Peering Over the Memory Wall: Design Space and Performance Analysis of the Hybrid Memory Cube

University of Maryland Systems & Computer Architecture Group Technical Report UMD-SCA-2012-10-01

Paul Rosenfeld*, Elliott Cooper-Balis*, Todd Farrell*, Dave Resnick°, and Bruce Jacob†
*Micron, Inc. °Sandia National Labs †University of Maryland

Abstract

The Hybrid Memory Cube is an emerging main memory technology that leverages advances in 3D fabrication techniques to create a CMOS logic layer with DRAM dies stacked on top. The logic layer contains several DRAM memory controllers that receive requests from high speed serial links from the host processor. Each memory controller is connected to several memory banks in several DRAM dies with a vertical through-silicon via (TSV). Since the TSVs form a dense interconnect with short path lengths, the data bus between the controller and banks can be operated at higher throughput and lower energy per bit compared to traditional DDRx memory.

This technology represents a paradigm shift in main memory design that could potentially solve the bandwidth, capacity, and power problems plaguing the main memory system today. While the architecture is new we present several design parameter sweeps and performance characterizations to highlight some interesting issues and trade-offs in the HMC architecture. First, we discuss how to best select link and TSV bandwidths to fully utilize the HMC's potential throughput. Next, we analyze several different cube configurations that are resource constrained to try to understand the trade-offs in choosing the number of memory controllers, DRAM dies, and memory banks in the system. Finally, to present a general characterization of the HMC's performance, we compare the execution of HMC, Buffer-on-Board, and quad channel DDR3-1600 main memory systems showing that for applications that can exploit HMC's concurrency and bandwidth, a single HMC can reduce full-system execution time over an extremely aggressive quad channel DDR3-1600 system by a factor of two.

1. Introduction

In recent years, the modern main memory system has been unable to maintain pace with ever-improving CPU designs. With today's multi-core and multi-threaded processors with aggressive pipelines and out of order scheduling, the demands on the main memory system are more onerous than ever. There are three major problems that today's systems encounter:

- Memory bandwidth is insufficient to meet the demands of modern chip multiprocessors as the number of cores on chip increases
- Memory capacity per core is insufficient to meet the needs of server and high performance systems because the capacity per channel is not increasing significantly [9, 10]
- Memory power consumption is beginning to dominate in large systems (i.e., High Performance Computing) [19]

In this paper we examine a new memory technology called the hybrid memory cube which has the potential to address all three of these problems at the same time. The HMC technology leverages advances in fabrication technology to create a 3D stack that contains a CMOS logic layer with several DRAM dies stacked on top. The logic layer contains several memory controllers, a high speed serial link interface, and an interconnect switch that connects the high speed links to the memory controllers. By stacking memory elements on top of controllers, the HMC can deliver data from memory banks to controllers with much higher bandwidth and lower energy per bit than a traditional memory system.

Although the HMC addresses the capacity and power problems, this paper will focus on attempting to characterize the performance of the HMC, examining the issues that must be considered in order to optimize a cube's performance. The rest of the paper is structured as follows: section 2 will briefly review the operation and problems with current SDRAM memory systems and discuss some currently proposed solutions to these problems, section 3 will give some background on the HMC architecture and discuss some of the benefits of this serially-attached stacked DRAM technology, section 4 provides an overview of our methodology, several design space explorations follow in section 5; random stream and full system performance results are presented in section 6; and finally we discuss related work in section 7 and conclude.

2. SDRAM Issues and Proposed Solutions

The modern DDRx memory system consists of a memory controller which issues commands to a set of memory devices that are soldered to a DIMM which is plugged into the motherboard. The memory controller issues commands over a dedicated set of command wires (i.e., RAS, CAS, WE, etc.), along with the addresses on a dedicated address bus. Data is transferred to and from the memory controller over a wide bi-directional 64-bit data bus. Transfers on the data bus happen on both the rising and falling edges of the DRAM clock.

The wide parallel data bus in DDRx systems creates a scaling problem. As the DRAM clock rates increase to try to keep pace with CPU bandwidth demand, the signal integrity degrades due to increased crosstalk and signal reflection. The problem is exacerbated by the fact that electrical contact to the DIMMs is maintained by physical pressure of the DIMM slot and not a permanent connection such as with solder. This has created a situation where as the DRAM clock increases, motherboards support fewer and fewer DIMMs per channel.

2.1. Proposed DDRx Solutions

Recently, industry has come up with several solutions to try to ameliorate the bandwidth and capacity issues while keeping the core DDRx technology unchanged.

2.1.1. DDR4. At the time of this writing, JEDEC is working to create a DDR4 standard which will serve as the replacement for current DDR3 technology. DDR4 represents the most incremental solution to the signal integrity problem: reduce the load on the memory bus by only allowing a single DIMM per memory channel. By limiting channels to a single DIMM, DDR4 is expected to scale at least to 3.2GT/s (2x the data rate of DDR3-1600 [11, 13]). Although this approach delivers higher bandwidth, it exacerbates the capacity problem: it is difficult to put hundreds of gigabytes of memory into a server if one can only install a single DIMM per channel. Since a single memory channel requires hundreds of CPU pins, scaling the number of channels to compensate for the loss of channel capacity seems an unlikely solution. Finally, the low availability and high cost of high density DIMMs often makes them impractical for big systems where the memory might cost an order of magnitude more than the other hardware in the system. However, the DDR4 standard will likely include extensions for stacked memory devices using TSVs as well as switch fabrics to help alleviate the capacity problem [13].

DDR4 introduces advanced I/O techniques such as Dynamic Bus Inversion, Pseudo Open Drain, along with new On Die Termination techniques to reduce bus power consumption and increase signal integrity. DDR4 devices will operate at 1.2V, resulting in a substantial energy savings over current DDR3 devices, which run at 1.35V. In addition to an increased I/O data rate, bank groups in DDR4 will add an extra level of parallelism inside of the DRAM device to help sustain higher throughput.

As a solution, DDR4 is able to incrementally increase the main memory bandwidth and lower the power consumption; however it sacrifices significant main memory capacity by requiring a single DIMM per channel.

2.1.2. LRDIMM. Load Reduced DIMM (LRDIMM) takes the approach of reducing the capacitive load on the memory bus by adding latching registers to the control, address, and data lines[1]. Since the electrical connections on the bus are now made to a single register as opposed to multiple devices in multiple ranks, the loading on the bus is reduced. The load reduction mitigates the capacity problem by allowing more DIMMs to be placed per channel than traditional DDRx while maintaining reasonably high clock speeds. For example, LRDIMM allows three DIMMs per channel at 1333MT/s at 1.5V whereas Registered DIMM (RDIMM) only allows two [8].

Although this is a step in the right direction, LRDIMM targets primarily the capacity problem, as the transfer rates of LRDIMM are the same as normal DDR3 devices. Other than the register chip, the core memory technology is unchanged in LRDIMM.

2.1.3. FB-DIMM. In 2007, JEDEC approved a new memory standard called Fully Buffered DIMM (FB-DIMM). FB-DIMM places a slave memory controller called an Advanced Memory Buffer (AMB) on each memory module. The modules communicate with the main memory controller using a high speed, full-duplex point-to-point link instead of a wide parallel bus. Since all connections between modules are point-to-point, each memory module must capture the data from the link and either process the request locally or forward the request to the next module in the chain.

By replacing the wide DRAM buses with high speed, point-to-point links, many more DIMMs can be placed in a channel while maintaining high data rate. Though FB-DIMM addressed the bandwidth and capacity problems of the memory system, it was never widely adopted due to power issues: the power consumption of the AMB proved to be the biggest problem with FB-DIMM since the AMB added a non-trivial power overhead to each DIMM. FB-DIMM allowed approximately 24x the number of DIMMs in the memory system [7] as compared to a DDR3 system and added approximately 4W of power overhead per DIMM [7, 18] resulting in power overheads that could exceed 100W. The power overhead of the memory was on par with CPU power consumption at the time.

In the end, FB-DIMM was abandoned by vendors and taken off industry road maps altogether.

2.1.4. Buffer on Board. Yet another approach to increasing capacity and bandwidth is the "buffer on board" (BOB) memory system. This type of memory system has been implemented by the major vendors (Intel, IBM, etc.) for their high end server systems. The BOB memory system is comprised of a master memory controller on the CPU die communicating with several slave memory controllers on the motherboard over high speed, full-duplex serial links. Whereas the CPU communicates with each slave controller using a packet-based protocol, the slave controllers communicate with commodity DDR DIMMs over a standard DDR memory channel. Each slave controller can control one or more DRAM channels.

By splitting off each slave controller and allowing it to act as a buffer between the wide DRAM channel and the CPU, the BOB memory system can achieve both high bandwidth and very large capacity. The capacity is increased because the serial interface requires far fewer CPU pins per channel as compared to a DDR channel. A large number of memory channels can use the same number of CPU pins as just a few DDR channels. Unlike the FB-DIMM memory system, which requires a slave controller for each memory module, the BOB system only requires a slave controller at the channel level. Although this reduces the number of extra controllers required in the system, a non-trivial power penalty remains to run them.

2.1.5. Problems With Proposed Solutions. Most of the previously discussed solutions take the approach of tweaking the

[1] This is in contrast to a Registered DIMM that only latches the high fan-out control and address signals, but not the data bus.

electrical characteristics or adding additional logic to commodity DIMMs. Overall, each of these solutions are reasonable for the short term, but none is viable in the long term since none addresses the main memory problem along all three dimensions. DDR4 lowers power consumption and increases bandwidth but fails to address capacity. LRDIMM bolsters capacity but keeps bandwidth the same while slightly increasing power consumption. FB-DIMM and Buffer on Board increase both bandwidth and capacity but impose non-trivial power overheads.

The reason none of these technologies can address the entire problem is that they maintain the DDRx memory technology at their core. In order to solve all three problems (bandwidth, capacity, and power) at the same time, it appears that the core DDRx memory technology must be abandoned and a new paradigm established.

3. Hybrid Memory Cube

3.1. Architecture Overview

The ***Hybrid Memory Cube*** (HMC) [2, 21] is a new memory device which is comprised of a CMOS logic layer die with several DRAM dies stacked on top. The logic layer contains several memory controllers which communicate with the memory storage elements in the DRAM dies through a vertical set of interconnects called ***Through-Silicon Vias*** (TSVs). Because the TSVs provide a very short interconnect path between dies and have a lower capacitance than PCB traces, data can be sent at a high data rate through the stack. Furthermore, smaller I/O drivers and simplified routing allow for a high density interconnect between the dies.

When stacked together, the logic layer die and the memory dies form a ***cube***. An illustration of the overall cube architecture can be seen in Figure 1. When discussing the hybrid memory cube, we will use the terminology set forth in the HMC specification [2].

3.1.1. TSVs & DRAM Stack. In order to increase the parallelism of the architecture, the dies are segmented vertically into independent ***vaults***. At the base of the DRAM stack, the CMOS logic layer contains one vault controller per vault as well as I/O and switch circuitry to move requests between the high speed links and vaults. Each vault controller controls several vertically stacked ***partitions*** having one or more memory banks.

A vault is roughly the equivalent of a traditional DDRx channel since it contains a controller and several independent banks of memory that all share a bi-directional data bus. However, unlike a traditional DDRx system, these connections are vastly shorter and more dense than the DDRx bus traces on a motherboard. Each partition is akin to a rank in a traditional DDRx system since it shares a TSV bus with the other partitions. Compared to a traditional DDRx system that may have up to four independent channels per CPU socket, an HMC could potentially provide the equivalent of 16 or more

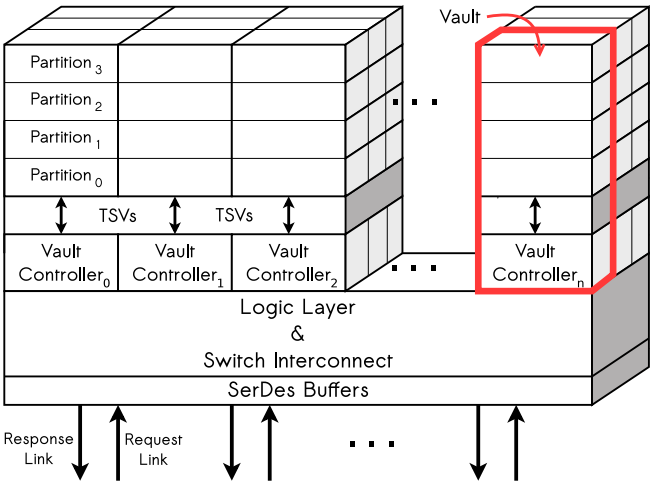

Figure 1: The HMC Architecture. Requests flow over the high speed links to an interconnect that transmits them to their target vault controller. Each vault controller sends commands to write/read data to/from the memory banks in each partition. After the DRAM access completes, data is transferred back through the interconnect to the high speed links where it is sent toward the CPU.

independent channels for higher memory parallelism.

3.1.2. Logic Layer I/O. Each cube communicates with the host processor or with other cubes over a series of full-duplex ***links***. Each link is comprised of several high speed lanes that typically run at several gigabits per second per bit-lane (gigabytes per second per link). A link contains independent request and response paths that are unidirectional and differentially signalled. Links contain buffers that queue requests before serializing the data into 16 byte flits [2]. Upon arriving at the other end of the link, the requests are de-serialized and error checked. A switch interconnect implemented in the logic layer routes requests and responses between the links and local vault controllers. Requests from any link can access any vault inside of a cube. If requests are not destined for a local cube, they can be routed to another link to be transmitted to another cube.

3.2. HMC Benefits

3.2.1. Capacity. One of the clear benefits of an HMC architecture is that it addresses the capacity and density problems of current DRAM technology. The capacitors inside of a DRAM die must maintain a minimum capacitance to be able to store charge long enough to avoid corruption or incessant refreshing. It is difficult to shrink the DRAM cell size while maintaining the same capacitance, and thus improvements in DRAM density have slowed in recent years. Due to the minimum required capacitance, it is unclear how much longer DRAM can continue to scale down to improve the density of a single DRAM die.

Through-silicon vias allow multiple DRAM dies to be

stacked together (currently demonstrated parts have 4 dies, but 8 dies have been mentioned). By adding dies vertically, a single cube can contain 4 or 8 times the storage in the same package footprint without having to further increase DRAM cell density.

Furthermore, since multiple cubes can be chained together, it is possible to add capacity by adding extra cubes. Since the high speed link interfaces regenerate the signal at each link, theoretically any number of cubes could be chained together (though in practice this will be limited by latency and loading considerations).

3.2.2. Parallelism and Aggregate Bandwidth. The TSVs provide a high bandwidth connection between the logic layer and the DRAM dies and can transfer data at over a hundred gigabytes per second. Since there are many independent vaults, each with one or more banks, there is a high level of parallelism inside of the cube. The high level of parallelism and the high density of the TSVs allow for a large number of relatively slow DRAM devices to take advantage of the high aggregate bandwidth provided by the links. Overall, the total aggregate bandwidth can be an order of magnitude higher than current DDRx systems (for example, [22] shows a sample HMC chip running with a prototype Intel CPU at 121GB/s).

3.2.3. Energy Efficiency. By radically decreasing the length and capacitance of the electrical connections the memory controller and the DRAM devices, the HMC is more energy efficient compared to DDRx memory devices. Additionally, since much of the peripheral circuitry is moved into the logic layer, the power cost of this circuitry is amortized over a large number of DRAM devices, saving on overall power consumption. Claims about energy efficiency range anywhere from 7x [1] to 10x [17] over current generation memory systems.

3.2.4. Device Heterogeneity. Since a TSV process allows for heterogeneous dies to be stacked together, each die can be optimized for a specific purpose without having to make the typical performance/power/density sacrifices required when using a process not optimized for the application. The CMOS logic layer is optimized for switching and I/O, while the DRAM dies are optimized for density and data retention. If these two dies were to co-exist in a single fabrication process, they would both suffer (i.e., if DRAM is built in a logic process, it can't be dense; if switching logic is built in a DRAM process, it can't switch quickly). As a result of the stacking, each die achieves good performance and energy efficiency while being closely linked by a high speed TSV interface.

3.2.5. Interface Abstraction. The original SDRAM standard purposely created many generations of "dumb" memory devices: the memory controller was responsible for ensuring all timing constraints were met. This enabled the DRAM devices to contain a minimal amount of circuitry that wasn't related to reading the DRAM array or transferring data. While this was a rational design decision at the time, it had the effect of curtailing innovation in the memory system: once the standard was written, nothing could be done to change the protocol. Any deviations from the standard protocol required the agreement of DRAM manufacturers, motherboard manufacturers, CPU manufacturers, etc. Any additions inside of a DIMM that were not protocol-visible would add to the product cost in a commodity market where consumers make their decisions based largely on price.

This problem was further exacerbated when the memory controller migrated onto the CPU die. The CPU now had to be intimately aware of the timing constraints of the particular memory it was driving.

In the Hybrid Memory Cube, the device at the end of the communication link is no longer "dumb" in that the CPU can communicate with the cube over a general protocol that is then converted into device-specific commands within the vault controller. This allows for innovation in a number of different ways. The first improvement is that the DRAM timing inside of the cube can be changed without changing the CPU interface. Furthermore, even the memory technology inside of the stack could be completely changed without affecting the interface. A second benefit of an abstract interface is that it allows any communication medium that is capable of delivering packets between a CPU and cube to be used. Already, researchers are thinking about how to replace electrical SerDes with high speed optical interconnects [24, 25, 27] to decrease power consumption.

4. Methodology

In the following experimental sections, we used HMCSim, a cycle-based simulator developed by our research group for Micron as part of their multi-year HMC design process.

For the first set of experiments (sections 5 and 6.1), we perform limit-case studies by using "toy" benchmarks such as random stream that are designed to emulate the behavior of HPC applications and systems, which can exploit extremely high bandwidth and exhibit almost no locality whatsoever.

For the second set of experiments (section 6.2), we perform full system simulation of applications from several benchmark suites on Linux using the MARSSx86 full system simulator [20], modeling a 16-core x86 CPU.

5. Design Space Exploration

The HMC architecture contains quite a large design space to be explored in order to optimize the performance.

Within the DRAM stack, decisions must be made as to how to expose the proper level of memory parallelism to effectively utilize the available TSV and link bandwidth. This includes choices such as the number of independent vaults, the number of DRAM dies that should be stacked on top, and how many banks are required. Each of these variables represent a performance trade-off within the cube. Outside of the DRAM stack, the choice of link bandwidth has a fundamental impact on the performance.

First, we discuss how the effective link bandwidth varies

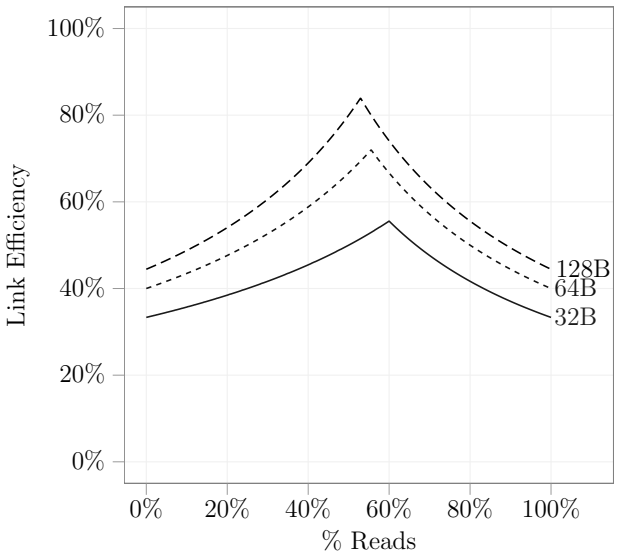

Figure 2: Link efficiency for various read/write ratios assuming a 16 byte packet overhead. For a 64 Byte request size, 56% reads yields a peak link efficiency of 72%.

with the request size and read/write ratio and how this impacts the selection of TSV and link bandwidths. Then we explore the trade-offs in stacking DRAM by modeling several "fixed resource" configurations: different cube configurations that have the same number of banks/TSVs/switch bandwidth/buffer space but are organized in different ways. Through these experiments, we show that, although the resources available are identical, their organization can have a non-trivial performance impact.

5.1. Link and TSV Bandwidth Selection

5.1.1. Packetization Overhead. The CPU communicates with the HMC using symmetric full-duplex serial links that transmit requests and data enclosed in packets. Similar to network traffic, each packet contains information necessary for routing, flow control, error checking, etc. in addition to the request or data. Unlike a single bi-directional bus, each link is composed of a dedicated request and response path which makes it sensitive to the read/write ratio: write data only flows on the request link while read data only flows on the response link. For example, if the memory access stream contains only writes, only the request link will be utilized, while the response link will be idle.

We compute the theoretical peak link efficiency for different read/write ratios by computing the ratio of data cycles to total cycles (where total cycles include data cycles, idle link cycles, and packet overhead cycles). It would be reasonable to expect that for symmetric links the ideal read/write ratio would be approximately 50%: write data would keep the request link busy while read data would keep the response link busy. Figure 2 shows the impact of the read/write ratio on the peak link efficiency for varying packet sizes with a 16 byte packet overhead. The interesting feature of this graph

Request Size	Peak Efficiency	Raw Bandwidth	Effective Peak Bandwidth
32B	55.6% at 60% Reads	80 GB/s	44.4 GB/s
		160 GB/s	88.9 GB/s
		240 GB/s	133.3 GB/s
		320 GB/s	177.8 GB/s
64B	71.9% at 56% Reads	80 GB/s	57.6 GB/s
		160 GB/s	115.1 GB/s
		240 GB/s	172.7 GB/s
		320 GB/s	230.2 GB/s
128B	83.9% at 53% Reads	80 GB/s	67.1 GB/s
		160 GB/s	134.2 GB/s
		240 GB/s	201.3 GB/s
		320 GB/s	268.5 GB/s

Table 1: Effective link bandwidth for various request sizes and a 16 byte overhead. Larger requests achieve higher effective bandwidth on the links.

is that the peak efficiency for all request sizes is greater than 50% reads. This is because of the fact that a read request incurs a packet overhead twice (once for the request packet and once for the response packet), and so the stream must contain a greater number of read requests to keep both links fully occupied. Table 1 summarizes the effective peak bandwidth (corresponding to the peak of each curve in Figure 2 of various link configurations and packet sizes.

It is also important to note that for all cases, it is impossible to use 100% of the link bandwidth to transmit data, due to packet overheads. As the request size becomes larger, the peak efficiency grows and shifts toward lower read/write ratios since the read request overhead is amortized over the larger data size. Larger request sizes also help to increase the link efficiency.

5.1.2. Selecting Link/TSV Bandwidth. From the discussion in the previous section it can be concluded that in order to counteract the reduction in effective link bandwidth, the link throughput should be greater than the aggregate throughput of the DRAM devices in order to maximize system-level throughput. To determine the proper link bandwidth for a specific aggregate TSV bandwidth, we configure two typical HMC configurations: 128 total banks in 16 vaults and 4 DRAM dies and 256 total banks in 16 vaults with 8 DRAM dies [21]. Then we select an aggregate TSV bandwidth and vary the aggregate link bandwidth based on typical values given in the HMC specification [2]. For this experiment, the switch parameters are set to provide effectively unlimited bandwidth between the links and the vaults so as to eliminate its impact. Figure 3 shows the results of several different link and TSV bandwidth combinations for three read/write ratios for the 128 and 256 total bank configurations.

In all cases, increasing the TSV bandwidth beyond 160GB/s yields diminishing returns (i.e., doubling the TSV bandwidth from 160 GB/s to 320 GB/s results in a marginal performance increase). 256 banks are able to provide a higher degree of memory parallelism which are able to more effectively utilize

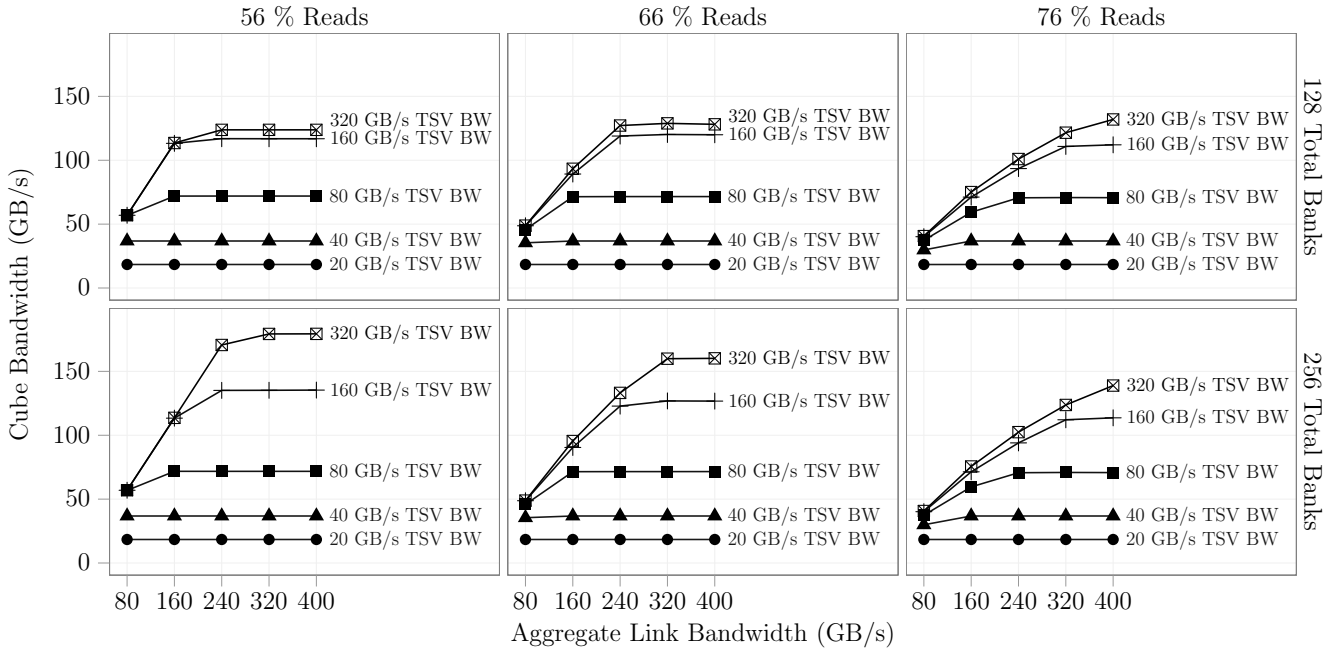

Figure 3: Link and TSV bandwidths for several read/write ratios. Link bandwidth needs to be over-provisioned with respect to the TSV bandwidth. However, increasing TSV bandwidth past 160GB/s and link bandwidth past 240GB/s yields diminishing returns. Configurations: 128 total banks: 16 vaults, 4 DRAM dies; 256 total banks: 16 vaults, 8 DRAM dies.

higher TSV bandwidths. As the link bandwidth increases, the overall throughput flattens out, indicating that the DRAM is unable to utilize the extra available bandwidth. Since the link overheads are significant at 66% and 76% reads, higher link throughput is required to drive the DRAM to the peak bandwidth. However, if the TSV bandwidth is fixed at 160 GB/s as discussed above, increasing the link bandwidth beyond 240 GB/s does not yield a significant advantage (except in the 76% case which must overcome a low effective link bandwidth).

Given these results, it appears that it is most efficient to pair 160 GB/s TSV bandwidth with 240 GB/s link bandwidth. This result matches Figure 2 since the peak link bandwidth for 64 byte requests is 172.7 GB/s, which most closely matches the 160 GB/s TSV bandwidth.

5.2. Fixed Resource Configurations

The discussion in the previous section assumes a specific 128 and 256 bank configuration (16 vaults, 4 or 8 DRAM dies). In this section we expand our design space by exploring a group of configurations which are constrained by a fixed set of resources but organized in different ways. That is, given a fixed aggregate TSV bandwidth, fixed available queue space in the logic layer, fixed switch bandwidth, fixed number of banks, etc. is there an optimal way to structure these resources?

The first experiment focuses on choosing the number of vaults and DRAM dies, while the second experiment examines total bank allocation.

5.2.1. Number of Vaults and DRAM Dies. In this section, we attempt to quantify the trade-offs of taking a fixed set of banks and creating different cube configurations out of them. For example, one can configure a cube with only a few vault controllers and many DRAM dies stacked on top. This would allow each vault controller to have extremely wide TSV buses to the memory banks, but it could potentially be limited by a lack of parallelism from having too few independent channels. At the other end of the spectrum, one can envision a cube with many vaults and fewer dies stacked on top. Such a configuration would create more parallelism from independent channels at the expense of having a narrower data path between the controller and memory.

We use a random stream with 56% reads and hold the aggregate TSV bandwidth at a constant 160 GB/s and the aggregate link bandwidth at a constant 240 GB/s (as discussed in section 5.1.2). Note that in section 5.1.2 the switch bandwidth was effectively infinite so as not to cause a bottleneck between the link and DRAM, whereas in this experiment the aggregate switch bandwidth is finite and held constant among all configurations (i.e., doubling the number of vaults halves the switch bandwidth available for an individual vault). Additionally, we hold the total number of queue entries available in the logic layer to be constant (such as command queues and queues that hold return data) such that doubling the number of vaults halves the available queue space in each vault. By also holding the total number of banks constant, the bandwidth per bank is also identical for all configurations. Therefore, the performance differences are solely the result of the interaction between the various components.

We track the utilization of each TSV bus by counting the

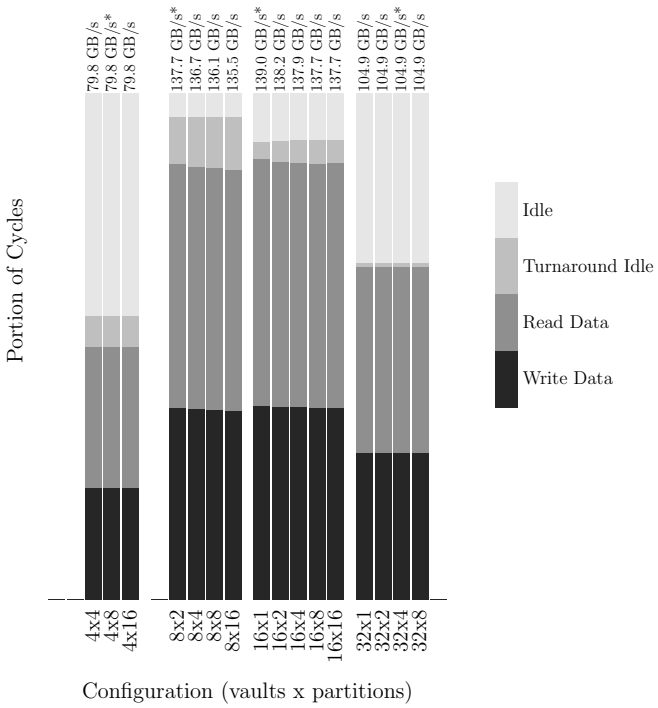

Figure 4: TSV utilization of various 256 bank configurations with 56% reads. The black and dark grey components represent useful throughput on the TSVs while the light greys are idle cycles. Bars are grouped by the number of vaults in the configuration and the top of each bar is labeled with the resulting bandwidth. The highest bandwidth configuration in each group of bars is denoted by an asterisk. Note that some configurations are invalid and are denoted by missing bars (ex: a configuration with 32 vaults and 16 dies would require 0.5 banks per partition, which is considered invalid).

number of cycles a TSV is transmitting data or being idle. Figure 4 shows the breakdown of TSV utilization averaged among all the TSVs. The darker components represent the portion of cycles spent transmitting useful data (reads and writes) while the lighter components represent two kinds of idle times when the TSVs are not being used to send data. The "turnaround idle" component represents equivalent of rank-to-rank switching time and write-to-read turnaround time in a traditional DDRx system. Since partitions are analogous to ranks in a traditional DDRx system (i.e., multiple partitions share a common TSV bus), the simulator adds an idle cycle between back-to-back data bursts from different partitions and between reads and writes. The final idle component represents the inability of the banks to utilize the TSVs for some reason (e.g., bank conflicts, insufficient request rate, etc.)

Bars are grouped by the number of vaults, and each bar within a group represents a different number of DRAM dies in the stack. Configurations requiring less than one bank per partition or greater than 16 banks per partition are considered invalid. A bar with an asterisk top represents the highest performing configuration for that group of bars.

It becomes immediately apparent that the configurations at the extremes (4 vaults and 32 vaults) do not perform as well as the others, as a large portion of their cycles are spent idling. In the four vault case, this is due to the fact that the each vault has a very wide TSV bus that transfers an entire 64 byte request in much fewer cycles than the DRAM access time. Therefore, the I/O and DRAM operations are not effectively overlapped, causing an inefficient use of the TSVs.

At the other end of the spectrum, the 32 vault configuration is limited by the interconnect between the links and the vaults. Since the aggregate switch bandwidth is held constant, each individual vault has limited switch paths to move requests in and out of the controller, thus causing sub-optimal request scheduling. This can be seen in the fact that, although this configuration has narrow TSVs and the data takes many cycles to transmit, the turnaround idle component is very small — indicating that the controller is unable to schedule requests to transmit back-to-back on the TSVs. This is also exacerbated by the limited queue depths per controller, which diminishes the opportunity for requests to be scheduled around conflicts.

The middle configurations (8 and 16 vaults) are able to efficiently move data in and out of the controller as well as to overlap I/O and DRAM access time. As the number of DRAM dies in the stack grows, there is a slight increase in the turnaround idle component. This is to be expected as the probability of reads to different partitions grows with the number of partitions and thus requires more turnaround cycles to be inserted.

As the number of vaults increases from 8 to 16, the turnaround component decreases significantly. Compared to a 16 vault configuration, each vault in the 8 vault configuration must service twice as many requests (albeit with twice as much available TSV bandwidth). The increased request load leads to a higher probability of needing to insert turnaround cycles. The problem is further accentuated by the fact that the relative cost of idling a wider bus for a single turnaround cycle is higher than idling a narrower bus for a single cycle (i.e., if a request typically takes n cycles followed by a cycle of turnaround, doubling the bus throughput will reduce the data time to $n/2$ cycles while keeping a single cycle turnaround penalty). These two factors together account for the decrease in the relative number of turnaround cycles when going from 8 to 16 vaults.

In this experiment, the configuration with 16 vaults and a single DRAM die on top has the highest performance. However, it is unlikely that it would be feasible to create a single DRAM die containing 256 banks and significant capacity. Therefore, it is fortunate that there is only a small penalty for adding extra banks by adding more DRAM dies to the stack (and thus reducing the required capacity per die).

5.2.2. Total Bank Allocation. The discussion in the previous section was limited to several 256 bank configurations. In

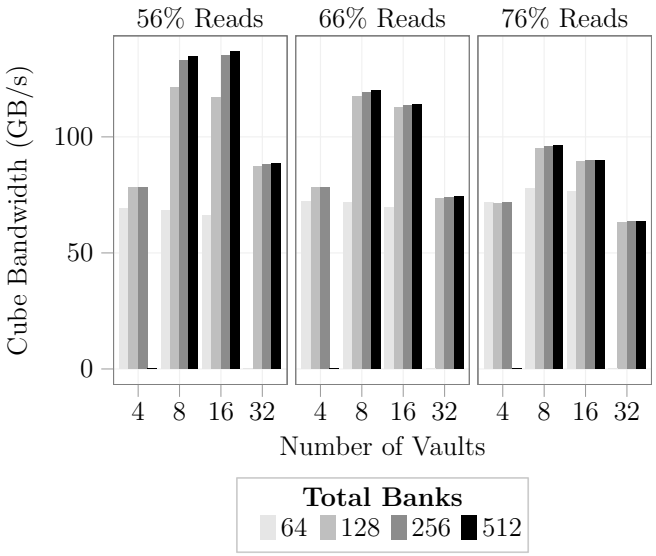

Figure 5: Effect of increasing total number of banks. A configuration with 128 total banks captures most of the throughput offered by 240GB/s link bandwidth and 160GB/s TSV bandwidth.

this section we attempt to determine the proper number of total banks required to efficiently utilize the cube's bandwidth. Each bank contributes a portion of the data bandwidth and thus having an insufficient number of banks would lead underutilized TSVs while too many banks could potentially saturate the available bandwidth.

As in the previous experiment we fix the aggregate link bandwidth at 240GB/s and the TSV bandwidth at 160GB/s as well as fixing the total logic layer queue entries and switch bandwidth. Only configurations with four stacked DRAM dies are chosen, as this yields the maximum number of valid configurations for 64 to 512 total banks.

Figure 5 shows the effect of different numbers total banks for various R/W ratios and numbers of vaults. Each group of bars corresponds to a certain number of vaults, and each colored bar in the group represents a different number of total banks.

As discussed in the previous section, the 4 and 32 vault configurations are unable to make efficient use of the DRAM and so are not sensitive to the total number of banks. The middle two graphs show that the 8 and 16 vault configurations can achieve almost peak performance using 128 banks. In fact, only in the 56% reads case (i.e., peak link performance) does increasing the number of banks to 256 yield a 4.2% and 6.5% increase in throughput in the 8 and 16 vault cases, respectively. Since each bank requires extra circuitry (sense amplifiers, row decoders), a trade-off could be made to build 128 larger banks with less overall circuitry instead of 256 smaller banks. This 128 bank system would provide enough parallelism to capture a large portion of the available throughput while reducing DRAM die complexity and power. Furthermore, some applications may be unable to utilize over one hundred gigabytes per second of memory bandwidth and thus one could reduce the bank-level parallelism and the aggregate throughput to save on cost and power budgets.

In conclusion, this section has highlighted some important issues in the design space of the HMC. The effective link bandwidth is sensitive to the read/write ratio and packetization overhead which requires links to have significantly higher throughput than the TSVs. Adding more capacity to the cube can be achieved by stacking more DRAM dies with only a small performance penalty due to increased turnaround overhead. With 160 GB/s TSVs, a 16 vault configuration is able to reduce the impact of turnaround idle time and make the most effective usage of the TSVs. Finally, in order to utilize 160 GB/s TSV bandwidth, at least 128 banks are required.

6. Performance Analysis

6.1. Memory Technology Comparison

Using our previous design space analysis as a guide, we configure an HMC with 128 total banks, 16 vaults, 4 DRAM dies, 240 GB/s aggregate link bandwidth, and 160 GB/s TSV bandwidth and compare its performance to other currently available memory technologies. We simulate a quad channel DDR3-1600 system in both single and dual rank configurations in DRAMSim2 [23]. Additionally, we simulate a Buffer-on-Board memory system using the BOBSim simulator [5] with four 6.4GHz channels each populated with four DDR3-1600 DIMMs. We implement a common interface to all three simulators and first run it with a random stream generator. The random stream is swept from 5% to 90% reads for all four memory systems and the average data bandwidth seen memory system boundary at the is recorded (i.e., packet overheads in BOB and HMC are not counted as bandwidth).

Figure 6 shows the results of the read/write sweep. Each curve is labeled with the peak achieved bandwidth and the percentage of theoretical peak for that memory system (which is computed based on the peak theoretical channel throughput for each memory system). The single rank DDR3 memory system does not have enough memory-level parallelism to utilize the available data bandwidth, while the dual rank configuration is able to achieve much better throughput. Since the DDR3 memory system uses a single bi-directional bus it is agnostic to read/write ratio. Interestingly, with the exception of the single rank DDR3-1600 system, all three of the memory systems achieve a peak bandwidth that is approximately 85% of their theoretical peak bandwidth.

As in the HMC, the Buffer on Board memory system uses full-duplex serial links with dedicated request and response paths to communicate with the memory devices. This makes the BOB system sensitive to the read/write ratio, but the peak performance is shifted toward higher read/write ratios since the links are asymmetric (request links are wider than response links to account for packet overheads).

Although the ideal link utilization graph shown in section

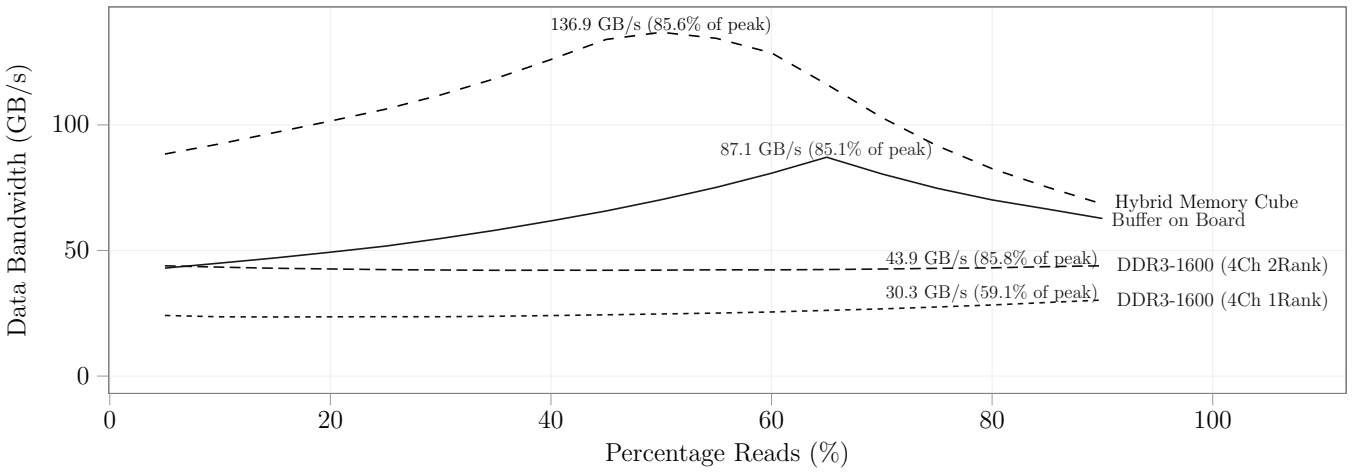

Figure 6: Memory technology comparison. Curves are labeled with their peak bandwidth and the percentage of the peak theoretical channel bandwidth.

5.1.1 has a sharp point, the HMC performance graph has a smoother curve near the maximum bandwidth value. This is partially due to the fact that the links are configured with more available bandwidth than the TSVs (even with packet overheads, the links provide greater than 160 GB/s effective peak bandwidth between 50% reads and 60% reads). Additionally, as the number of reads increases, the TSV turnaround overhead grows larger due to a higher incidence of reads going to different partitions and thus decreasing the TSV utilization. This shifts the maximum throughput to a peak at 50% reads from the peak link efficiency at 55.6% reads for 64 byte requests.

Overall, even when choosing aggressive comparison points (quad channel DDR3 and BOB), the HMC outperforms these technologies over the entire spectrum of read/write ratios. Although the performance results presented here show promise for delivering the higher bandwidth that modern CMPs demand, our discussion has not included power. Previous sources report 7-10x power savings as compared to DDR3 which would make the HMC technology even more attractive, as it would support much higher throughput while simultaneously using less energy per bit.

6.2. Full System Simulations

Up to this point our discussion has focused on saturating a single cube with a random stream to try to understand how an HMC responds to the most demanding access patterns. In this section we attempt to ascertain a more concrete picture of the HMC's performance for a variety of workloads running in a full system simulator presenting results as full system execution time. Once again, we use a highly aggressive dual rank, quad channel DDR3-1600 system as our comparison baseline.

We augment the MARSSx86 [20] full system simulator to utilize our HMC simulator as well as using the publicly available DRAMSim2 bindings for MARSSx86 for the DDR3 comparison. MARSSx86 is a cycle-based simulator that models an out-of-order, superscalar x86 multi-core CPU. It combines the emulation capabilities of QEMU with very detailed x86 timing models which allows it to boot an unmodified Linux operating system. Once a simulation begins, both user and kernel execution are simulated.

CPU	16 Out of Order x86 cores @ 3.2GHz
L1 Cache	128K L1-I / 128K L2-D
L2 Cache	2MB shared L2

Table 2: MARSSx86 Configuration

Suite	Name	Input Size
PARSEC 2.1	facesim	simlarge
	fluidanimate	simlarge
	streamcluster	simlarge
MANTEVO	miniFE	100x100x100
NAS OMP 3.3	ft	Class A
	lu	Class C
	sp	Class C
Other	STREAM	8M elements

Table 3: Workload Configurations

MARSSx86 is configured to simulate a 16 core CPU running at 3.2GHz and 4GB of memory. A summary of the parameters for MARSSx86 can be found in table 2.

In order to reduce simulation time and to ignore uninteresting parts of the workload such as application initialization and shutdown, the programs are annotated with a "region of interest" hook. These extra function calls are inserted into the workload to start the simulation directly before the core computation begins and stop the simulation after it ends. The PARSEC benchmarks contain predefined regions of interest while the other workloads are annotated by hand to include the primary work done by the program. All of the workloads presented here are compiled with OpenMP support and run as 16 threads inside of MARSSx86. A list of workloads and their

input sizes is given in Table 3.

Since each run simulates an identical portion of a program's execution, it is possible to directly compute the speedup between two runs by comparing the simulated runtime of the region of interest.

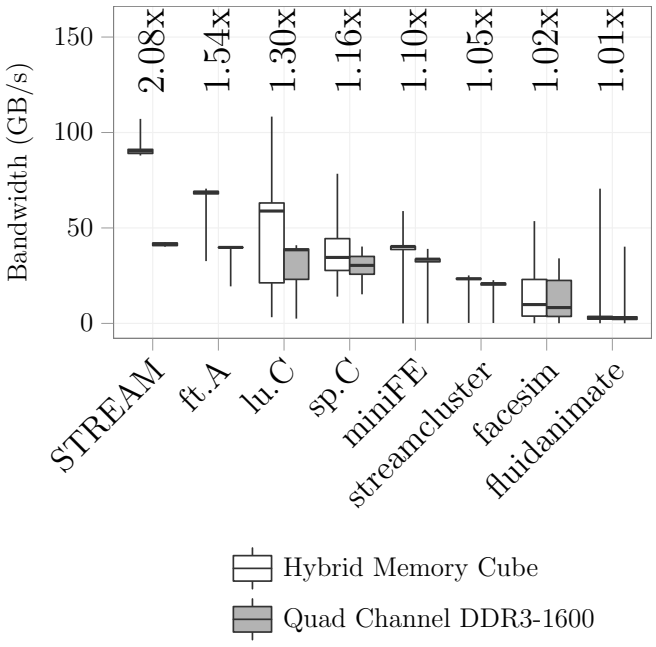

Figure 7: Performance of workloads in full system simulation. The box plot summarizes the bandwidth distribution over time with whiskers extending to the minimum and maximum bandwidth. The speedup of HMC over quad channel DDR3-1600 is shown for each workload.

Figure 7 shows a box plot that summarizes the bandwidth distribution over time for the HMC and quad channel DDR3-1600 memory systems for our full system workloads. Each group of the boxes is labeled with the speedup of the HMC over the DDR3-1600 memory system for a given workload.

Immediately, it becomes evident that, with the exception of STREAM, ft.A, and lu.c, most of the workloads are not able to achieve a significant speedup with the HMC. These benchmarks are not memory bandwidth bound, and so we see that the average bandwidth for both HMC and DDR3 are almost identical. Even STREAM, the most memory intensive benchmark in our set, is unable to stress the HMC enough to reach the throughput levels seen in our previous random stream simulations. Since all other parameters in the full system simulation are identical except for the memory system model used, it is only possible for the HMC to improve the performance of memory intensive sections of the program. To make the problem worse, we have observed in previous experiments that the coherence protocol between the private and shared caches often leads to a bottleneck between the CPU core and main memory.

There is, however, an interesting trend within these benchmarks: long whiskers on many of the workloads indicate that nearly all of the programs have some memory intensive sections that can take advantage of the HMC's performance. This is especially dramatic in fluidanimate, which has the lowest average bandwidth but has a peak bandwidth that exceeds many of the other benchmarks' peaks.

To gain a better understanding of the execution of these workloads, we compare the time-series execution of several benchmarks running in full system mode with the HMC, quad channel DDR3-1600, and "perfect" memory systems. The "perfect" memory system is a model that only adds latency and has infinite bandwidth (i.e., any number of requests can be added per cycle and they are completed after a fixed latency). We add a latency of $t_{RCD} + t_{CL} + t_{Burst}$ for a DDR3-1600 device, which represents the minimum time to open a row and stream data out of that row. Although it the perfect model doesn't represent a realizable memory, it serves as a way to characterize how memory intensive a particular application is. That is, if a particular workload's execution time cannot be significantly decreased with the perfect model, the workload is not memory intensive.

The time-series executions of several workloads is shown in Figure 8. In these graphs, the x-axis represents the simulated time. Since workloads are simulated between identical points in the program, a shorter line indicates a lower execution time. So, for example, the three distinct curves for ft.A show that the DDR3 system takes almost 1500 milliseconds to execute; HMC takes roughly 800; and the ideal execution time is just under 500.

The most memory intensive of the workloads shown in Figure 8 is the STREAM benchmark. All three lines clearly show the 10 identical iterations of STREAM as it executes. Furthermore, in the HMC line it is possible to see the different phases of STREAM within each iteration. The first phase of each iteration is a simple "copy" operation which requires zero FLOPS for two memory accesses, making it the most memory intensive. The subsequent phases (scale, sum, triad) require several FLOPS to do computations in between memory accesses, lowering the memory intensity. The final phase (triad) phase requires the greatest number of floating point computations along with one constant which is likely to hit in the CPU cache. This makes it the least memory intensive and can be seen as the small dip at the end of each iteration. The DDR3 line varies within a much smaller range of bandwidth indicating that all phases of each iteration are saturating the memory system.

It is clear that STREAM is memory intensive by the drastic execution time differences between the three memory models: as the amount of available bandwidth increases from DDR3-1600 to HMC to perfect, each of the 10 iterations of STREAM become more compressed in time. With the increased memory parallelism and bandwidth of the HMC, STREAM executes

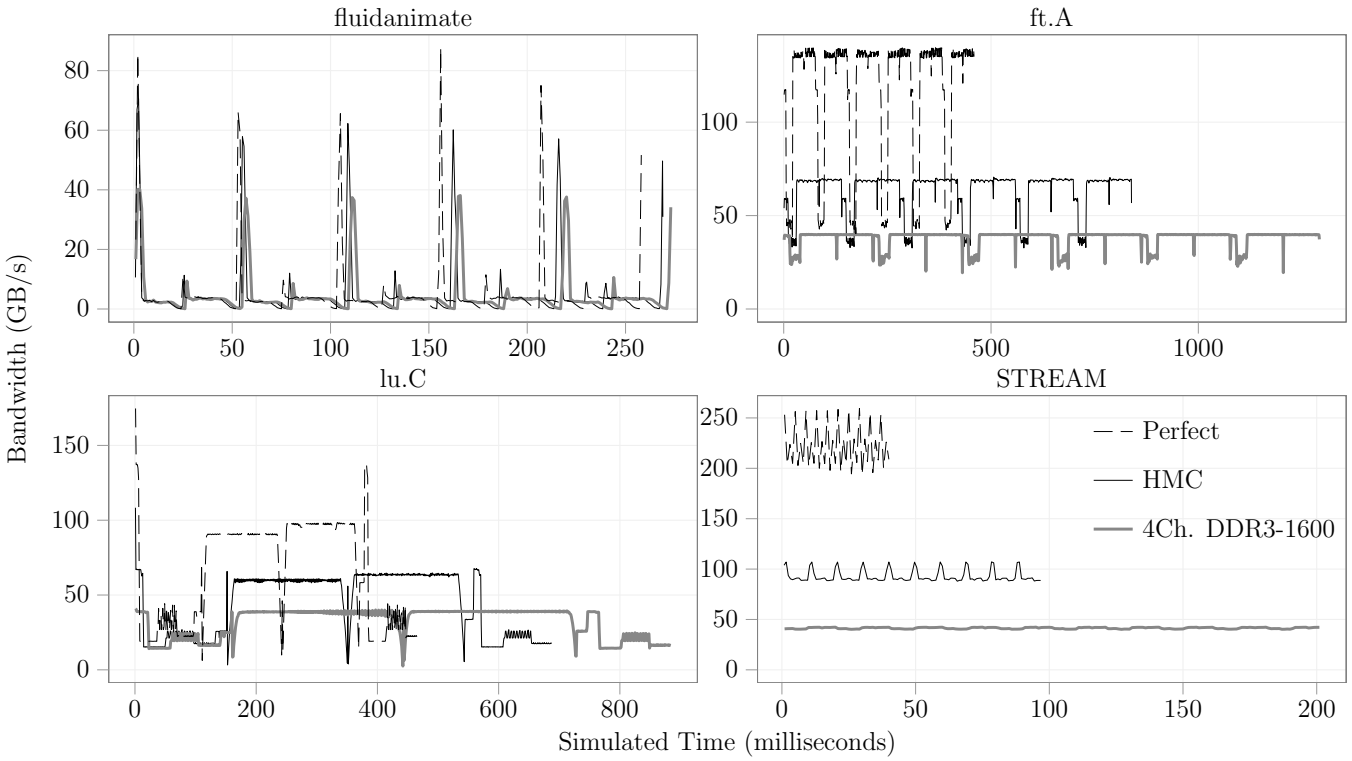

Figure 8: Bandwidth time series of the execution of several benchmarks with an HMC and an extremely aggressive DDR3-1600 configuration in a full system simulator between two identical points in the program. A shorter line corresponds to a lower execution time.

2.1x faster than with the quad channel DDR3-1600 memory system.

A similar pattern can be seen in the execution of ft.A (also shown in figure Figure 8). The iterations of ft.A can be seen in all three lines, and each iteration becomes shorter in time as the bandwidth of the memory system increases. However, even with a perfect memory system, ft.A can only generate 140 GB/s of memory bandwidth and thus is unable to see such a dramatic speedup from an improved memory system. Similarly, lu.C is unable make full use of extra memory bandwidth and achieves a modest speedup.

Finally, fluidanimate represents a workload that is memory intensive in very short bursts. Thus, even with a perfect memory system, it is unable to achieve any significant performance gains due to the memory system.

From this initial set of simulations we conclude that some truly memory intensive workloads stand to gain a significant performance boost when using the HMC while less intensive workloads will be able to maintain their performance at a lower energy per bit.

7. Related Work

7.1. DRAM on CPU Stacking

Much of the previous work on 3D stacked memory has focused on trying to put DRAM or SRAM stack directly on top of a CPU. Low level studies studies [4,6,28] attempt to characterize the design space of different stack organizations for power and performance. Several system level studies [3, 12, 14–16] show how stacking memory on top of the CPU can result in tremendous power and performance gains. However, since all of this work focuses on stacking the memory directly on top of the CPU, they explore a fundamentally different type of architecture than the one discussed in our work.

7.2. Serially Attached Stacked DRAM

Udipi, et al. [25] examine how the use of emerging silicon photonics can be efficiently utilized to connect a CPU to an off-chip 3D stacked memory. Unlike much of the other related work in this area, their main memory is not stacked directly on top of the CPU. They propose an interface die that sits below the memory dies and converts the photonic packet interface into electrical signals and handles the low-level memory scheduling details. They examine various configurations of photonic stops in order to most effectively amortize the photonics power costs. Finally, they propose an “unscheduled” interface policy between the CPU memory controller and the DRAM to try to reduce complexity.

They build on their previous work [26] that proposes to alleviate the “overfetch” problem (i.e., bringing an enormous number of bits into the row buffers but only using a tiny fraction of them) by making DRAM rows much shorter. One of

the biggest challenges they cited in this work was the lack of throughput between the smaller banks and the main memory controller. In their new work, the TSVs provide a low latency and high bandwidth path from the banks to the interface die, thereby eliminating the bottleneck. They are able to take advantage of the parallelism of a large number of banks through the usage of the TSVs.

Although the architecture presented in [25] is very similar to the HMC (high speed links providing an abstract memory interface connected to an interface die with multiple memory controllers), their work focuses more heavily on the photonics aspect of optimizing the architecture.

8. Conclusion

The Hybrid Memory Cube is an emerging memory technology with the potential to address the three main deficiencies of today's main memory systems by increasing capacity and bandwidth per core and lowering the energy cost per bit. Since the HMC represents a fairly radical departure from current DRAM designs, we have highlighted several interesting performance issues. First, we discussed the impact of packetized link interfaces and their influence on the selection link and TSV throughputs to account for the reduction in effective link bandwidth. Then, several performance effects were highlighted by using using several configurations with fixed resources to determine the most efficient cube organization (number of DRAM dies, memory controllers, banks, etc). We presented some performance characterizations that show that an HMC outperforms currently available main memory technologies for random stream workloads. Finally, we showed the performance of several multi-threaded benchmarks running in a multi-core full system simulation environment. We conclude that memory intensive workloads can achieve a non-trivial speedup while less memory intensive applications can gain more modest speedups, but all workloads can benefit from the HMC's power reduction.

References

[1] "Hybrid Memory Cube: Experimental DRAM," in *Intel Developer Forum*, 2011. [Online]. Available: http://download.intel.com/newsroom/kits/idf/2011_fall/pdfs/IDF_2011_Rattner_Presentation.pdf

[2] "Hybrid memory cube specification 1.0," Hybrid Memory Cube Consortium, Tech. Rep., 2013. [Online]. Available: http://www.hybridmemorycube.org/files/SiteDownloads/HMC_Specification%201_0.pdf

[3] B. Black, M. Annavaram, N. Brekelbaum, J. DeVale, L. Jiang, G. H. Loh, D. McCaule, P. Morrow, D. W. Nelson, D. Pantuso, P. Reed, J. Rupley, S. Shankar, J. Shen, and C. Webb, "Die Stacking (3D) Microarchitecture," in *Microarchitecture, 2006. MICRO-39. 39th Annual IEEE/ACM International Symposium on*, dec. 2006, pp. 469–479.

[4] K. Chen, S. Li, N. Muralimanohar, J. H. Ahn, J. B. Brockman, and N. P. Jouppi, "CACTI-3DD: Architecture-level modeling for 3D die-stacked DRAM main memory," in *Design, Automation Test in Europe Conference Exhibition (DATE), 2012*, march 2012, pp. 33–38.

[5] E. Cooper-Balis, P. Rosenfeld, and B. Jacob, "Buffer-on-board memory systems," in *Proceedings of the 39th annual International Symposium on Computer architecture (ISCA '12)*, June 2012.

[6] M. Facchini, T. Carlson, A. Vignon, M. Palkovic, F. Catthoor, W. Dehaene, L. Benini, and P. Marchal, "System-level power/performance evaluation of 3D stacked DRAMs for mobile applications," in *Design, Automation Test in Europe Conference Exhibition, 2009. DATE '09.*, april 2009, pp. 923–928.

[7] B. Ganesh, A. Jaleel, D. Wang, and B. Jacob, "Fully-buffered DIMM memory architectures: Understanding mechanisms, overheads and scaling," pp. 109–120, 2007.

[8] Inphi, "Introducing LRDIMM – A New Class of Memory Modules," Whitepaper, 2011.

[9] B. Jacob, S. W. Ng, and D. T. Wang, *Memory Systems: Cache, DRAM, Disk*. Morgan Kaufmann Pub, 2007.

[10] B. Jacob, "The memory system: You can't avoid it, you can't ignore it, you can't fake it," *Synthesis Lectures on Computer Architecture*, vol. 4, no. 1, pp. 1–77, 2009.

[11] JEDEC, "Main Memory: DDR3 & DDR4 SDRAM," http://www.jedec.org/category/technology-focus-area/main-memory-ddr3-ddr4-sdram.

[12] T. Kgil, S. D'Souza, A. Saidi, N. Binkert, R. Dreslinski, T. Mudge, S. Reinhardt, and K. Flautner, "PicoServer: using 3D stacking technology to enable a compact energy efficient chip multiprocessor," in *Proceedings of the 12th international conference on Architectural support for programming languages and operating systems*, ser. ASPLOS-XII. New York, NY, USA: ACM, 2006, pp. 117–128.

[13] T. Legler, "Choosing the DRAM with Complex System Considerations," presented at the Embedded Systems Conference, 2012.

[14] C. C. Liu, I. Ganusov, M. Burtscher, and S. Tiwari, "Bridging the processor-memory performance gap with 3D IC technology," *Design Test of Computers, IEEE*, vol. 22, no. 6, pp. 556–564, nov.-dec. 2005.

[15] G. H. Loh, "3D-Stacked Memory Architectures for Multi-core Processors," in *Proceedings of the 35th Annual International Symposium on Computer Architecture*, ser. ISCA '08. Washington, DC, USA: IEEE Computer Society, 2008, pp. 453–464.

[16] G. L. Loi, B. Agrawal, N. Srivastava, S.-C. Lin, T. Sherwood, and K. Banerjee, "A thermally-aware performance analysis of vertically integrated (3-D) processor-memory hierarchy," in *Proceedings of the 43rd annual Design Automation Conference*, ser. DAC '06. New York, NY, USA: ACM, 2006, pp. 991–996.

[17] Micron, "Revolutionary Advancements in Memory Performance ," http://www.youtube.com/watch?v=kaV2nZSkw8A.

[18] ——, "TN-47-21: FBDIMM – Channel Utilization (Bandwidth and Power) Scalable Power," Tech. Rep., 2006.

[19] L. Minas, "The Problem of Power Consumption in Servers," 2009. [Online]. Available: http://software.intel.com/sites/default/files/m/d/4/1/d/8/power_consumption.pdf

[20] A. Patel and F. Afram, "MARSSx86: Micro-ARchitectural and System Simulator for x86-based Systems," 2010. [Online]. Available: http://www.marss86.org/

[21] J. T. Pawlowski, "Hybrid Memory Cube (HMC)," in *Hot Chips 23*, August 2011. [Online]. Available: http://www.hotchips.org/wp-content/uploads/hc_archives/hc23/HC23.18.3-memory-FPGA/HC23.18.320-HybridCube-Pawlowski-Micron.pdf

[22] J. Rattner, "Intel Speaks about the Hybrid Memory Cube," http://www.youtube.com/watch?v=VMHgu_MKjkQ, 2011.

[23] P. Rosenfeld, E. Cooper-Balis, and B. Jacob, "DRAMSim2: A cycle accurate memory system simulator," *Computer Architecture Letters*, vol. 10, no. 1, pp. 16 –19, jan.-june 2011.

[24] G. Sandhu, "DRAM Scaling & Bandwidth Challenges," in *NSF Workshop on Emerging Technologies for Interconnects (WETI)*, 2012.

[25] A. N. Udipi, N. Muralimanohar, R. Balasubramonian, A. Davis, and N. P. Jouppi, "Combining memory and a controller with photonics through 3D-stacking to enable scalable and energy-efficient systems," in *Proceedings of the 38th annual international symposium on Computer architecture*, ser. ISCA '11. New York, NY, USA: ACM, 2011, pp. 425–436.

[26] A. N. Udipi, N. Muralimanohar, N. Chatterjee, R. Balasubramonian, A. Davis, and N. P. Jouppi, "Rethinking DRAM design and organization for energy-constrained multi-cores," in *Proceedings of the 37th annual international symposium on Computer architecture*, ser. ISCA '10. New York, NY, USA: ACM, 2010, pp. 175–186.

[27] D. Vantrease, R. Schreiber, M. Monchiero, M. McLaren, N. P. Jouppi, M. Fiorentino, A. Davis, N. Binkert, R. G. Beausoleil, and J. H. Ahn, "Corona: System Implications of Emerging Nanophotonic Technology," in *Proceedings of the 35th Annual International Symposium on Computer Architecture*, ser. ISCA '08. Washington, DC, USA: IEEE Computer Society, 2008, pp. 153–164.

[28] C. Weis, N. Wehn, L. Igor, and L. Benini, "Design space exploration for 3D-stacked DRAMs," in *Design, Automation Test in Europe Conference Exhibition (DATE), 2011*, march 2011, pp. 1–6.