohibitively higher latency than DDR3 for changing the direction of transmissions. Since write requests are much fewer than read requests, we also consider 3-lane WLNK, following the same design philosophy as the FBDIMM where the write bandwidth is a half of the read bandwidth. Atop 64 data bytes, 8 ECC, 4 packetization, and 19 8b/10b-encoding overhead bytes comprise a data packet (i.e., total 95 bytes). We do not consider a smaller packet transporting fewer data bytes (e.g., 16 bytes transferred over a single DIMM clock cycle like FBDIMM [19]) because of high packetization overhead. For 6-lane RLNK and WLNK to provide the same bandwidth as a DDR3 data bus transferring 64-byte data over 4 DIMM clock cycle (4×1.25*ns* = 5*ns* for DDR3), the bit rate of each lane should be 25.3Gbps.

Table 2: HSI latency.

	Latency (*ns*)	Latency (cycles)
ALNK	24.5 ~ 29.0	20 ~ 24
RLNK (and WLNK w/ 6 lanes)	28.3 ~ 32.8	23 ~ 27
WLNK w/ 3 lanes	33.3 ~ 37.8	27 ~ 31

Table 1 summarizes the specification of HSI-based serial and DDR3-based parallel interfaces. The required bit rate of each lane for ALNK, RLNK, and WLNK lies within the practical range depicted in Figure 1. For simplicity, we decide that each lane of ALNK provides the same bit rate as each lane of RLNK (i.e., 25.3 Gbps/lane).

Latency: We conservatively assume T_{TxSER} and T_{RxDES} in Eq. (1) are 6 and 9 symbol clock cycles where the symbol clock frequency is 625MHz based on a proprietary HSI implementation. T_{LINK} is 5*ns* (10*ns* for a 3-lane WLNK) and 1.2*ns* to transport a read (write) data packet and an address/command packet, respectively, as summarized in Table 1. Finally, the upper limit of variable latency is two symbol clock cycles plus one internal I/O bus clock cycle. The latency values for transferring an address/command packet and a read or write data packet over the HSI-based channel are summarized in Table 2.

Total Bandwidth: A DDR3-based channel needs total 131 I/O pins (64-bit data, 8-bit ECC, 18-bit differential data strobe, 4-bit differential clock, 2-bit on-die termination, and 35-bit address/command signals). In contrast, an HSI-based channel needs only 12 (15) lanes or 24 (30) pins with 3-lane (6-lane) WLNK. Therefore, we can replace one DDR3-based channel with up to four HSI-based ones that provide 4× higher bandwidth using 6-lane WLNK (or 4× and 2× higher read and write bandwidth with 3-lane WLNK) while using 8% (27%) fewer pins than one DDR3-based parallel channel. Considering that 6-lane WLNK can provide marginally higher GPU performance than 3-lane WLNK but uses 25% more pins per channel, we use 3-lane WLNK for performance and power evaluations in this paper.

3.2 Timing Protocol

In this section, we describe how an HSI-based channel services memory requests while preserving the design and timing protocol of the standard DIMM.

Read: Figure 4 illustrates an example of servicing two consecutive read requests to the same row. First, MC issues an activate (ACT) command. After certain latency, the ACT

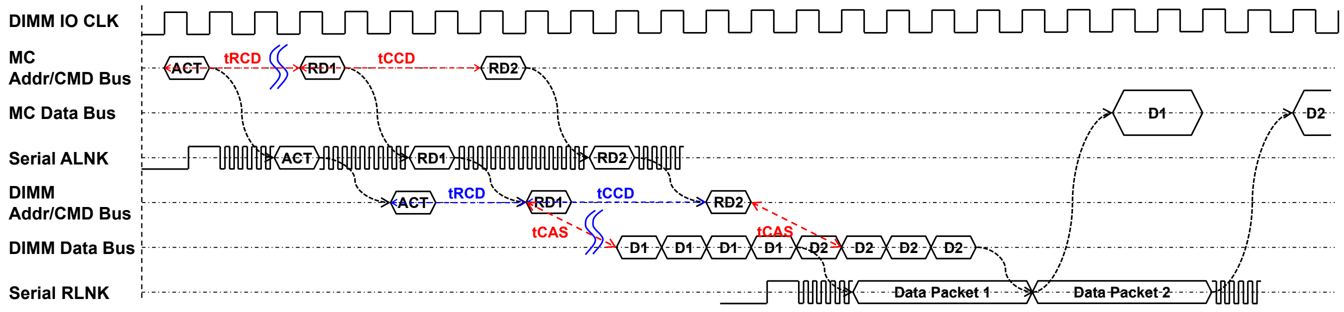

Figure 4: A read transaction of an HSI-based memory channel.

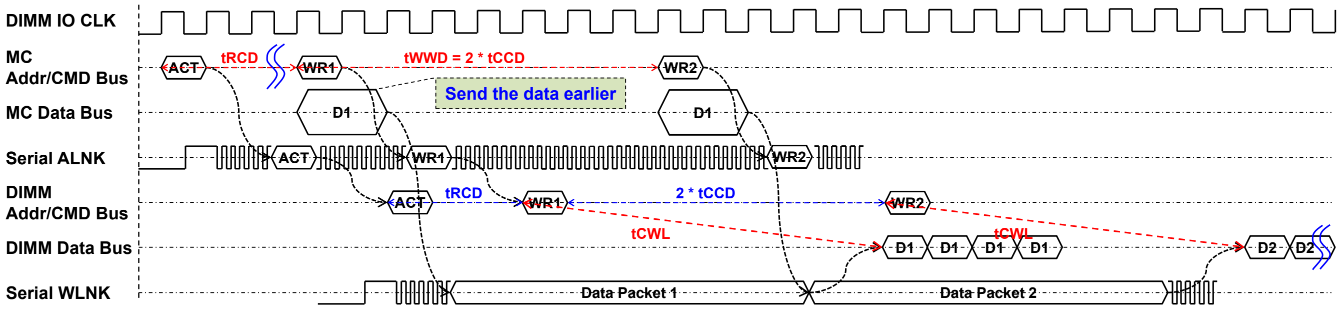

Figure 5: A write transaction of an HSI-based memory channel.

command packet is transmitted through ALNK and received by the bridge chip, which places the ACT command retrieved from the packet on the DIMM address/command bus. Second, following the DDR3 timing protocol, MC issues a column read (RD) command after tRCD. However, due to the variable latency incurred by CDC (T_{VarCDC}), the RD command packet can arrive at the bridge chip earlier than expected, violating the tRCD constraint. Thus, the bridge chip needs to buffer the RD command until it satisfies the tRCD constraint. After tCAS, the DIMM transmits the requested data to the bridge chip over 4 consecutive DIMM clock cycles. Then the bridge chip transmits the read data packet to MC through RLNK, which also takes 4 DIMM clock cycles. For the next request to the same row, MC issues another RD command after tCCD and the bridge chip relays the RD command such that it satisfies the tCCD constraint on the DIMM side. The key point is that neither MC nor DIMM needs to be modified for servicing read requests.

Write: To service a write request, MC first issues a column write command (WR) and then it places the write data on the DIMM data bus after tCWL (i.e., 11 DIMM cycles for DDR3), following the DDR3 timing protocol. However, the latency of transporting a write data packet through 6-lane (3-lane) WLNK is up to 7 (11) DIMM clock cycles longer than that of transferring the WR command packet through ALNK (Table 2). Thus, the bridge chip cannot place the write data on the DIMM data bus on time. One solution for this problem is buffering the WR command at the bridge chip until the write data arrives at the bridge chip. However, this requires us to modify the MC's timing control for all the subsequent commands due to the delay. Alternatively, MC can simultaneously send both the write data and WR command packets to the bridge chip, overlapping their latency difference with tCWL (Figure 5). This is achieved by re-programming the MC's timing parameter corresponding to tCWL with 0. Using this approach, the bridge chip can place the write data on the DIMM data bus on time. More importantly, MC completes the write request on time with respect to the WR command and thus other write-related timing parameters such as tWTP and tWTR are unchanged. For the next write request, however, MC has to wait for 2×tCCD if the HSI-based channel uses 3-lane WLNK that provides only the half bandwidth of 6-lane RLNK (or the DDR3 data bus). This requires us to introduce a new timing parameter–tWWD (=2×tCCD) to MC.

4. Hybrid Memory Channel Architecture

4.1 Architecture

In Section 3, we showed that one parallel channel can be replaced with four serial channels (4× higher bandwidth) under a pin constraint. Such high bandwidth clearly benefits the bandwidth-consuming GPU. However, we also showed that the serial channel has significantly longer latency than the parallel one due to the overhead (Section 3.1). While GPU can usually tolerate the increased latency, CPU typically cannot. This suggests that SCHP comprised of both CPU and GPU cannot rely on either parallel or serial channels.

Exploiting both parallel and serial channels, we propose hybrid memory channel architecture–Alloy comprised of parallel and serial channels (Figure 6). Alloy can provide lower latency than using only serial channels and higher bandwidth than using only parallel channels under a package pin constraint. Assume that the baseline is comprised of two parallel channels denoted by *2p-ch*. Using the same or fewer pins than *2p-ch*, Alloy can be architected to provide one parallel and four serial channels (*1p-4s-ch*). Alternatively, we may consider replacing ALNK in each serial channel with the parallel address/command bus to nearly halve the latency added to the serial channel (the sum of ALNK and RLNK latency values). Since this increases the number of pins for one serial channel from 24 to 59, we can replace one parallel channel with two of such parallel-serial channels

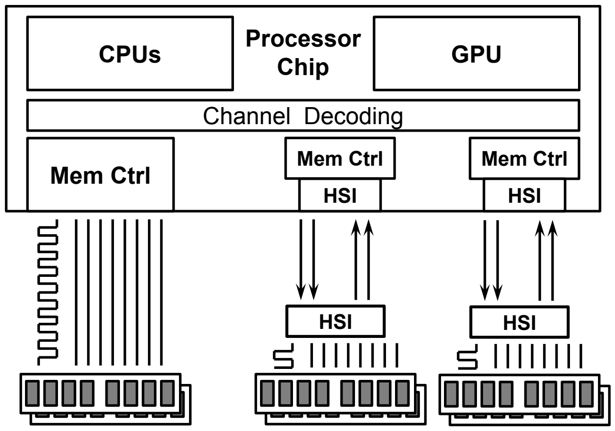

Figure 6: Hybrid memory channel architecture.

(*1p-2ps-ch*). This parallel-serial channel still provides 2× higher bandwidth than one parallel channel with 10% fewer pins.

4.2 Adaptive Memory Channel Partitioning

The memory management subsystem of operating system (OS) manages physical (memory) pages. When a program accesses a virtual (memory) page that is not yet mapped to a certain physical page, a page fault is raised. Then the OS kernel is invoked to allocate a physical page from a free list for the program using page allocation primitives. This allocation process can be enhanced by exploiting hardware information to smartly pick the page (e.g., located in a specific DRAM bank) from the free list. For example, the page coloring technique is extensively used to improve the performance of cache and memory systems [27]. Especially in the multi-core era, it is also used for various implementations partitioning last-level caches [28], DRAM banks [29] and channels [30] to reduce the inter-application interference.

Specifically in our context, leveraging a standard physical page mapping technique, OS can obliviously allocate physical pages of latency-sensitive (CPU) applications (e.g., App-0 in Figure 7) to a serial channel (Figure 7(a)). This may notably degrade the performance of CPU running such applications due to longer latency of servicing memory requests sent to the serial channels. To minimize such performance degradation, we propose an adaptive memory channel partitioning technique optimized for Alloy. This technique maps the physical pages of latency-sensitive (CPU) and bandwidth-consuming (GPU) applications (e.g., App-1 in Figure 7) *preferably* to low-latency parallel and high-bandwidth serial channels, respectively (Figure 7(b)).

Note that GPU applications also have their sequential portion running on CPU, and hence page migration might be beneficial when execution switches between CPU and GPU. However, typically, CPU initializes/prepares the data for parallel kernels running on GPU and retrieves the results afterwards. Since the GPU computation typically dominates the total execution time of real-world GPU applications processing realistic input data sets, we do not consider the overhead of possible page migration in this paper, from the perspective of performance evaluation.

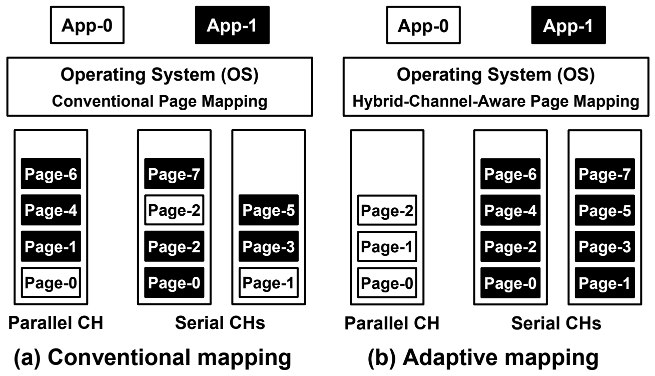

Figure 7: Memory channel partitioning technique for hybrid memory channel architecture.

Using our technique, Alloy can service memory requests from CPU or GPU with a desirable channel (i.e., low-latency parallel or high-bandwidth serial channels), improving the performance of both CPU and GPU simultaneously compared to our 2p-ch baseline. Note that simply applying the application-aware channel partitioning technique for 2p-ch can degrade GPU throughput, because the channel partitioning constrains the memory bandwidth that would be available for GPU applications.

4.3 Static Power Management

The power and energy consumption of I/O interfaces is expected to significantly increase due to aggressive growth in memory bandwidth demand [8]. Thus, another key motivation to adopt HSI is higher energy efficiency (i.e., lower pJ/bit) than DDR*x* (Figure 1); HSI transferring data (or active HSI) typically consumes much lower power than active DDR*x*. However, idle HSI also consumes as high power as active HSI because it needs to maintain the bias for LVDS links and keeps transmitting/receiving dummy symbols for CDR. In contrast, idle DDR3 consumes much less power than active DDR3. Although HSI can reduce its idle (or static) power consumption by entering a power-down state, it typically required long exit latency from a power-down state, forcing it to mostly stay in active state. Consequently, HSI may consume higher average power (energy) than DDR3 under low utilization. This has been one of the key concerns of HSI compared to DDR3.

Recent studies on HSI demonstrated low-power states that can considerably reduce static power consumption (e.g., [31]) and wake-up latency (e.g., [32,33]). The three typical HSI power states are: (i) the "burst" state represents an active state that HSI is transmitting/receiving or ready to transmit/receive a packet immediately; (ii) the "standby" state still maintains the bias for LVDS while turning off Tx/Rx and putting PLL/DLL in idle state; (iii) the "hibernate" state turns off the bias to consume lower static power consumption but incurs much longer wake-up latency than the standby state.

Leveraging (ii) and (iii), we propose the following power management policies that decide when and which low-power state HSI enters at runtime. When MC has no pending request, it puts ALNK into standby state after waiting for a certain time denoted by $T_{standby}$. Similarly, MC puts RLNK into standby state immediately when it has no pending read request, since the wakeup latency can be completely hidden by tCAS for RLNK. Further, MC can put WLNK into a more aggressive power-down state; it immediately puts WLNK into hibernate state because of two

Table 3. Wakeup latency and power consumption (per lane) for different power states.

Power State	Wakeup Latency (ns)	Power / lane (mW)
Burst	--	63
Standby	10	13
Hibernate	100	0.6

Table 4. System configurations.

# of CPU cores	4	# of GPU SMs	12	Channels/Ranks/Banks	2 / 2 / 8
CPU Freq	4.0 GHz	GPU Freq	800 MHz	DRAM I/O Freq	800 MHz
CPU core width	4	GPU # of registers per SM	16384	Data bus width	64 bits / channel
CPU IL1 and DL1	32KB-private / 2-way	GPU # of threads per SM	1024	Row buffer size	1 KB / bank / DRAM chip
CPU L2 per core	4MB-shared / 16-way	GPU # of CTAs per SM	8	tCAS / tRCD / tRP	13.75 / 13.75 / 13.75 ns
CPU store buffer	16 per core	on-chip memory per SM	32KB	MC request buffer size	32

reasons. First, write requests are fewer than read requests. Second, MC typically buffers write requests and write them to the memory when the write buffer occupancy is higher than a certain threshold value denoted by Th_{drain}. Hence, WLNK is expected to experience a long idle period between two events draining the write buffer.

When a new read request arrives, MC wakes up ALNK and RLNK at the same time. This may incur some performance penalty. However, it is GPU that mostly accesses the serial channels under our adaptive memory channel partitioning technique while GPU can mostly tolerate such latency increase. As soon as the write buffer occupancy is equal to or higher than a wake-up threshold value that we introduce and is denoted by $Th_{wake\text{-}up}$ ($< Th_{drain}$), MC starts to wake up WLNK and ALNK in advance such that both WLNK and ALNK are ready to transmit address/command and data when the write requests begin to be issued.

5. Evaluation Methodology

Simulation Framework: We use a cycle-level gem5+GPGPU-Sim simulator [34,35,36] for our evaluations. We augment a cycle-driven enhanced MC model to replace the event-driven model originally built in gem5, verify the timing parameters and protocols with the DDR3-1600 [7], and develop a separate protocol for HSI based on the detailed architecture presented in Section 3.2. We configure the simulator similar to AMD A10-6800K APU [37] (Table 4). The CPU cores share the 4MB L2 cache while GPU does not share the L2 with CPU; the number of stream multiprocessors (SMs) in GPGPU-Sim is configured to provide a similar maximum GFLOPS to the GPU in AMD APU [38].

Benchmark: We evaluate 21 SPEC CPU2006 benchmarks. The name and memory intensity measured by memory requests per kilo instructions (MPKI) are summarized in Table 5. We fast-forward the first one billion instructions in functional mode and then simulate one billion instructions in cycle-level mode. We use 13 Rodinia GPGPU benchmarks [39] for GPU. Table 6 summarizes the abbreviation, memory intensity, row-buffer locality (RBL), and ratio of effective bandwidth to the maximum bandwidth. Due to the lack of graphics application supported by the simulators, we cannot evaluate graphics applications. Nonetheless, Table 6 shows the evaluated benchmarks exhibit a wide range of memory intensities and row-buffer localities. Therefore, we expect that SCHPs running graphics applications may benefit from Alloy as well.

Table 5. CPU benchmarks.

Low		Medium		High	
Name	MPKI	Name	MPKI	Name	MPKI
povray	0.005	astar	0.785	leslie3d	11.3
GemsFDTD	0.006	gobmk	0.841	libquantum	14.9
tonto	0.034	calculix	1.17	bwaves	15.3
h264ref	0.106	sphinx3	1.21	milc	21.8
sjeng	0.461	namd	1.29	lbm	55.7
gromacs	0.599	bzip2	1.89	mcf	73.4
		omnetpp	2.50		
		soplex	2.50		

6. Evaluation

6.1 Performance

We first show that simply replacing the parallel interfaces with the serial ones improves GPU performance significantly but degrades CPU performance unacceptably. Then we show Alloy minimizes this performance degradation when CPUs are running alone, and finally demonstrate substantial performance improvement when both CPU and GPU applications are running in the system.

GPU-only Scenario: Figure 8 plots the normalized IPC and effective bandwidth usage of GPU applications using *2p-ch*, *4ps-ch* and *8s-ch*. The two channels of *2p-ch* can be replaced with four parallel-serial or eight serial channels, which are denoted by *4ps-ch* and *8s-ch*, respectively. *4ps-ch* can offer 2× higher bandwidth but ~30*ns* longer latency than *2p-ch*; it can provide 2× lower bandwidth but ~30*ns* shorter latency than *8s-ch*. As expected, most GPU applications demand high bandwidth while exhibiting some tolerance to increased latency; their performance is more sensitive to bandwidth than latency. On average, *4ps-ch* and *8s-ch* can offer 41% and 80% higher performance than *2p-ch*, respectively. Considering the effective bandwidth usage, we

Table 6. GPU benchmarks.

Benchmark	Abbrev.	MPKC	RBL	Eff / Peak BW
Needleman-Wunsch	NW	114	0.21	23%
Streamcluster	SC	152	0.69	30%
HotSpot	HSP	297	0.81	59%
Particle Filter	PTFT	313	0.98	63%
LU Decomposition	LUD	334	0.93	67%
Back Propagation	BKP	354	0.94	71%
SRAD	SRAD	397	0.55	79%
Breadth-First Search	BFS	402	0.54	80%
LavaMD	LavaMD	433	0.96	87%
PathFinder	PFD	436	0.97	87%
CFD Solver	CFD	448	0.94	90%
K-Means	Kmeans	464	0.90	93%
Heart Wall	HTW	468	0.92	94%

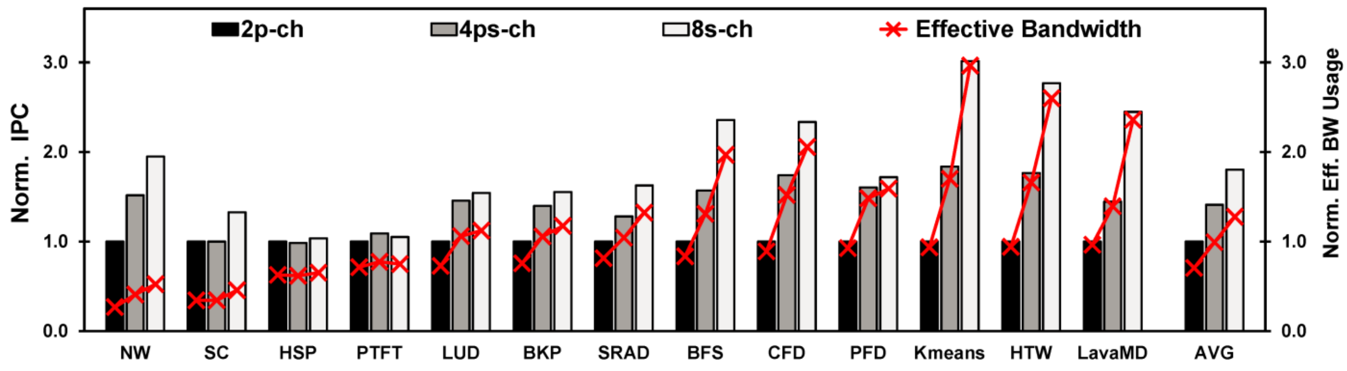

Figure 8: IPC and effective bandwidth usage of GPU applications running on the SCHP using *2p-ch*, *4ps-ch*, and *8s-ch*. The IPC and effective bandwidth are *normalized* to the IPC and maximum bandwidth of the SCHP using *2p-ch*.

see that *2p-ch* uses 71% of the maximum bandwidth. In contrast, *4ps-ch* and *8s-ch* consume 29% and 57% higher effective bandwidth than *2p-ch*, respectively; *8s-ch* even uses 28% higher bandwidth than the maximum bandwidth of *2p-ch*.

CPU-only Scenario: Using *4ps-ch* and *8s-ch* significantly improves the performance of GPU running bandwidth-hungry applications, but it may not be acceptable for the performance of CPU running latency-sensitive applications. Figure 9 plots the normalized IPC of 21 SPEC CPU2006 benchmarks using *2p-ch*, *4ps-ch, 8s-ch*, and *1p-4s-ch* assisted by our adaptive channel partitioning technique (*1p-4s-ch-part*). On average, *4ps-ch* and *8s-ch* offer 6.6% and 12.5% lower performance than *2p-ch*, respectively. Especially for memory-intensive applications, the performance degrades significantly (e.g., 23% and 37% for *mcf*). Figure 10 plots the breakdown of the total memory latency by MC queuing plus DRAM service and HSI latencies. *4ps-ch* and *8s-ch* actually decrease the sum of MC queuing plus DRAM service latencies by 13% and 19%, respectively, because they provide more channels than *2p-ch*. However, the reduction of these latencies is outweighed by the increased latency introduced by HSI. In contrast, *1p-4s-ch-part* exhibits only 1.3% (at most 4.9% for *mcf*) performance degradation on average. The performance degradation results from the fact that the bandwidth available to CPU applications is only a half of the maximum bandwidth of *2p-ch*, due to channel partitioning. However, since CPU applications are mostly more sensitive to latency than bandwidth, *1p-4s-ch-part* performs much better than *4ps-ch* and *8s-ch* while only slightly worse than *2p-ch*.

CPU-GPU Scenario: We choose four GPU benchmarks

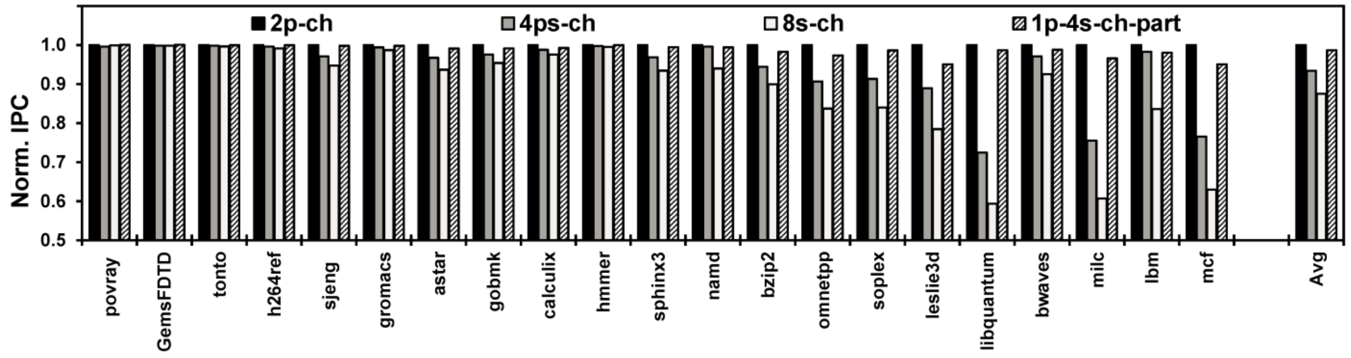

Figure 9: IPC of CPU applications running on the SCHP using *2p-ch*, *4ps-ch*, *8s-ch*, and *1p-4s-ch-part*.

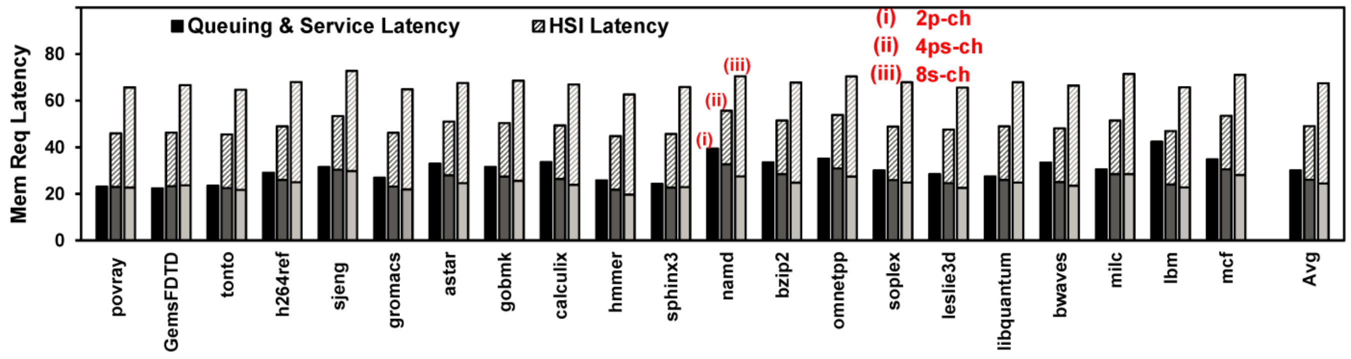

Figure 10: Latency breakdown of *2p-ch*, *4ps-ch*, and *8s-ch* by MC queuing plus DRAM service and HSI latencies.

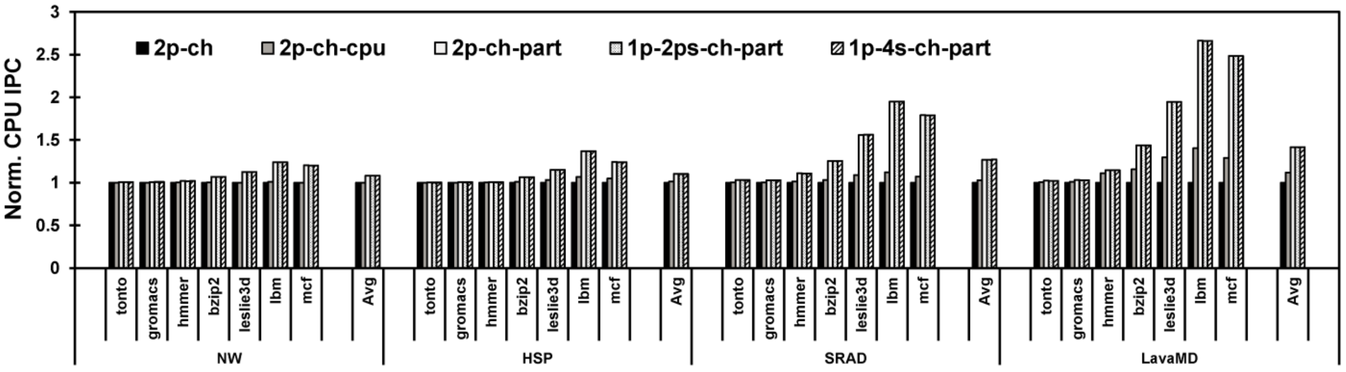

Figure 11: IPC of CPU applications co-running with GPU applications using *2p-ch*, *2p-ch-cpu, 2p-ch-part*, *1p-2ps-ch-part*, and *1p-4s-ch-part*. The average IPC for each GPU application is the geometric mean across all 21 SPEC CPU2006 benchmarks.

with different memory intensities and row-buffer localities, and co-run each with all 21 SPEC CPU2006 benchmarks. Figure 11 plots the normalized IPC of CPU applications co-running with GPU applications; the baseline uses *2p-ch* without enabling a memory channel partitioning technique. We also evaluate (i) two *2p-ch* configurations: always prioritizing CPU requests (*2p-ch-cpu*) and assisted by the adaptive memory channel partitioning technique (*2p-ch-part*) and (ii) two Alloy configurations: *1p-2ps-ch-part* and *1p-4s-ch-part*.

First, *2p-ch-cpu* does not significantly improve CPU performance compared to *2p-ch*. This is because ideally a CPU request is expected to be issued to a DRAM bank as soon as it arrives at MC due to higher priority than GPU requests, but a GPU request to the same DRAM bank may be already scheduled even before the high-priority CPU request arrives. Consequently, this CPU request has to wait for the prior GPU request to be serviced. Such a situation is likely to frequently happen as GPU applications generate far more memory requests per unit time than CPU applications. In contrast, *1p-2ps-ch-part* or *1p-4s-ch-part* mostly prevents GPU requests from interfering CPU requests by mapping their physical memory pages to different channels, improving the CPU performance by 8%, 10%, 27%, and 42% for CPU applications co-running with NW, HSP, SRAD and LavaMD, respectively, compared to baseline *2p-ch*.

Second, *2p-ch-part* achieves the same performance improvement as *1p-2ps-ch-part* or *1p-4s-ch-part*. However, *2p-ch-part* may not be a practical option considering its negative impact on GPU performance. Figure 12 plots the normalized IPC of GPU applications with *2p-ch*, *2p-ch-cpu*, *2p-ch-part*, *1p-2ps-ch-part*, and *1p-4s-ch-part*. We do not separately plot the cases when a GPU application is running with different CPU applications as in Figure 11, since CPU applications exhibit very limited impact on GPU performance. Although *2p-ch-part* substantially improves CPU performance, it significantly degrades GPU performance since GPU applications are now partitioned to only one of two parallel channels. On average, *2p-ch-part* provides 39% lower GPU performance than *2p-ch*. In contrast, *1p-2ps-ch-part* or *1p-4s-ch-part* gives the same CPU performance as *2p-ch-part* while offering notably higher GPU performance than *2p-ch-part*. Specifically, *1p-2ps-ch-part* provides 2× higher bandwidth to GPU applications and hence delivers 30% higher performance than *2p-ch-part*. However, compared to *2p-ch*, *1p-2ps-ch-part* degrades GPU performance by 9% since its serial channels have longer latency while only providing the same bandwidth to GPU applications as *2p-ch*. Therefore, a better architectural choice is using *1p-4s-ch-part* that can provide 2× higher bandwidth for GPU applications than *2p-ch*, yielding 33% (72%) higher GPU performance than *2p-ch* (*2p-ch-part*) on average.

In Figure 11 and Figure 12, both CPU and GPU applications are sorted by memory intensity where the rightmost one is most memory-intensive. Figure 11

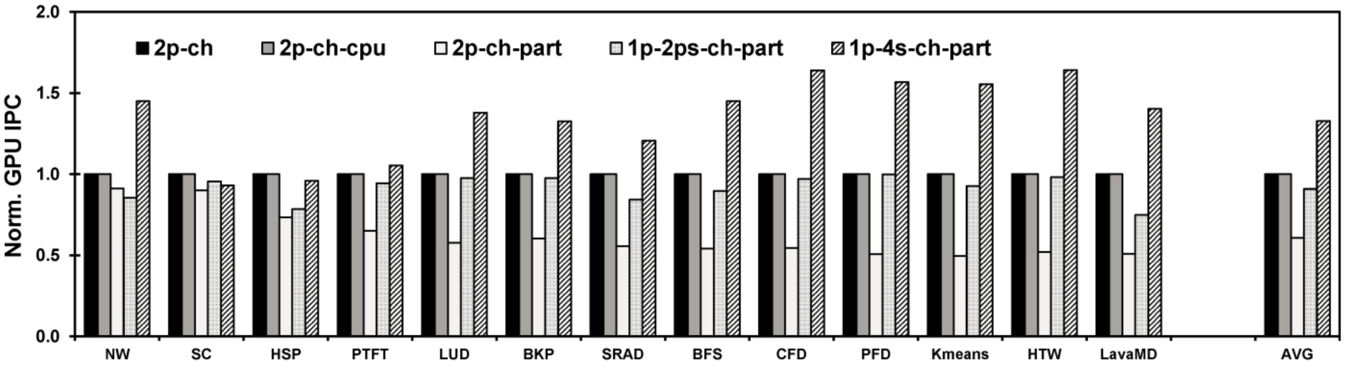

Figure 12: IPC of GPU applications co-running with CPU applications using *2p-ch*, *2p-ch-cpu, 2p-ch-part*, *1p-2ps-ch-part*, and *1p-4s-ch-part*.

demonstrates that the adaptive memory channel partitioning technique generally allows CPU applications with higher memory intensity (applications to the right in each group) to obtain higher improvement when CPU applications co-run with memory-intensive GPU applications (groups to the right). This is because more memory-intensive GPU applications more negatively impacts on CPU applications without using the adaptive memory channel partitioning technique.

Similarly, Figure 12 shows that *1p-4s-ch-part* generally provides higher performance improvement than *2p-ch* for more memory-intensive applications. Specifically, *1p-4s-ch-part* achieves 54% higher performance than *2p-ch* for the six most memory-intensive applications on average than *2p-ch*. Note that *1p-4s-ch-part* gives slightly lower performance for SC and HSP because these two GPU applications are not very memory intensive. Although NW is not memory intensive similar to SC and HSP, *1p-4s-ch-part* provides much higher performance than *2p-ch* for NW. This is because each thread block of *NW* loads all its data into the GPU's on-chip shared memory at the beginning of the kernel execution and then the kernel mainly operates on the data in the shared memory. As a result, the off-chip memory accesses happen in a bursty way such that NW is indeed memory-bound although its average memory intensity is not high [39].

Other Workload Mix Scenarios: In above subsections, we show the performance evaluation results for just single CPU/GPU application and one CPU + one GPU application mix. There are many more scenarios mixing applications with different memory characteristics and/or different number of running applications. Generally, under high memory system utilization with more applications, the increased total bandwidth and concurrency provided by Alloy with more channels are likely to obtain higher improvement in system performance. We have focused on the critical performance of each CPU and GPU applications due to page limit, but also present the following two other workload mix scenarios briefly. For a mix of four CPU applications, on average *1p-4s-ch-part* is only 0.8% and 3% worse than the baseline *2p-ch* and *2p-ch-part*, respectively. And for a mix of four CPU plus one GPU applications, *1p-4s-ch-part* improves performance of CPU and GPU by 27% and 33% than baseline *2p-ch*, respectively; similar to the above CPU-GPU scenario case, *2p-ch-part* shows the same improvement as *1p-4s-ch-part* for CPU performance, but it delivers 39% lower GPU performance compared to *2p-ch*.

6.2 Peak I/O Power and Energy

In this section, we first compare the peak power consumption of HSI with those of DDR*x*. Second we evaluate energy consumption of *1p-4s-ch-part* assisted by our power management technique.

Peak Power: The power consumption of an interface can be calculated as the product of bit rate (Gb/s) and energy consumption per bit transfer (pJ/bit). Consider DDR3-1600 as an example. A data bus, which consists of 64 data lanes and 16 data strobe lanes, operates at 1.6Gb/s with 7.5pJ/bit [9], consuming (1.6×80)×7.5 = 960mW. An address/command bus, which is comprised of 35 lanes plus 4 clock lanes, operates at 0.8Gb/s with 5.5pJ/bit [9], consuming 172mW. In comparison, HSI operating at 22Gb/s with 1.9pJ/bit is demonstrated in 40nm technology [10]. Since we use a slightly higher bit rate (25.3Gb/s) in this paper, we take a higher pJ/bit (2.5pJ/bit) for our analysis. With 12 lanes, the total power consumption of a serial channel is (25.3×12)×2.5 = 759mW (i.e., 33% lower than DDR3-1600).

Table 7. Comparison of peak power consumption between DDR*x*- and HSI-based memory channels.

	bit rate	# of lanes	pJ/bit	(Peak) Power
1× DDR3-1600	1.6 Gb/s	80	7.5	1.1 W
	0.8 Gb/s	39	5.5	
1× HSI-based	25.3 Gb/s	12	2.5	0.76 W
1x DDRx-6400	6.4 Gb/s	80	22	**12 W**
	3.2 Gb/s	39	8.5	
4x HSI-based	25.3 Gb/s	4×12	2.5	**3.0 W**

The above comparison, however, may not be fair because serial channels provide much higher memory bandwidth under a package pin constraint; while pJ/bit largely depends on the target data rate (Figure 1). Hence, it may be more appropriate to compare the power consumptions of four serial channels with one DDR*x*-based channel supporting a higher bit rate (i.e., 6.4Gb/s) to provide the same bandwidth as the four serial channels. It is reported that DDR*x*-6400 will operate at 22pJ/bit and 8.5pJ/bit for the 6.4Gb/s data bus and the 3.2Gb/s address/command bus, respectively [9]. The comparison of power consumption between DDR*x* and serial channels is summarized in Table 7. We can see that an HSI-based channel exhibits much better scalability than a DDR*x*-based parallel channel for increasing the bandwidth under a package pin constraint.

Energy: We evaluate our power management policies applied to *1p-4s-ch-part*. We set $T_{standby}$, Th_{drain} and $Th_{wake\text{-}up}$ as 10ns, 80% and 75%, respectively; we can dynamically adjust these values across applications and/or intervals of each application, but we leave such explorations as future work. Figure 13 presents the percentage of time that each link can be put into low-power states when various GPU applications are running. For most benchmarks, WLNK exhibits the most opportunity to stay in low-power states, since MC buffers write requests and drains the write buffer in a bursty way as described in Section 4.3. On average, WLNK can stay 60% of time in hibernate state, while ALNK and RLNK remain 25% and 46% of the runtime in standby state, respectively. We also measure the performance penalty when applying these power management policies. For all 14 GPU benchmarks, *1p-4s-ch-part* supported by our power management technique (*1p-4s-ch-part-pm*) shows at most 2.8% performance degradation (only 1.1% on average), still providing 32% higher performance than *2p-ch*.

Figure 14 plots the energy consumption of (i) *2p-ch*; (ii) four serial channels of *1p-4s-ch-part* that always stays in

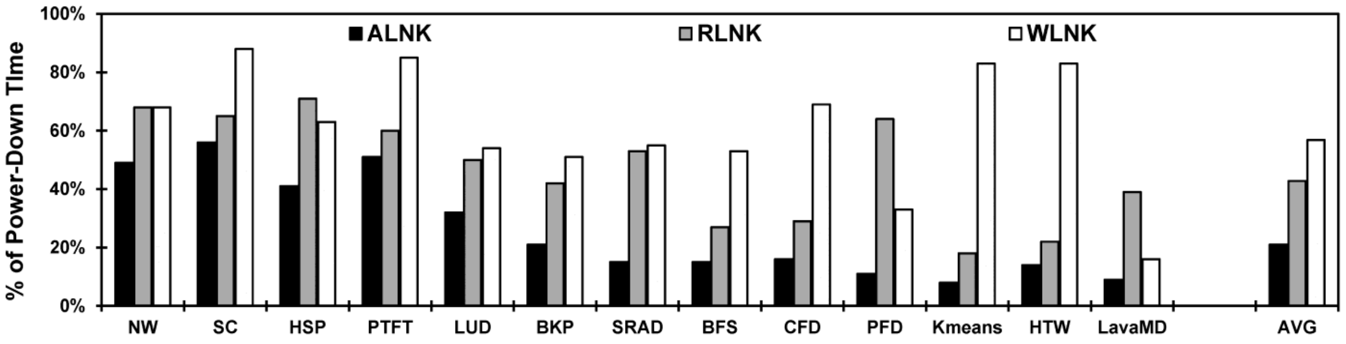

Figure 13. Percentage of time each link can be put into low-power states. ALNK and RLNK in standby state; WLNK in hibernate state.

burst state; and (iii) four serial channels of *1p-4s-ch-part-pm*. It also plots the breakdown of total energy consumption of these channels by the energy consumption for transferring useful data and the wasted static energy consumption during idle time. We can see that (i) *1p-4s-ch-part-pm* consumes 57% lower energy than *2p-ch* for transmitting useful data; (ii) our power management technique reduces the overall energy consumption of HSI-based channels by 39% with only 1% performance degradation; (iii) HSI-based channels consume 4% lower interface energy than *2p-ch* even including the static energy consumption incurred during idle time.

Finally, similar to earlier discussion on the peak power consumption of the memory channels, the above comparison may not be completely fair since the four serial channels in *1p-4s-ch-pm* provide 2× higher bandwidth than *2p-ch*. Furthermore, *1p-4s-ch-part-pm* reduces execution time by 32%, considerably reducing SCHP energy consumption. Thus, if we consider the total energy consumption of both processor and the channels together, *1p-4s-ch-part-pm* can consume notably lower total energy than *2p-ch*. Besides, if we consider a parallel interface with higher bandwidth that can provide the same bandwidth as four serial channels in *1p-4s-ch* under a package pin constraint, the energy consumption of the parallel channels can be notably higher than the four serial channels according to Table 7.

7. Related Work

FBDIMM [19] uses a narrow and fast parallel bus between MC and the DIMMs, where the multi-drop parallel interface is also replaced with a point-to-point interface between MC and an intermediate buffer. The key motivation of FBDIMM is to solve the capacity issue since the signal integrity degradation on high-speed bus limits the number of DIMMs per channel. As explained in Section 2, the serial interface used in FBDIMM is different from HSI. FBDIMM also uses LVDS and forward clocking which are the sources of high static power consumption and long exit latency from a power-down state [33], respectively. In contrast, the unique features of HSI, such as embedded clocking and asynchronous lanes, significantly increase the bit rate and lower the exit latency from a power-down state [33]. The high bit rate also indicates longer and variable latency that need to be properly handled from architecture design perspective.

Decoupled DIMM [40] decouples the bandwidth match between a channel and a single rank of DRAM devices. It allows the channel to operate at a much higher data rate than that of a rank, while the combined bandwidth of multiple ranks (e.g. 2) still provides the same bandwidth as the memory bus. A similar concept is exploited by buffer output on module (BOOM) [41] to enable the use of low-power mobile memory devices for high-end servers. It aggregates multiple ranks of low-power DRAM devices to provide high bandwidth. These two studies are orthogonal to our work. We achieve high bandwidth by providing multiple channels each with fewer pins. For each serial channel, we can use these techniques to either relax the DRAM device speed requirement for higher power efficiency, or further increase the bandwidth to have multiple ranks operate in aggregation. A study on buffer-on-board (BOB) memory system [26] generalizes the technique of placing intermediate buffer and

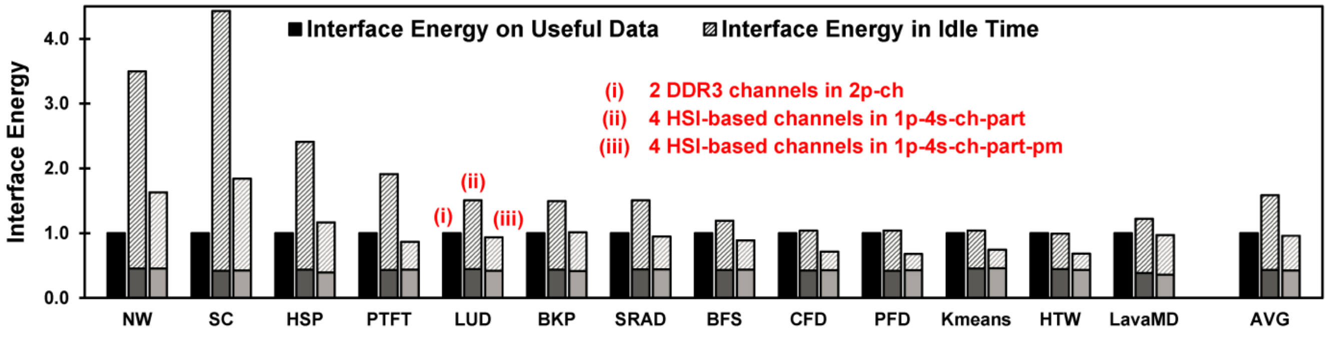

Figure 14. Energy consumption of (i) the two parallel channels of *2p-ch* and (ii) the four serial channels of *1p-4s-ch-part*.

logic between CPU and DRAM and using a narrow and fast parallel bus between CPU and the intermediate buffer.

HMC stacks multiple DRAM dies atop a logic die implementing DRAM control, data routing, error correction, and high-speed off-chip interconnects [24,25]. Between the processor and the logic layer, HMC employs a serial interface (e.g., 15Gbps/lane). With higher bandwidth and improved concurrency, the system latency is expected to decrease under high memory system utilization. However, the *critical latency* is inevitably increased by its serial interface. Therefore, the key concept of Alloy, hybrid-interface channel architecture presented in this paper is orthogonal to HMC technology while HMC can constitute serial channels.

MLP aware heterogeneous memory system [42] proposed to have three different memory modules each optimized for separate key metrics: latency, bandwidth, and power. Then by adaptively allocating physical pages from an appropriate memory module to an application based on its memory characteristics, it can increase system performance and reduce memory power consumption. We also provide heterogeneity to the memory system, but we target fundamentally different problems highlighting bandwidth scalability challenges.

8. Conclusion

In this paper, we proposed Alloy, hybrid memory channel architecture using both parallel and serial interfaces for a memory interface to overcome the poor scalability of the conventional parallel interface. First, we introduced the internal architecture of a serial link, and described the key HSI features that enable a high bit rate at low pJ/bit. Second, we provided the detailed channel architecture based on HSI. Our architecture requires no change to the conventional DIMM architecture and minimal changes to MC timing protocol. Third, to exploit the high bit rate and overcome the long latency of HSI, we proposed *Alloy* consisting of one parallel and multiple serial channels under a pin constraint. *Alloy*, assisted by our hybrid-channel-aware channel partitioning scheme, can simultaneously improve the performance of both CPU and GPU applications co-running on an SCHP. Finally, we performed a detailed analysis on memory interface power consumption. The comparison against DDRx parallel channels suggests major advantage of HSI-based serial channels especially for high bandwidth demand.

Acknowledgement

This work was supported in part by the National Science Foundation and Samsung Electronics under grants (CNS-1217102 and SLSI-201301GE001). Nam Sung Kim has a financial interest in AMD and Samsung Electronics.

Reference

[1] Satish Damaraju et al., "A 22nm IA multi-CPU and GPU System-on-Chip," in *ISSCC*, 2012.

[2] Alexander Branover, Denis Foley, and Maurice Steinman, "AMD Fusion APU: Llano," *IEEE Micro*, vol. 32, no. 2, pp. 28-37, 2012.

[3] Bryan Black et al., "Die stacking (3D) microarchitecture," in *MICRO*, 2006.

[4] Gabriel H. Loh and Mark D. Hill, "Efficiently enabling conventional block sizes for very large die-stacked DRAM caches," in *MICRO*, 2011.

[5] Joseph Romen Cubillo et al., "Interconnect design and analysis for Through Silicon Interposers (TSIs)," in *3D Systems Integration Conference*, 2012.

[6] Nagesh Vodharalli, "Needs, challenges and status in low-cost silicon interposers," in *Global Interposer Technology Workshop*, 2012.

[7] "240pin unbuffered DIMM based on 4Gb B-die - Datasheet," Samsung Electronics, Rev. 1.3, May. 2012.

[8] Bryan Casper, "Energy efficient multi-Gb/s I/O: circuit and system design techniques," in *IEEE Workshop on Microelectronics and Electron Devices*, 2011.

[9] Kanit Therdsteerasukdi et al., "Utilizing radio-frequency interconnect for a many-DIMM DRAM system," *IEEE Journal on Emerging and Selected Topics in Circuits and Systems*, vol. 2, no. 2, pp. 210 - 227, 2012.

[10] Kwangmo Jung, Amir Amirkhany, and Kambiz Kaviani, "A 0.94mW/Gb/s 22Gb/s 2-tap partial-response DFE receiver in 40nm LP CMOS," in *ISSCC*, 2013.

[11] Kambiz Kaviani et al., "A 0.4mW/Gb/s 16Gb/s near-ground receiver front-end with replica transconductance termination calibration," in *ISSCC*, 2012.

[12] Chih-Fan Liao and Shen-Iuan Liu, "A 40 Gb/s CMOS serial-link receiver with adaptive equalization and clock/data recovery," *IEEE Journal of Solid-State Circuits*, vol. 43, no. 11, pp. 2492 - 2502, 2008.

[13] Meisam Honarvar Nazari and Azita Emami-Neyestanak, "A 15Gb/s 0.5mW/Gb/s 2-tap DFE receiver with far-end crosstalk cancellation," in *ISSCC*, 2011.

[14] Jonathan E. Proesel and Timothy O. Dickson, "A 20-Gb/s, 0.66-pJ/bit serial receiver with 2-stage continuous-time linear equalizer and 1-tap decision feedback equalizer in 45nm SOI CMOS," in *IEEE Symposium on VLSI Circuits*, 2011.

[15] Yasuhiro Take, Noriyuki Miura, and Tadahiro Kuroda, "A 30 Gb/s/Link 2.2 Tb/s/mm^2 inductively-coupled injection-locking CDR for high-speed DRAM interface," *IEEE Journal of Solid-State Circuits*, vol. 46, no. 11, pp. 2552 - 2559, 2011.

[16] Thomas Toifl et al., "A 3.1mW/Gbps 30Gbps quarter-rate triple-speculation 15-tap SC-DFE RX data path in 32nm CMOS," in *IEEE Symposium on VLSI Circuits*, 2012.

[17] Huaide Wang and Jri Lee, "A 21-Gb/s 87-mW transceiver with FFE/DFE/Analog equalizer in 65-nm CMOS technology," *IEEE Journal of Solid-State Circuits*, vol. 45, no. 4, pp. 909-920, 2010.

[18] Yu-Ming Ying and Shen-Iuan Liu, "A 20Gb/s digitally adaptive equalizer/DFE with blind sampling," in *ISSCC*, 2011.

[19] Brinda Ganesh, Aamer Jaleel, David Wang, and Bruce Jacob, "Fully-buffered DIMM memory architectures: understanding

mechanisms, overheads and scaling," in *HPCA*, 2007.

[20] "IBM Power 795 (9119-FHB) technical overview and introduction," IBM, 2nd Ed., Feb. 2013.

[21] "Intel® 7500/7510/7512 scalable memory buffer. Datasheet," Intel, Apr. 2011.

[22] (2007, July) The AMD Memory Roadmap: DDR3, FBD and G3MX Examined. [Online]. http://www.anandtech.com/show/2287/4

[23] Rachata Ausavarungnirun, Kevin Kai-Wei Chang, Lavanya Subramanian, Gabriel H. Loh, and Onur Mutlu, "Staged memory scheduling: achieving high performance and scalability in heterogeneous systems," in *ISCA*, 2012.

[24] Joe Jeddeloh and Brent Keeth, "Hybrid Memory Cube: new DRAM architecture increases density and performance," in *VLSIT*, 2012.

[25] "Hybrid memory cube specification," Hybrid Memory Cube Consortium, Rev. 1.1, Feb. 2014.

[26] Elliott Cooper-Bails, Paul Rosenfeld, and Bruce Jacob, "Buffer-on-board memory systems," in *ISCA*, 2012.

[27] Edouard Bugnion, Jennifer M. Anderson, Todd C. Mowry, Mendel Rosenblum, and Monica S. Lam, "Compiler-directed page coloring for multiprocessors," in *ASPLOS*, 1996.

[28] Jiang Lin et al., "Gaining insights into multicore cache partitioning: Bridging the gap between simulation and real systems," in *HPCA*, 2008.

[29] Lei Liu et al., "A Software Memory Partition Approach for Eliminating Bank-level Interference in Multicore Systems," in *PACT*, 2012.

[30] Sai Prashanth Muralidhara, Lavanya Subramanian, Onur Mutlu, Mahmut Kandemir, and Thomas Moscibroda, "Reducing memory interference in multicore systems via application-aware memory channel partitioning," in *MICRO*, 2011.

[31] Brian Leibowitz et al., "A 4.3 GB/s Mobile Memory Interface With Power-Efficient Bandwidth Scaling," *IEEE Journal of Solid-State Circuits*, vol. 45, no. 4, pp. 889-898, 2010.

[32] M. Hossain et al., "A 400MHz - 1.6GHz fast lock, jitter filtering ADDLL based burst mode memory interface," in *IEEE Symp. on VLSI Circuits (VLSIC)*, 2013, pp. 244-245.

[33] Krishna T. Malladi et al., "Rethinking DRAM Power Modes for Energy Proportionality," in *MICRO*, 2012.

[34] Nathan Binkert et al., "The gem5 simulator," *ACM SIGARCH Computer Architecture News*, vol. 39, no. 2, pp. 1-7, 2011.

[35] Ali Bakhoda, George L. Yuan, Wilson W. L. Fung, Henry Wong, and Tor M. Aamodt, "Analyzing CUDA workloads using a detailed GPU simulator," in *ISPASS*, 2009.

[36] Hao Wang, Vijay Sathish, Ripudaman Singh, Michael J. Schulte, and Nam Sung Kim, "Workload and power budget partitioning for single-chip heterogeneous processors," in *PACT*, 2012.

[37] AMD A10-6800K specifications. [Online]. http://www.cpu-world.com/CPUs/Bulldozer/AMD-A10-Series%20A10-6800K.html

[38] (2014, Apr.) Radeon HD 8000 Series. [Online]. http://en.wikipedia.org/wiki/Radeon_HD_8000_Series

[39] Shuai Che et al., "Rodinia: A benchmark suite for heterogeneous computing," in *IISWC*, 2009.

[40] Hongzhong Zheng, Jiang Lin, Zhao Zhang, and Zhichun Zhu, "Decoupled DIMM: Building high-bandwidth memory system using low-speed DRAM devices," in *ISCA*, 2009.

[41] Doe Hyun Yoon, Jichuan Chang, Naveen Muralimanohar, and Parthasarathy Ranganathan, "BOOM: Enabling mobile memory based low-power server DIMMs," in *ISCA*, 2012.

[42] Sujay Phadke and Satish Narayanasamy, "MLP aware heterogeneous memory system," in *DATE*, 2011.