Buffer-On-Board Memory System

Elliott Cooper-Balis, Paul Rosenfeld, Bruce Jacob
University of Maryland
{ecc17,prosenf1,blj}@umd.edu

Abstract

The design and implementation of the commodity memory architecture has resulted in significant performance and capacity limitations. To circumvent these limitations, designers and vendors have begun to place intermediate logic between the CPU and DRAM. This additional logic has two functions: to control the DRAM and to communicate with the CPU over a fast and narrow bus. The benefit provided by this logic is a reduction in pin-out to the memory system and increased signal integrity to the DRAM, allowing faster clock rates while maintaining capacity. The problem is that the few vendors utilizing this design have the same general approach, yet the implementations vary greatly in their non-trivial details.

A hardware verified simulation suite is developed to accurately model and evaluate the behavior of this buffer-on-board memory system. A study of this design space is used to determine optimal use of the resources involved. This includes DRAM and bus organization, queue storage, and mapping schemes. Various constraints based on implementation costs are placed on simulated configurations to confirm that these optimizations apply to viable systems. Finally, full system simulations are performed to better understand how this memory system interacts with an operating system executing an application with the goal of uncovering behaviors not present in simple limit-case simulations. When applying insights gleaned from these simulations, optimal performance can be achieved while still considering outside constraints (i.e., pin-out, power, and fabrication costs).

1. Introduction

The modern memory system has remained essentially the same for almost 15 years. Design decisions made in the past, when the disparity between the CPU and memory clock were not considered to be an issue, are now preventing the memory from providing the capacity and bandwidth that today's systems and applications demand. Modifications must be made to prevent the memory hierarchy from becoming an even greater bottleneck, further impeding performance gains in modern systems.

To support some level of memory system customization and expandability, commodity DRAM is purchased on a PCB called a dual in-line memory module (DIMM). Each DIMM uses physical contact (i.e., pins that plug into a motherboard slot) to provide electrical connectivity with the rest of the system. This physical contact is sufficient for electrical signals that operate at low speeds ($<$100MHz), but as the memory clock has increased to maintain pace with the CPU, this solution is proving to be less than ideal. The signal integrity at these physical contacts is greatly degraded as the memory clock is increased due to signal reflection and cross-talk. This issue is exacerbated as more DIMMs are placed in a channel and the further a particular DIMM is located from the memory controller's signal drivers [14, 12, 23].

As a result of these issues, when manufacturers increase the memory clock, they must reduce the number of DIMMs allowed in a channel to avoid extraneous costs of mitigating these signal integrity issues. This severely limits the total capacity supported in a system. For example, the original DDR SDRAM standard allowed four DIMMs in a channel, while DDR2 doubled speeds and reduced the number of DIMMs to two, and the higher-speed DDR3 variants (i.e., DDR3-1600) only allow a single DIMM of depth [12]. While it is possible to place higher capacity DIMMs in the channel, the overall rate of increase in capacity of a DIMM has slowed due to the difficulties in decreasing the size of a DRAM cell's capacitors. At the same time, the cost of these high-capacity DIMMs does not scale linearly with their capacity; instead, the cost per gigabyte is significantly greater.

The FB-DIMM memory system was originally introduced to solve these issues. Each FB-DIMM uses standard DDRx DRAM devices and has additional logic called the advanced memory buffer (AMB). The AMB allows each memory channel to operate on a fast and narrow bus by interpreting a new packetized protocol and issuing DRAM specific commands to the DRAM devices. Unfortunately, the high speed I/O on each AMB resulted in unacceptable

levels of heat and power dissipation [14, 2]. The inclusion of the AMB also resulted in more expensive DIMMs relative to similar capacity DDRx DIMMs. These issues ultimately led to the failure of the standard.

To fill the void left by FB-DIMM, vendors such as Intel, AMD, and IBM have devised new architectures to try and resolve the memory capacity and bandwidth issues. Although similar, this new architecture makes key changes which prevent the issues that plagued FB-DIMM: while each memory channel still operates on a fast, narrow bus, it contains a single logic chip as opposed to one logic chip per FB-DIMM. This allows the new architecture to use of existing low-cost DIMMs, prevents excessive power and heat in the logic, and reduces high variance in latency.

While this buffer-on-board memory system has already been implemented in a small number of high-end servers, the problem exists that each of these implementations differs in non-trivial details. The contribution of the present work is an examination of this new design space to determine optimal use of the resources involved and performance enhancing strategies. This includes proper bus configurations for various types of DRAM, necessary queue depths to reach peak DRAM efficiency, and address mappings to ensure an even request spread and to reduce resource conflicts.

2. Modern Memory Systems

Over the past five years, there have been numerous efforts to devise a next-generation memory system. While many ideas have been proposed, no clear solution has been widely adopted.

The most ubiquitous form of memory in use today is the JEDEC standardized double data-rate (DDR) synchronous DRAM. For the past 15 years, this has been the dominant form of commodity memory, eventually breaking into mobile and supercomputing markets due to an overwhelming abundance of parts. The widespread success is attributed to the standardization of the device packaging, pin-out, and operating protocol. The current generation (DDR3) can support transfer rates of up to a theoretical 12.8GB/s while operating at 1600Mbit/s [5].

The DDR SDRAM memory system operates on a 64-bit data bus that contains one or more ranks. A rank of memory is a group of DRAM devices that operate in synchrony by receiving and handling the same requests, at the same time. The memory controller, now typically located on the CPU die, issues DRAM-specific commands to the devices to retrieve and write data. This includes commands to sense data out of the DRAM array, read or write data, or refresh the bits in the array to prevent data loss from capacitive leakage. When reading or writing data out of the device, data is driven on both rising and falling edges of the memory clock, which is where the double data rate nomenclature arises.

The latest DDR SDRAM standard is the Load Reducing-DIMM (LR-DIMM) [15]. Similar to the Registered-DIMM (RDIMM) which buffers the control and address lines before reaching the DRAM devices, LR-DIMM places buffers on all of the signals between the CPU and DRAM devices. This includes the entire data bus, along with the control and address lines. These buffers help maintain signal integrity and circumvent some of the issues which result when increasing the memory clock. The end result is a system which can maintain acceptable capacity while still being able to provide the bandwidth modern systems require.

The FB-DIMM memory standard was introduced in 2004 with the intention of alleviating the problems with the current design. Each FB-DIMM uses the same DRAM devices as a DDR SDRAM DIMM, but operates on a faster, narrow bus. This was made possible by the inclusion of a small controller on each DIMM called the advanced memory buffer (AMB). The AMB interprets the packetized protocol that is now used over the narrow bus. It allows a much higher capacity (up to 768 GB per system) and significantly higher bandwidth per pin due to the increased clock rate.

An FB-DIMM memory channel operates on two separate logical buses: the northbound channel and the southbound channel [14]. These channels are different widths to account for the disparity between reads and writes, with the northbound channel (going toward the CPU) consisting of 14 data lanes and the southbound channel (going away from the CPU) consisting of 10 data lanes [4]. To account for such narrow buses, requests and responses are encapsulated in packets which are called frames. As these frames are sent on their respective channels, the AMB on each DIMM interprets the contents to determine proper routing or to generate standard DRAM commands for local DRAM devices.

Unfortunately, an unforeseen consequence of the architecture's point-to-point nature and the use of high speed I/O in each AMB caused unacceptable levels of heat and power dissipation. Under heavy load, a fully populated FB-DIMM channel requires over 100W to operate [3], on par with CPUs at the time. The AMB also increased the monetary cost of each DIMM; some FB-DIMM modules cost almost twice that of a similar capacity DDRx DIMM. Lastly, request latency was worse than DDRx systems as a result of serializing and interpreting frames in each AMB. Because of these issues, adoption of the standard slowed, and FB-DIMM was eventually removed from all major technology roadmaps. While no clear successor to FB-DIMM has been proposed, major vendors have taken it upon themselves to find solutions to the capacity and bandwidth issues.

IBM [6, 25], Intel [21], and AMD [19] have all proposed or implemented a similar memory architecture to fill the void that was left when FB-DIMM was abandoned. Like FB-DIMM, these new architectures place logic between the

DRAM and the CPU that is responsible for both controlling the DRAM and communicating with the CPU over a relatively faster and narrow bus. Representations of each of these architectures and key specifics can be seen in **Figure 1**.

While each of these memory systems has the same approach to solving the issues described above, they are vastly different in specific details. This includes the width and speed of the narrow buses now used in each channel, the number of ranks of memory per channel, the type of DRAM, and the total number of channels per processor or core. For example, IBM's Power 795 memory system has eight logically-separate memory channels each composed of 8 data lanes for requests and 16 data lanes for responses. These buses are used to communicate with an "advanced buffer chip" [6], each of which are attached to only a single rank of DDR3-1066 DRAM. On the other hand, Intel's Scalable Memory Interface (SMI) uses buses which are 9 and 12 data lanes for requests and responses, respectively. The logic chip, called the Scalable Memory Buffer (SMB), is responsible for controlling two separate DDR3 channels (of various speeds), each with two ranks. Lastly, AMD's G3 Memory Extender (G3MX) was designed to use a bus of 13 data lanes for requests and 20 data lanes for responses with four ranks of DDR3 attached to each G3MX but was canceled in 2008 [18].

Clearly, these systems are vastly different while still trying to accomplish the same goal with the same approach. The discrepancies in these designs dictate the necessity for a design space examination with the goal of providing optimal use of all resources involved. Before this can be done, a generalized view of the architecture must be defined.

3. Buffer-On-Board Memory System

The *buffer-on-board* (BOB) memory systems seen in **Figure 1** can be seen as specific implementations of the generalized model seen in **Figure 2**. This model consists of multiple DRAM channels populated with commodity LR-, R-, or U-DIMMs. Each of these *BOB Channels* could be considered identical to a regular, JEDEC-standardized memory system. The control and data bus, operating protocol, and timing constraints are the same ones used in a normal memory system. These channels are each controlled by a *simple controller* which is the intermediate logic located between the DRAM and the main, on-die memory controller. The simple controller is also responsible receiving requests and returning data back to the main memory controller (as opposed to the DRAM communicating directly to the main memory controller).

Communication between the simple controller and the CPU occurs over a *link bus* which is narrower and faster than the DRAM bus which communicates with the DIMMs.

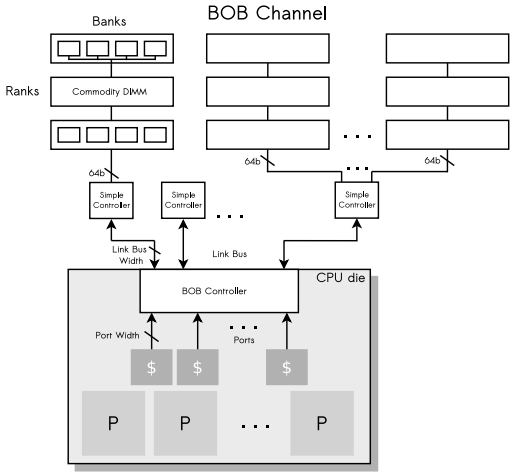
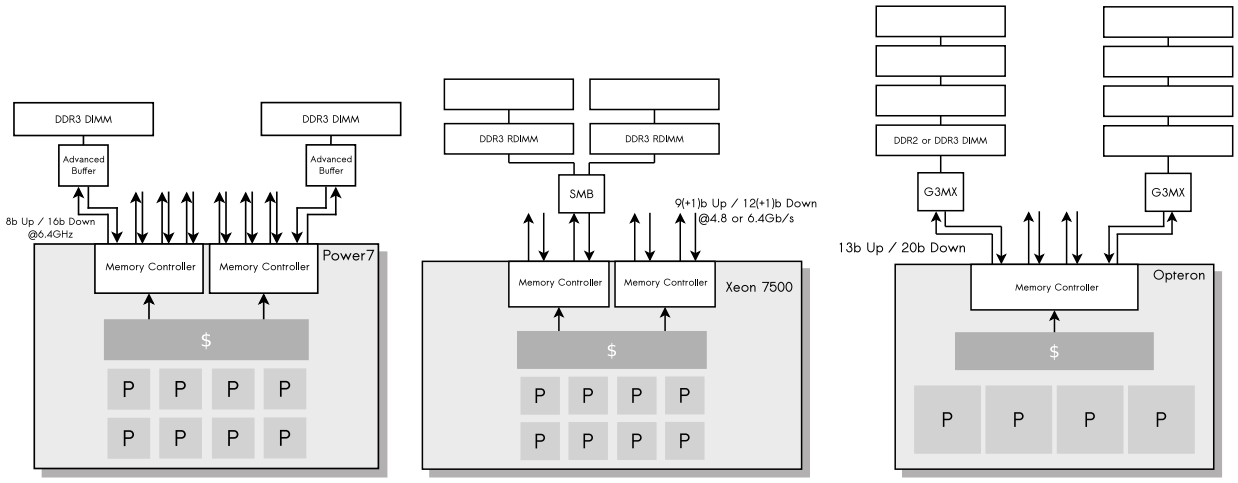


Figure 2. The BOB memory system architecture

Unlike the DRAM bus, which has separate control and data signals, the lanes which comprise a link bus are for general purpose communication. The full–duplex link bus is comprised of request (towards the DRAM) and response (towards the CPU) data lanes which may be different widths and operate at some speed faster than the DRAM.

Request packets sent over a link bus must contain the address, the request type, and the data if the request is a write. Upon receiving a request packet, the simple controller must translate the contents into a series of DRAM specific commands (i.e., ACTIVATE, READ, WRITE, etc.) which are issued to the ranks attached to the DRAM bus. A *command queue* is responsible for storing requests in need of scheduling. Once data has been received from the DRAM data bus, the *read return queue* is responsible for storing data before it can be serialized and sent on the response link bus. Each *response packet* will contain data as well as the address of the initial request for identification purposes. This is necessary due to requests being scheduled out of order both within the BOB controller and simple controller, and may be completed at different times. The total size of these packets is important for the generalized model as it determines the amount of time it takes a request or response packet to travel on the link bus.

The main *BOB controller* which resides on the CPU die is another essential aspect of the architecture. Aside from the typical functionality of a memory controller, such as address mapping and returning data to the cache, the BOB controller is also responsible for packetizing requests and interpreting response packets sent to and from the simple controllers over the narrow link bus. Since the link bus is narrower than the DRAM bus, requests and responses must be encapsulated within a packet which is sent over the



(a) IBM Power 795 memory system (b) Intel SMI/SMB memory system (c) AMD G3MX memory system

Figure 1. Example buffer-on-board memory systems

link bus during multiple clock cycles. This is accomplished with a serialize-deserialize (*SerDes*) interface and associated buffer for the request and response path of each link bus. Communication with the cache and CPU is executed over *ports*, which are logically separate, full-duplex lanes. A cross-bar switch ensures that a request from any port is capable of being sent to any link bus.

4. BOB Simulation Suite

To properly evaluate this new architecture, a simulation suite is developed with a strong focus on hardware verification and comprehensive, detailed system modeling. Two separate simulators are used in this suite: a BOB memory system simulator developed by the authors and MARSSx86 [16], a multi-core x86 simulator developed at SUNY-Binghamton. Together, they create an accurate model of a processor which boots an operating system, launches an application, and interacts with the cache and memory system.

The BOB memory system model is a cycle-based simulator written in C++ that encapsulates the main BOB controller, each BOB channel, and their associated link bus and simple controller. Each of the major logical portions of the design have a corresponding software object and associated parameters that give total control over all aspects of the system's configuration and behavior. Some simple examples include the type of DIMMs and number of ranks within an individual BOB channel, the total number of BOB channels, queue depths, or speed and width of each link bus. The BOB simulator may be run in one of two modes – a stand-alone mode where requests from a parameterizable, random address generator or trace file are issued directly to the memory system or a full-system simulation mode where the BOB simulator receives requests from MARSSx86.

MARSSx86 merges the highly detailed, out-of-order x86 pipeline models from PTLSim [24] with the QEMU emulator [8]. MARSSx86 augments the original PTLSim models with multi-core simulation capability and a malleable coherent cache hierarchy. The ability to simulate multi-core environments is critical since multithreaded workloads are quickly becoming the rule rather than the exception. Additionally, it is hard to imagine a single threaded application being able to take full advantage of the tremendous bandwidth provided by the BOB memory system. MARSSx86 also provides full system simulation, which allows the simulator to capture the effects of virtual memory and kernel interaction. The CPU models are highly configurable and make it relatively easy to add a memory system simulator. It is possible to change the internals of the CPU or behavior of caches to take advantage of new features (for example, replacing a traditional memory bus with a number of ports).

Another important aspect of the framework is its ability to validate its behavior against that of actual hardware. Since the DIMMs used in a BOB memory system utilize the same DRAM devices as those in a commodity system, this portion of the simulator is validated in the same manner as DRAMSim2 [17]. Micron Technology publicly provides Verilog HDL models for each of the DRAM devices that it produces. These models determine whether or not a timing constraint has been violated based on a series of inputs from a hardware behavioral simulator like ModelSim. During a BOB simulation, each DRAM channel produces a bus trace file which is used in conjunction with ModelSim and these Micron HDL models to ensure the timing of both commands and data are cycle accurate at the DRAM

level. The BOB simulator uses a DDR3-1066 device (MT41J512M4-187E), a DDR3-1333 device (MT41J1G4-15E), and a DDR3-1600 device (MT41J256M4-125E), yet any JEDEC standard device will work. For further details on this verification process see [17].

5. Simulation Results

When evaluating the characteristics and behavior of this new architecture, we performed two experiments: a limit-case simulation where a random address stream is issued into a BOB memory system whenever resources permit (i.e., as fast as possible) and a full system simulation where an operating system is booted on an x86 processor and applications are executed. The limit-case study is useful for identifying the achievable maximum sustained bandwidth and the behavior of the system under heavy stress. Many server and HPC applications generate address streams that have little locality (temporal or spatial) and appear random. In contrast, a full system simulation gives a much more detailed picture of the new memory system's interaction with the cache and processor, operating system, and applications. For benchmarks, we used the NAS parallel benchmarks, the PARSEC benchmark suite [9], and *STREAM*. We emphasized multi-threaded applications to demonstrate the types of workloads this memory architecture is likely to encounter. For design tradeoffs we included implementation costs such as total pin count, power dissipation, and physical space (or total DIMM count).

5.1. Limit-Case Simulations

For the limit-case simulations, the BOB simulator is run in a stand-alone mode where memory transactions are issued from a generated request stream that can be tailored to issue at a specific frequency or read-write ratio. For these simulations, requests are issued as soon as resources are available and with a mix of 2/3 reads and 1/3 writes.

5.1.1. Simple Controller & DRAM Efficiency

Commodity memory system design has been examined and analyzed extensively [10, 14, 22, 11]. Since each BOB channel uses commodity DIMMs, operates using the same data and command bus, and requires the same operating protocol and timing constraints, it stands to reason that the previous insights, optimizations and analysis targeting commodity systems should apply here as well; simulations confirm this. For example, optimal rank depth for each DRAM channel is between two and four for all speed-grades. One rank does not provide enough parallelism and more than four ranks results in a drop in DRAM bus efficiency as the bus must be idled more frequently. The peak efficiency achieved by the DRAM bus in each BOB channel is also verified by manufacturers results [1] and prior research [20]. Address mapping is also an essential factor in the resulting DRAM efficiency [22] but is explored later under the full-system simulation.

The simple controller must also have features that differentiate it from a commodity memory controller as a result of serializing communication on the link bus. The read return queue within each simple controller is responsible for storing requested data before it is packetized and transferred out on the response link bus back to the main BOB controller. If this queue is full, no further read or write commands will be issued to the DRAM until there is space within this queue. This is necessary to guarantee data will not be lost when it is returned from the DRAM devices.

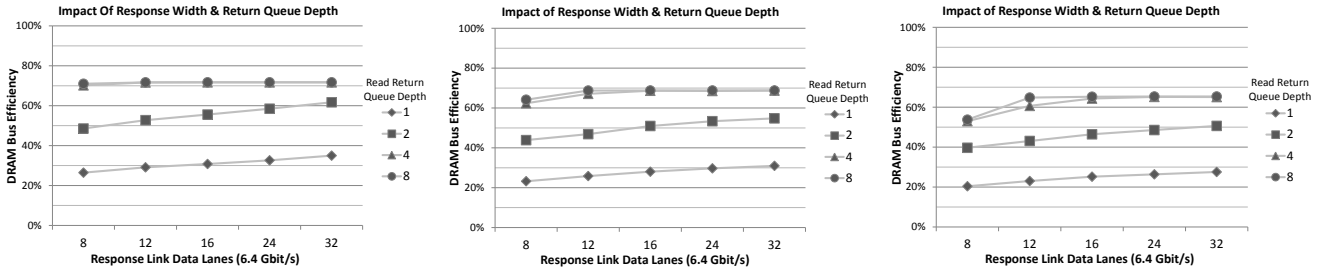
The rate at which items are removed from this queue is determined by both the width and speed of the response link bus. A parameter sweep is performed on both the depth of the read return queue and the configuration of the response link bus to detail the impact these decisions have on the achievable efficiency of different speeds of DRAM. The results can be seen in can be seen in **Figure 3**. Even though the response link bus is a factor in some cases, in order to reach peak DRAM efficiency, a read return queue must have at least enough capacity for four responses packets. With a depth of four, the response link bus is the determining factor in whether or not the DRAM reaches this peak efficiently; this is discussed below.

5.1.2. Link Bus Configuration

The efficiency of each DRAM channel is inherently linked with the overall performance of a BOB memory system. Therefore, optimal system configurations are ones in which the request link bus and response link bus do not negatively impact the DRAM efficiency. The width and speed of these buses should be configured such that request and response packets can be sent at a rate that does not stall the DRAM, either due to a lack of available requests issuable to the DRAM or due to an inability to clear the read return queue quickly enough.

An accepted rule-of-thumb is that a typical request stream will have a read-to-write request ratio of approximately 2-to-1. Implemented systems have accounted for this fact by weighting response paths more than requests paths. This can be seen starting from the FB-DIMM standard, which had the northbound bus (for responses) 40% larger than the southbound bus (for requests) [4]. The new architectures detailed above adopt this convention as well with Intel's SMI response bus 33% larger [7] than the request bus, and IBM's Power7 system whose response bus is twice as wide as the request bus [6].

Limit-case simulations are performed with eight DRAM channels attached to various request and response link bus



(a) DDR3-1066 (b) DDR3-1333 (c) DDR3-1600

Figure 3. Impact of read return queue depth and response bus width on DRAM bus efficiency

configurations. The results can be seen in **Figure 4**. The simulation results show a clear peak bandwidth where additional link width or speed has no impact on the overall performance. When each DRAM channel is capable of reaching peak efficiency, this peak bandwidth is simply a product of the achievable bandwidth and the total number of DRAM channels. To prevent the request and response link bus from having a negative impact on DRAM performance, each must be capable of meeting bandwidth requirements dictated by the DRAM and request stream.

These requirements have several factors. First, the achievable DRAM bandwidth determined by the speed grade and the expected efficiency of that device is clearly an important factor in how each link bus should be configured. Second, like FB-DIMM [13], the read-write ratio has a significant impact on the utilization of each link bus. Lastly, because the link buses are responsible for transmitting packets and packet overhead as well as data, this must be accounted for as well. Incorporating all of these factors together results in **Equations 1 & 2**; these equations dictate the bandwidth required by each link bus to prevent them from negatively impacting the efficiency of each channel. In **Figure 4**, link bus configurations which have a bandwidth equal to or greater to the values dictated by these equations are capable of achieving the peak possible bandwidth for the simulated system.

The unidirectional nature of each link bus causes sensitivity to the read-write request mix similar to that seen in FB-DIMM [13]. While weighting the response link bus more than the request link bus might be ideal for many application request streams, performance will be significantly different as soon as the request mix changes. **Figure 5** shows the impact of different read-write request ratios on weighted and unweighted link bus configurations during limit-case simulations. Intuitively, when the request mix is weighted in the same fashion as the link buses, the DRAM can reach peak efficiency. Unfortunately, this behavior is an unavoidable side-effect of serializing the communication on unidirectional buses and a decision should be made based on the most likely workload the memory system will see.

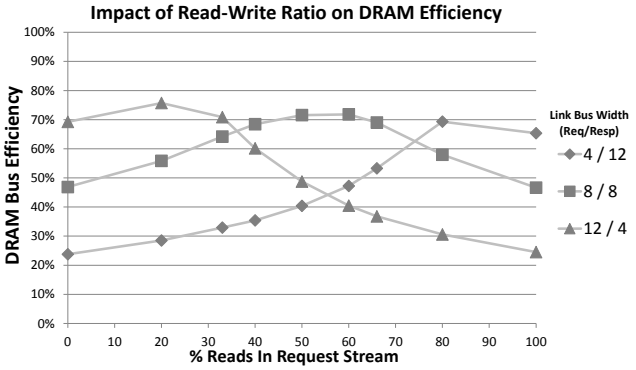


Figure 5. Impact of read-write ratio on different weighted link bus configurations

5.1.3. Multi-Channel Optimization

If the link bus configurations provide bandwidth that can not be fully utilized by a single logical channel of DRAM, it is possible for multiple logically independent channels of DRAM to share the same link bus and simple controller without negatively impacting performance. This will reduce costs such as pin-out, logic fabrication, and physical space. The pin-out of the CPU can be reduced for an equivalent number of DRAM channels since fewer link buses would be required. This will also reduce the number of simple controllers, which will reduce fabrication costs and the physical space necessary to place them on the motherboard. While reducing these costs, it is important to note that complexity of the simple controller will increase at the same time. The pin-out of the simple controller must be increased to support multiple DRAM channels and the logic within must be replicated for each of the logically independent channels (which will in turn increase the power requirements of each controller). An example of this optimization can be seen in Intel's SMI/SMB architecture – each SMB supports two separate channels of DDR3.

The link bus bandwidth requirements defined by **Equa-**

$$BW_{Request} = (BW_{DRAM} \times Efficiency) \times \left[\%Writes \times \left(1 + \frac{WritePacketSize}{RequestSize}\right) + \%Reads \times \frac{ReadPacketSize}{RequestSize}\right] \quad (1)$$

$$BW_{Response} = (BW_{DRAM} \times Efficiency) \times \left[\%Reads \times \left(1 + \frac{ResponsePacketSize}{RequestSize}\right)\right] \quad (2)$$

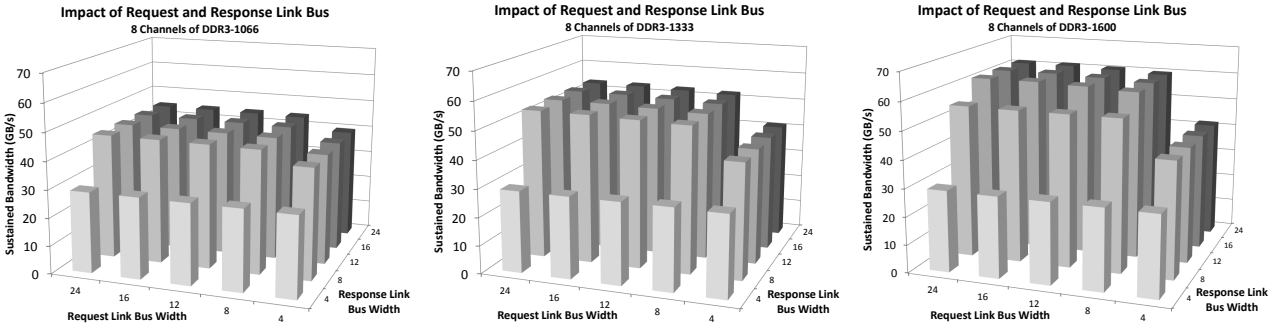


Figure 4. Sustained bandwidth of eight BOB channels using various link bus configurations

tions 1 & 2 can be modified to account for this optimization:

$$BW_{Request}{}' = (NumDRAMChannels) \times BW_{Request} \quad (3)$$

$$BW_{Response}{}' = (NumDRAMChannels) \times BW_{Response} \quad (4)$$

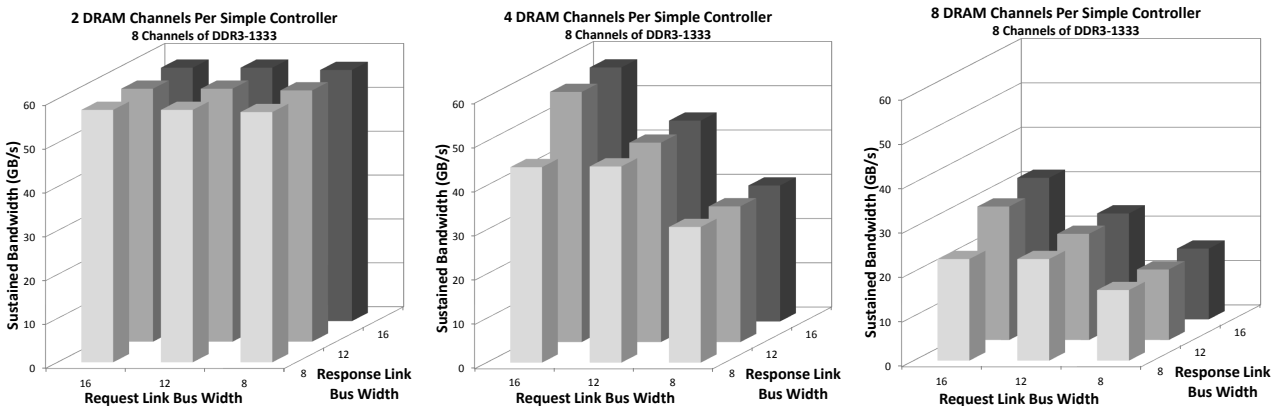
The number of DRAM channels which can be supported by a single simple controller and link bus is determined by the available bandwidth of the request and response link bus. If this available bandwidth meets the bandwidth computed in **Equations 3 & 4**, performance will not be negatively impacted. **Figure 6** shows the sustained aggregate bandwidth of 8 DDR3-1333 channels as the number of DRAM channels per simple controller is increased. With two DRAM channels sharing a link bus and simple controller, there is minimal performance impact. This configuration uses half the number of simple controllers and CPU pins, which significantly reduces system costs. The lack of available bandwidth of each link bus becomes an issue once four DRAM channels share the same simple controller and respective link bus. Using **Equations 3 & 4**, four DRAM channels of DDR3-1333 need 24.4 GB/s response bandwidth and 13.7 GB/s request bandwidth in order to operate at maximum efficiency (70%); this makes it difficult for DRAM channels in a 4-to-1 configuration to reach peak efficiency. Once eight DRAM channels are sharing a single simple controller, the response link bus is utilized over 99% of the time, causing performance to drop significantly.

5.1.4. Cost Constrained Simulations

When implementing an actual system, costs such as the required CPU pin-out, power, physical space, and the monetary cost of the DIMMs are all important aspects that need to be considered. So far simulations have not taken these constraints into account, instead aiming for a general overview of the system's behavior by exploring the design space outside of what might actually be feasible. To ensure that the insights gained from these simulations actually apply to real–world situations, constraints should be placed on various dimensions of the design space. For example, one constraint could be the total number of CPU pins allotted to communicate with the memory system. The total number of CPU pins is a significant portion of the fabrication cost, so optimal use of the ones available is critical. A fixed number of pins can be configured as link buses in numerous ways, from a few wide buses to numerous narrow ones. Another example constraint could be a maximum number of DIMMs allowed in a single system, either due to physical space, monetary, or power limitations.

Intel's SMB is used to determine some of the other costs involved with this architecture. In an Intel-based system, the SMB dissipates 7 watts idle and up to 14 watts under load [7]. Within the simulator, these values are used to determine system power. Each simple controller consumes 7 watts of background power and an additional 3.5 watts for each DRAM channel it controls (i.e., a simple controller which controls four DRAM channels will consume 7 watts background power and 14 watts for the channels it operates, totaling 21 watts under load). The pin-out of the SMB is also used to determine the appropriate number of pins required to control a channel of DRAM. Of the 655 pins on each SMB package, 147 are used to control a single channel of DRAM. Therefore, when determining the pin cost of a simple controller in the simulator, a multiple of 147 is used depending on the number of DRAM channels.

Further limit-case simulations are performed with outside constraints placed on aspects of the BOB system. In this system, eight DRAM channels, each with four ranks (32 DIMMs making 256 GB total) are allowed while the



(a) 2-to-1 Ratio (b) 4-to-1 Ratio (c) 8-to-1 Ratio

Figure 6. Sustained bandwidth during limit-case simulations of eight DDR3-1333 channels with varying degrees of multi-channel utilization

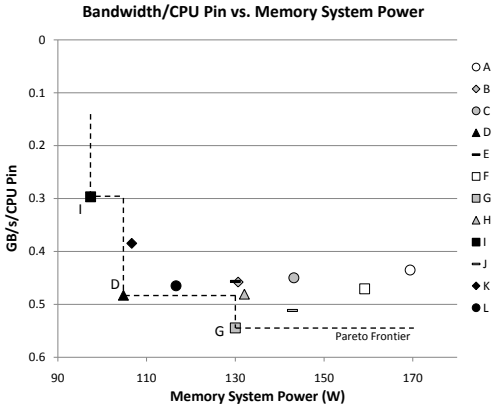


Figure 7. Pareto frontier analysis plot of configurations in Table 1

CPU has up to 128 pins which can be used for data lanes to comprise various link buses; these lanes are operated at 3.2 GHz (6.4 Gb/s). The theoretical peak of this system is 85.333 GB/s (eight channels of DDR3-1333 whose theoretical peak bandwidth is 10.666 GB/s each). Even with these constraints, there are still numerous ways to configure a BOB memory system. Some of these possibilities (**Table 1**) are simulated and the and results can be seen in **Figure 7**. **Note** : Y-Axis in **Figure 7** is inverted.

To perform a fair Pareto frontier analysis, relevant costs must be incorporated into the data. To do this, the sustained bandwidth is normalized against the total number of CPU pins that are utilized, since some configurations do not utilize all 128 pins. The color of each data point corresponds with the relative complexity of the simple controller, where the black points are configurations where the simple controller requires 588 pins (for four DRAM channels), the gray points require 294 pins, and the white points require 147. It is clear from **Figure 7** that some configurations of the available resources (DRAM and pins) are more desirable than others. The Pareto frontier analysis dictates that configurations *D*, *G*, and *I* are Pareto equivalent and the most optimal for the parameters tested. Since the simple controller complexity is not accounted for in this analysis, there is still a decision to be made about which configuration is best suited for a particular situation. If raw performance or a less complex simple controller is more desirable, then *G* is better suited, yet if system power consumption is a concern, *D* and *I* are more ideal. This analysis also clearly shows the downside to configurations where a simple controller drives only a single channel of DRAM (white points). In this situation, the power dissipation from having 8 separate simple controllers far exceeds that of the other configurations with no benefit in performance.

In conclusion, the limit-case simulations above have provided the following insights: (1) Optimal rank depth for each BOB channel is between two and four, similar to commodity systems; (2) The read return queue in each simple controller must have capacity for at least four responses to prevent the DRAM from stalling, thus reducing efficiency; (3) The request and response link bus must have adequate bandwidth dictated by **Equations 1 & 2** to ensure DRAM performance is unhindered; (4) Read-write sensitivity is shown to have an impact on DRAM efficiency in a manner similar to FB-DIMM; (5) Implementation costs can be reduced by implementing the multi-channel utilization, which can operate optimally when each link bus adheres to **Equations 3 & 4**.

Config Name	Request Bus Width	Response Bus Width	DRAM : Simp. Controller	Simp. Controller Pin-Out	# Of Simp. Controller	CPU Data Lanes
A	8	8	1:1	147	8	128
B	12	12	2:1	294	4	96
C	16	16	2:1	294	4	128
D	16	16	4:1	588	2	64
E	32	32	4:1	588	2	128
F	4	8	1:1	147	8	96
G	8	12	2:1	294	4	80
H	8	16	2:1	294	4	96
I	8	32	4:1	588	2	80
J	12	16	2:1	294	4	112
K	12	32	4:1	588	2	88
L	16	32	4:1	588	2	96

Table 1. Configuration parameters for various tested systems. Optimal configurations highlighted

5.2. Full System Simulations

The BOB configurations which were determined to be Pareto optimal in **Figure 7** are used within full system simulations. MARSSx86 is configured with 8 out-of-order cores running at 3.2 GHz with 4 MB LLC. The operating system used is Ubunutu 9.10 with Linux kernel 2.6.31. *STREAM* is executed for 10 iterations using a 2 million element array size. PARSEC benchmarks are executed using "simlarge" data sets. NAS benchmarks are run using "C"-sized datasets. The *mcol* workload walks a 64 MB matrix row-by-row. The impact of the memory system on the execution time of these benchmarks will be visible with a full system simulation. This is possible because the MARSSx86 CPU model will stall thread execution when waiting for pending memory transactions.

5.2.1. Performance & Power Trade-offs

Statistics about the execution of each benchmark can be seen in **Table 2**. These values provide a clearer picture of benefits and drawbacks of each system and are necessary to account for the achieved performance and execution time of each benchmark when comparing configurations. Each BOB configuration achieves the best relative performance during at least one benchmark, showing that the application is the ultimate determining factor in the performance of the memory system.

STREAM and *mcol* generate the greatest average bandwidth among the benchmarks which are executed, yet the behavior of the memory system during these benchmarks is drastically different. These benchmarks achieve significantly different performance from each of the BOB configurations, yet the best performing configuration is different. This is due to the request mix generated during the region of interest; the *STREAM* benchmark generates a request stream of 46% reads and 54% writes while the *mcol* benchmark issues 99% reads. The relatively balanced request ratio of *STREAM* favors the parallelism and relatively balanced link-bus widths of configuration *G*, whose execution time is 2.9% less than *D* and 49% less than *I*. *I*'s performance is significantly worse due to the inability of the request link bus to provide the DRAM channels with requests at a sufficient rate. Conversely, during *mcol*, *I* performs significantly better than both *D* and *G*; this is a result of the wide response link bus in *I*, which can easily handle the inordinate amount of read requests during this benchmark. The execution time when using *I* is 15.6% less than *G* and 36.6% less than *D*.

For benchmarks where the memory system is largely idle, the average bandwidth and execution time are relatively similar; therefore, comparing performance provides little insight. When the memory system is not heavily utilized, the power dissipation and energy play a much larger factor in determining the best configuration. For example, the execution time of *fluidanimate* differs by less than 6% across all systems, yet the energy per bit consumed within configuration *G* and is 16% greater than *D* (which has the least). The increased energy consumption is a result of a greater number of simple controllers – four, whereas *D* and *I* only have two. A greater number of simple controllers is a benefit during some memory intensive benchmarks, but when the memory system is mostly idle, the increased power consumption becomes a detrimental factor.

5.2.2. Address & Channel Mapping

In a BOB memory system, address mapping occurs in two separate but equally important places – within the main BOB controller and within each simple controller. The mapping that occurs in the BOB controller takes a portion of the physical address to determine which channel should receive that request. This process is essential to evenly spread out requests over all available channels. An imbalanced mapping will overload a particular channel and cause contention on the link buses and subsequently within the DRAM. The mapping within each simple controller is similar to commodity systems and takes portions of the physi-

	STREAM					mcol				
Conf.	Power (W)	GB/s	Time(ms)	W / GB/s	Energy(j)	Power (W)	GB/s	Time(ms)	W / GB/s	Energy(j)
D	105.7	32.61	117.9	**3.24**	**12.5**	94.9	21.33	148.8	**4.45**	**14.1**
G	120.5	33.54	114.5	**3.59**	**13.8**	114.1	27.13	111.7	**4.20**	**12.8**
I	91.4	17.10	224.5	**5.34**	**20.5**	104.7	32.34	94.3	**3.23**	**9.9**
	facesim					**mg**				
Conf.	Power (W)	GB/s	Time(ms)	W / GB/s	Energy(j)	Power (W)	GB/s	Time(ms)	W / GB/s	Energy(j)
D	78.5	3.06	1419	**25.65**	**124.3**	93.1	18.87	3782	**4.93**	**352.5**
G	92.5	3.11	1438	**29.74**	**146.0**	101.8	18.35	3887	**5.55**	**395.8**
I	78.4	2.92	1540	**26.84**	**133.5**	92.9	17.13	4162	**5.43**	**377.6**
	fluidanimate					**sp**				
Conf.	Power (W)	GB/s	Time(ms)	W / GB/s	Energy(j)	Power (W)	GB/s	Time(ms)	W / GB/s	Energy(j)
D	78.5	3.06	508.4	**25.65**	**40**	97	23.27	1380	**4.17**	**133.9**
G	92.5	3.11	500	**29.74**	**46.3**	111.2	23.43	1367	**4.74**	**151.9**
I	78.4	2.92	531.6	**26.84**	**41.7**	93	18.86	1698	**4.93**	**157.9**

Table 2. Statistics of benchmarks during full system simulation using optimal BOB configurations

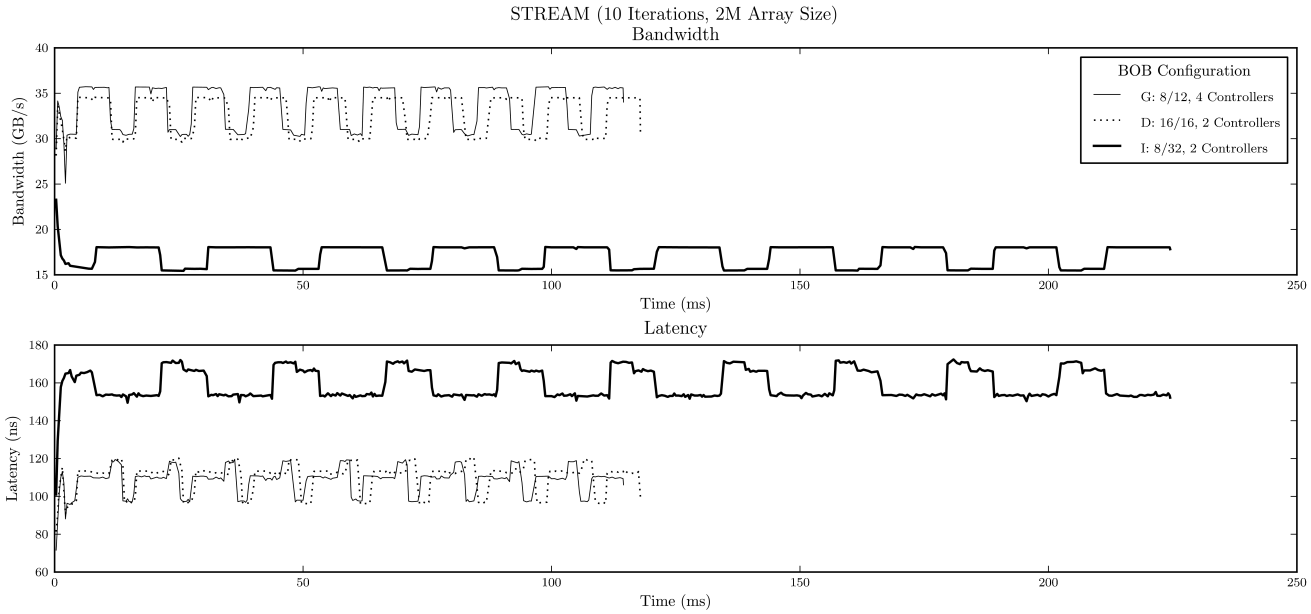


Figure 8. Full system simulations running the *STREAM* benchmark

cal address to determine which resources within the DRAM channel are used to satisfy a request; this includes the rank, bank, row, and column.

During limit-case simulations, address and channel mappings provide little insight due to the random nature of the address stream. During a full-system simulation this is no longer the case and these important aspects of a BOB memory system can be explored. Such behavior can be seen in **Figure 9** where various channel mapping schemes used during the execution of the PARSEC benchmark *facesim* result in widely different channel utilization. The request spread has a significant impact on the overall execution of the benchmark (**Figure 10**). The ideal channel mapping (RW:BK:RK:CLH:CH:CLL:BY) results in an execution time of 425.9 ms, which is almost half the runtime relative to the worst channel mapping scheme (CH:RW:BK:RK:CL:BY), whose execution time was 750.3 ms. The lower the order of the bits that are used to determine the channel address, the better the performance. This is because lower order bits flip more frequently than higher order bits and are more likely to evenly spread out requests across all channels. (**Note :** CH:Channel, RK:Rank, BK:Bank, RW:Row, CL:Column, CLH:Column High, CLL:Column Low, BY:Byte Offset)

With the optimal bits determined for mapping the channel address, the DRAM mapping can be examined. This mapping is identical to the address mapping in commodity memory systems; portions of the physical address are used to determine which DRAM resources should be used. As with the channel mapping, the likelihood of a particular bit

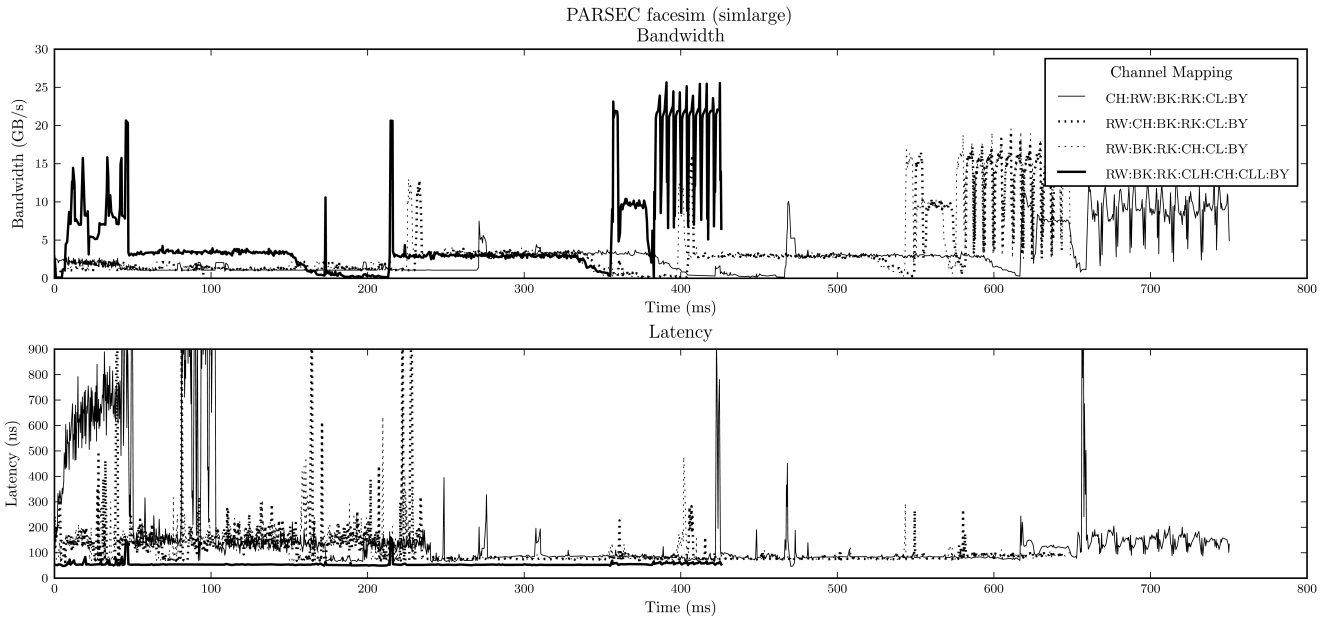


Figure 10. Various channel mapping during full system simulation of configuration G running *facesim*

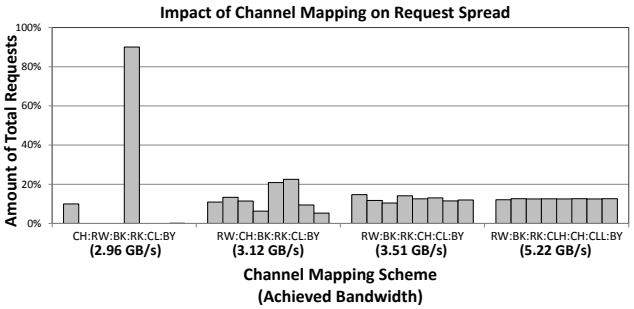


Figure 9. Request spread over all channels in *G* **resulting from address mapping schemes while executing PARSEC** *facesim***.**

flipping is a key aspect in how that bit should be used to map resources. To properly utilize the parallelism within a channel and DRAM device, bits that flip more frequently should be used to map resources of greater parallelism within the system. An example of this can be seen during the execution of NAS benchmark *sp* (**Figure 11**). Address mapping schemes that map the rank and bank with lower-order bits (which flip more frequently) achieve greater performance (up to an average of 25 GB/). The worst performing DRAM mapping scheme only achieves 34% of the bandwidth of the most optimal scheme and shows that even when requests are evenly spread over all channels with an optimal channel mapping, the DRAM mapping scheme still has a significant impact on performance.

6 Conclusion

Limitations in the commodity memory system have forced system designers to implement a new architecture which allows an increase in both speed and capacity. They have achieved this by placing intermediate logic between the CPU and DIMMs, thereby increasing signal integrity. This architecture has been implemented in HPC and server systems, yet non-trivial details about each system varies greatly. A cycle accurate and hardware verified simulator was developed to characterize the behavior of this new type of memory system and to determine optimal use of the available resources. This includes necessary queue depths to reach peak DRAM channel efficiency, proper bus configurations, and address mappings which utilize all the parallelism within the system. Cost-constrained simulations also provided evidence of similar performance between vastly different configurations, proving outside constraints can still be considered without sacrificing performance. While many issues still face the modern memory system, the buffer-on-board architecture provides many features and benefits that improve a system's capacity and performance, making it an ideal near-term solution.

References

[1] DDR3 Power Estimates, Affect of Bandwidth, and Compar-

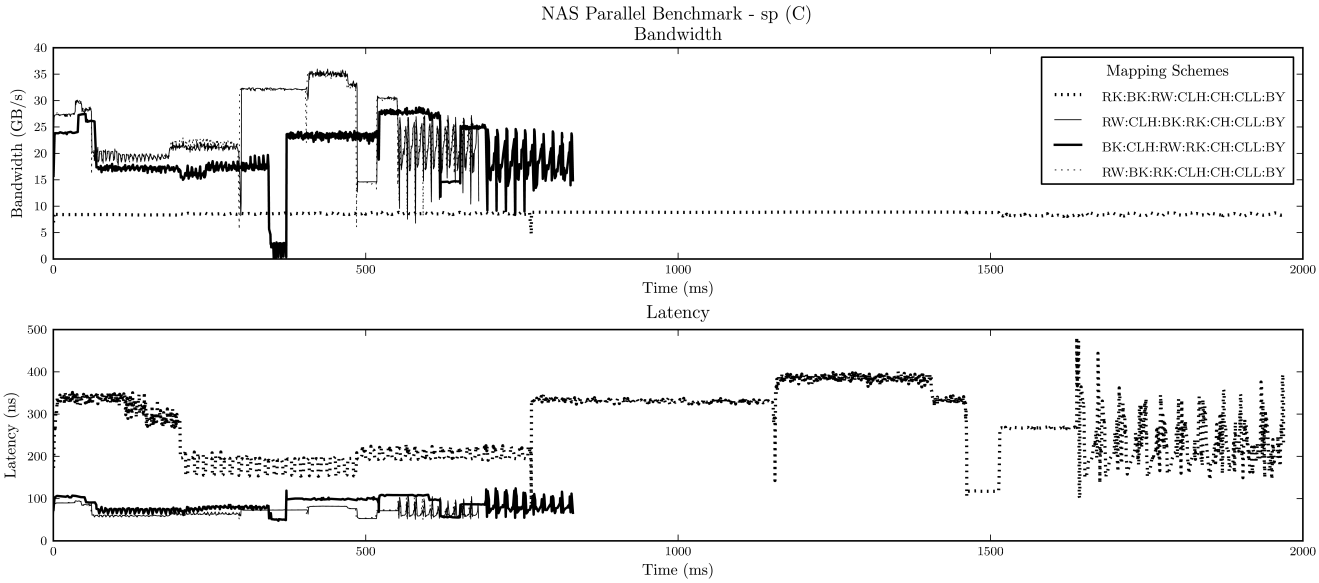


Figure 11. DRAM mapping during full system simulation of configuration G running *sp*

isons to DDR2. Technical report, Micron Technology Inc., April 2007.

[2] Low-Power Fully Buffered DIMM. Whitepaper, Netlist Inc., 51 Discovery, Suite 150, Irvine, CA, 2007.

[3] Netlist FBDIMM preliminary power analysis Training. Technical report, Netlist, September 2007.

[4] DDR2 SDRAM FBDIMM. Datasheet, Micron Technology, Inc., 2008.

[5] DDR3 SDRAM Specification. Technical report, Association, JEDEC Solid State Technology, July 2010.

[6] IBM Power 795 Technical Overview and Introduction. Datasheet, IBM, September 2010.

[7] Intel 7500 Scalable Memory Buffer. Technical report, Intel, March 2010.

[8] F. Bellard. QEMU, a fast and portable dynamic translator. In *Proceedings of the annual conference on USENIX Annual Technical Conference*, ATEC '05, pages 41–41, Berkeley, CA, USA, 2005. USENIX Association.

[9] C. Bienia and K. Li. PARSEC 2.0: A New Benchmark Suite for Chip-Multiprocessors. In *Proceedings of the 5th Annual Workshop on Modeling, Benchmarking and Simulation*, June 2009.

[10] V. Cuppu, B. Jacob, B. Davis, and T. Mudge. A Performance Comparison of Contemporary DRAM Architectures. In *Proc. 26th Annual International Symposium on Computer Architecture (ISCA'99)*, pages 222–233, Atlanta GA, may 1999. Published by the IEEE Computer Society.

[11] V. Cuppu, B. Jacob, B. Davis, and T. Mudge. High-performance DRAMs in workstation environments. *IEEE Transactions on Computers*, pages 1133–1153, 2001.

[12] H. Fredriksson and C. Svensson. Improvement Potential and Equalization Example for Multidrop DRAM Memory Buses. *IEEE Transaction On Advanced Packaging*, 32(3):675–682, 2009.

[13] B. Ganesh, A. Jaleel, D. Wang, and B. Jacob. Fully-buffered DIMM memory architectures: Understanding mechanisms, overheads and scaling. In *High Performance Computer Architecture, 2007. HPCA 2007. IEEE 13th International Symposium on*, pages 109–120, 2007.

[14] B. Jacob, S. W. Ng, and D. T. Wang. *Memory Systems : Cache, DRAM, Disk*. Morgan Kaufmann, 2008.

[15] M. LaPedus. Micron rolls DDR3 LRDIMM. *EE Times*, 2009.

[16] A. Patel, F. Afram, S. Chen, and K. Ghose. MARSSx86: A Full System Simulator for x86 CPUs. In *Design Automation Conference 2011 (DAC'11)*, 2011.

[17] P. Rosenfeld, E. Cooper-Balis, and B. Jacob. DRAMSim2: A Cycle Accurate Memory System Simulator. *IEEE Computer Architecture Letters*, 99(RapidPosts), 2011.

[18] G. Shigehiro. AMD's Next Server Platform "Maranello". *PC Watch*, 2008.

[19] A. L. Shimpi. The AMD Memory Roadmap: DDR3, FBD and G3MX Examined. *AnandTech.com*, 2007.

[20] S. Srinivasan. *Prefetching vs The Memory System : Optimizations for Multi-Core Server Platforms*. PhD thesis, University of Maryland, 2007.

[21] J. Suarez. Enterprise X-Architecture 5th Generation. Technical report, March 2010.

[22] D. T. Wang. *Modern DRAM Memory Systems : Performance Analysis and a High Performance, Power-Constrained DRAM Scheduling Algorithm*. PhD thesis, University of Maryland, 2005.

[23] S. Woo. DRAM and Memory System Trends. October 2004.

[24] M. T. Yourst. PTLsim: A cycle accurate full system x86-64 microarchitectural simulator. pages 23–34, 2007.

[25] H. Zheng, J. Lin, Z. Zhang, and Z. Zhu. Decoupled DIMM: building high-bandwidth memory system using low-speed DRAM devices. In *Proceedings of the 36th annual international symposium on Computer architecture*, ISCA '09, pages 255–266, New York, NY, USA, 2009. ACM.