MLP Aware Heterogeneous Memory System

Sujay Phadke and Satish Narayanasamy
University of Michigan, Ann Arbor
{sphadke,nsatish}@umich.edu

***Abstract*—Main memory plays a critical role in a computer system's performance and energy efficiency. Three key parameters define a main memory system's efficiency: latency, bandwidth, and power. Current memory systems tries to balance all these three parameters to achieve reasonable efficiency for most programs. However, in a multi-core system, applications with various memory demands are simultaneously executed.**

This paper proposes a heterogeneous main memory with three different memory modules, where each module is heavily optimized for one the three parameters at the cost of compromising the other two. Based on the memory access characteristics of an application, the operating system allocates its pages in a memory module that satisfies its memory requirements. When compared to a homogeneous memory system, we demonstrate through cycle-accurate simulations that our design results in about 13.5% increase in system performance and a 20% improvement in memory power.

I. Introduction

Off-chip main memory plays a critical role in determining the performance and power of a computer system. There are three main attributes that determine the efficiency an off-chip memory: latency (L), bandwidth (B), and power (P). Memory chip designers often need to make trade-offs between these parameters. For instance, Reduced Latency DRAM (RL-DRAM) [6] can operate at a latency of about `25ns` but offers only about 4 GB/s bandwidth. While DDR3-2133 chip [5] can offer nearly 18 GB/s, it operates at a much higher latency of about `45ns`. It may be possible to operate a DRAM chip at a lower voltage to reduce power, but that comes at the cost of increased latency and reduced bandwidth.

Currently, memory system designers try to balance across the three parameters for the main memory design. However, a general purpose multi-core system would be simultaneously running a diverse set of applications with different memory requirements. Some applications are bandwidth-bounded (e.g. graphics) and some others are latency-bounded (e.g. pointer-intensive applications). There are applications for which main memory system's performance does not matter (e.g. computationally intensive applications with high data locality). For such applications a power-efficient main memory would be a better option. The diversity of applications' memory demand in a system is only likely to increase in future as processors like AMD's Fusion [1] start to integrate specialized and domain specific cores into a multi-core processor.

In this paper, we present for the first time, a heterogeneous main memory for future multi-core systems. It consists of three memory modules. Each module is optimized for one of the above mentioned three parameters at the cost of sacrificing the other two.

978-3-9810801-7-9/DATE11/

To take advantage of our heterogeneous memory, we propose an offline profiling algorithm that classifies an application into one of the three types (L, B, P) based on its memory demand. Level-2 (L2) cache miss rate and Memory Level Parallelism (MLP) [10] are two key characteristics of an application that define its main memory demand. MLP is the ability to generate and service multiple outstanding L2 misses in parallel. When a memory operation results in a cache miss, modern out-of-order processors have the ability to execute later memory operations which exposes MLP [16]. If a later memory operation also incurs a cache miss, its latency can be hidden by servicing it in parallel with the earlier cache miss. Hence, performance of an application with high MLP tend to be insensitive to memory latency. Two applications with the same cache miss rate need not have similar degree of MLP, because MLP is more dependent on how clustered the cache misses are during an execution and whether a memory access is dependent on preceding memory accesses or not.

In general, scientific and graphics applications with data-level parallelism exhibit high MLP (hence, latency insensitive) and high L2 miss-rate, and therefore can benefit from a bandwidth optimized memory. Applications that heavily use pointer-based data structures tend to have low MLP and also incur significant L2 misses, and therefore can benefit from a low latency memory. Applications with low L2 miss-rate and high MLP do not require a high performance memory, and therefore can benefit from a low power memory.

Based on the above observations, we propose an offline algorithm that classifies an application into any one of the three types by profiling its L2 miss rate and MLP characteristics. We also propose a page allocation policy for an operating system which uses our profiler's classification to allocate pages of an application to the appropriate memory module at runtime.

We use M5 [8], an execution driven multi-core simulator, to model 4, 8 and 16 core processor configurations. We use DRAMSim [20] to model performance and power characteristics of our memory designs in detail. For each processor configuration, we evaluate nearly 120 workloads constructed from a set of SPEC CPU2006 benchmarks [7]. On average, our heterogeneous memory system improves system performance by 13.5% and improves memory system's power consumption by 20% for a 4-core system.

II. Motivation

Applications vary widely in terms of their main memory demands. Fig. 1 presents a case study to illustrate this observation. For a processor with one core (configuration shown

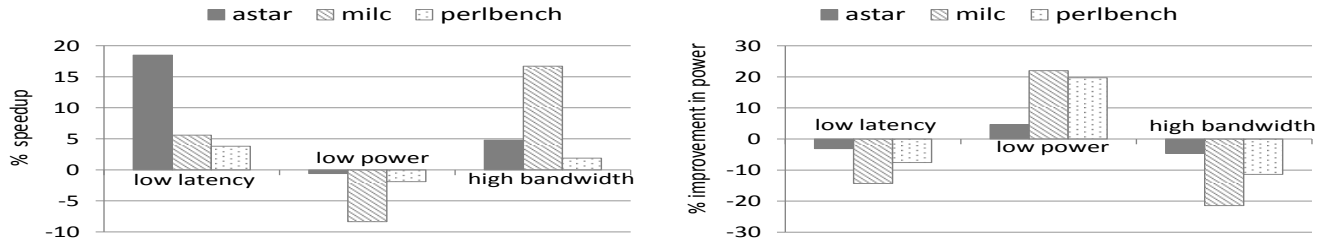

Fig. 1. Performance and power of memory devices optimized either for latency, bandwidth or power, when compared to DDR3.

in Table II), we study three benchmarks with diverse memory demands. `astar` is sensitive to latency, `milc` is sensitive to bandwidth, and `perlbench` has high locality and only rarely accesses memory. We consider three different types of memory modules optimized for either latency (M_L), bandwidth (M_B) or power (M_P)), at the cost of sacrificing some efficiency of the other two parameters. The detailed configurations for these three devices are discussed in Section IV (Table I). We evaluate their performance and power with respect to a commonly used DDR3-1600 memory, which we consider as our baseline.

As shown in Figure 1, a latency optimized memory improves `astar`'s performance by 18.5% for only about 1% power cost. But it is ill-suited for the other two applications. For `milc`, it improves performance only by about 5% for a significant power cost of 14%. The reason is that `milc` is bandwidth bounded and less sensitive to memory latency. Low power memory significantly reduces power by 20% for `perlbench` while incurring only 1% performance cost, because `perlbench` rarely accesses memory. However, low power memory is not suitable for `milc` as it suffers 8% performance penalty. Bandwidth bounded `milc`'s performance could be improved by nearly 17% using bandwidth optimized device. Though it comes at a power cost, the net energy-delay product is still better than baseline. But performance of the other two applications does not improve significantly while using bandwidth optimized memory.

Thus, if a multi-core system executes a mixture of the above three types of programs, then it would be beneficial to employ a memory device that contains three different memory modules where each one is heavily optimized for one of the three parameters.

III. Design Tradeoffs in DRAM Architecture

In this section we describe the baseline DRAM architecture and the trade-offs involved in its design.

DIMMs, Channels and Memory Controller: The main memory consists of a number of modules called *DIMMs* (dual in-line memory module) [11]. The DIMM modules are connected to the system bus (or a channel) which in turn connects them to a memory controller. A memory controller queues up memory requests from processor cores, arbitrates, sends control signals to DRAM, gets the data and sends it to the requested processor core. Modern processors have multiple integrated memory controllers servicing multiple channels. In this paper, all of our memory designs assume a single channel connected to three modules.

Ranks and Chips: A DIMM contains multiple ranks. A rank is composed of multiple chips. Each $\times$N chip has pins to support read/write of N bits in a cycle. For a 64-bit data bus and $\times 8$ chip, each rank needs 8 DRAM chips. A cache line is usually striped across chips so that its parts can be accessed in parallel.

Banks: Each chip is divided into *banks*. Contents of a single logical bank span across multiple chips and they can be accessed in parallel. Higher number of banks can service more memory requests in parallel, thereby providing higher memory bandwidth. However, higher number of banks also require more complex control logic and area [11] leading to higher bank access latency, and therefore could adversely impact the performance of a latency sensitive application that generates fewer parallel memory requests.

Arrays and Row Buffers: Within each bank, data is arranged in an array like fashion, with rows and columns. The DRAM row buffer and latches the data in the entire row when any part of it is accessed. A DRAM row is typically much larger than a single memory request. If a memory read accesses an already open row, it would experience smaller memory array read latency. Row buffer hit rates are higher for application with a higher spatial locality. In general, bandwidth bounded applications with regular memory access patterns tend to have a higher spatial locality, and therefore benefit from larger rows. Latency bounded applications with less regular memory access behavior, however, tend to exhibit poor spatial locality and therefore do not benefit from a large row buffer. In fact, for a latency sensitive application, a smaller row buffer would be beneficial as it would take lesser time to drive a smaller row. For a given DRAM array size, smaller rows may require longer column bit lines, but in most memory devices, benefits of reducing the row access time (RAS) generally outweighs relatively small increase in column access time (CAS). Thus, for bandwidth sensitive applications a larger row is beneficial, whereas for latency sensitive applications a smaller row is beneficial.

Prefetch buffer: The prefetch buffer in DDR SDRAMs is part of the chip architecture, and acts as the interface between the array and the DRAM interface. It can fetch multiple data words in a burst without the need for additional column address signals. A prefetch buffer with a depth x allows the interface I/O clock to operate x times faster than the memory array clock. DDR3, which we assume for all our designs in this paper, can support a buffer of maximum depth eight. But it can also support a smaller buffer depth that reduces latency but sacrifices memory bandwidth.

IV. Design of Heterogeneous Memory system

In this section, we present the design of a heterogeneous memory system to achieve performance and power improvements over the baseline homogeneous memory system.

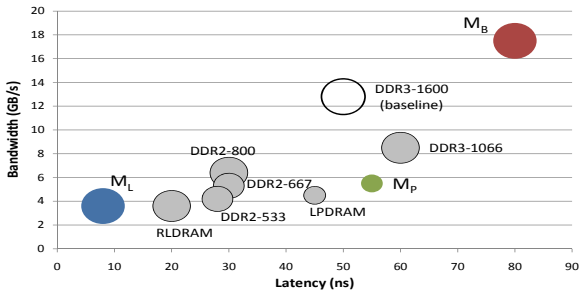

Fig. 2. Memory design space showing latency and bandwidth for different memory types. The size of the bubble corresponds to the average power.

TABLE I
HETEROGENEOUS MEMORY MODULES DESIGN SPACE

	# banks per module	Rowsize (bytes/ row)	Addr. mode	Pre-fetch buffer	Voltage (V)	Latency (ns)	B/W (GB/s)	Power (est.)
M_L	4	32	SRAM	2n	1.8	8	3.6	V. High
M_B	16	512	DRAM	8n	1.65	80	17.5	V. High
M_P	2	64	DRAM	1n	1.5-1.3	55	5.5	Low
DDR3-1600	8	256	DRAM	8n	1.5-1.3	50	12.8	High
DDR3-1066	8	256	DRAM	8n	1.5	60	8.5	High
DDR2-800	4	128	DRAM	4n	1.8	30	6.4	Med
DDR2-667	4	128	DRAM	4n	1.8	30	5.3	Med
DDR2-533	4	128	DRAM	4n	1.8	28	4.2	Med
ESDRAM 166 [22]	4	512	DRAM	2n	3.3	11	1.6	Med
LPDRAM	4	256	DRAM	2n	1.5	45	4.5	Low
RLDRAM 800	8	64	SRAM		1.5	20	3.6	High
FCRAM 200	4	128	DRAM		1.8	25	3.46	Low
PCM [15]	4	256	-		1.2	R: 50-100; W: 1000	0.1	V. Low

A. Memory Design Choices

The spectrum of design characteristics (latency, bandwidth and power) for several known implementations is shown in Fig. 2 and their detailed characteristics are presented in Table I. Sources for this data for some designs are provided in the table. The rest were obtained from the Micron data sheets [2]. Since different manufacturers quote different timings for the same DRAM (due to manufacturing methods, etc.), we provide an average value for the observed access latency. Power depends on activity which we derived using the Micron's power calculator [5]. Characteristics of three of our designs (M_L, M_P, M_B) were estimated using our simulation models which are discussed in Section VI-A.

As shown in Fig. 2 memory system architects balance the trade-off between three parameters. For example, DDR3–1600 has a high bandwidth, but comes at the cost of high latency and power. RLDRAM optimizes latency and is used in high speed network packet processing. However, the peak bandwidth of RLDRAM is much lower than the DDR3 family. LPDRAM [4] is optimized for power to support mobile applications at the cost of higher latency (compared to DDR2) and lower bandwidth (compared to DDR3). In short, there is no single homogeneous memory which can provide best possible latency, power and bandwidth.

B. Heterogeneous Architecture Overview

As described in the previous section, there are many different design points which we can use to build a heterogeneous memory. In this paper we discuss one simple heterogeneous main memory design. Our design is composed of three memory modules where each of them is either optimized for latency, bandwidth or power. We start with the DDR3-1600 design and optimize its structures for any one of the three parameters. For simplicity, our architecture assumes a single channel connecting three memory modules to an on-chip memory controller. Also, it uses the industry standard DDR3 interface as the interface of choice for all three types of memories. The interface can support varied latency modules and offers high bandwidth [3].

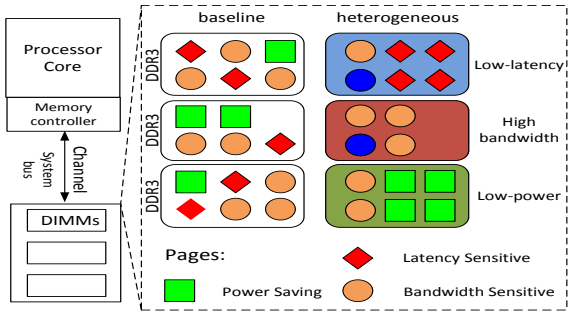

Fig. 3. Page Allocation in baseline homogeneous memory versus heterogeneous memory

Fig. 3 illustrates the difference between a homogeneous and a heterogeneous memory architecture. In our heterogeneous memory, based on the demand of an application, its pages would be stored in the appropriate memory module (Figure 3). For bandwidth sensitive applications, in addition to improved device efficiency, it is also important to exploit bank-level parallelism to issue simultaneous read/write commands to different banks in a pipelined fashion and achieve higher bandwidth. Therefore, we allocate pages of a bandwidth sensitive application across all the banks in all the memory modules. A profiling algorithm to classify applications and a heterogeneous memory aware page allocation policy are described in Section V. The rest of this section describes the three different memory modules used in our heterogeneous memory.

1) Latency Optimized Memory Module (M_L): DRAM latency is the time required to fetch data from the memory array and send it to the requesting CPU core. As described in Section III, smaller row buffers could reduce the worst case memory latency. While smaller rows could degrade the performance of bandwidth bounded applications with higher spatial locality, it could improve the performance for many latency-sensitive applications with poor spatial locality. For our latency optimized memory module, we study DRAM module with relatively smaller rows (32 bytes/row).

Smaller prefetch buffer depth could reduce I/O buffer latency due to lesser logic overhead and circuit area. The tradeoff is the reduced memory bandwidth. We use a prefetch buffer of depth `2n`.

When compared to DRAM addressing mode, an SRAM addressing mode (implemented in RLDRAM [6]) reduces memory latency. In traditional DRAM devices, the address is supplied in two consecutive clock cycles (command and bank address in the first clock cycle, remaining address in the second clock cycle). This provides the advantage of reducing the number of pins required on the controller side by half. In SRAM addressing, the entire address is provided in one clock cycle which reduces latency. However, power consumption

could be higher and pin bandwidth could be lower.

2) Bandwidth Optimized Memory Module (M_B): Bandwidth refers to the rate at which data can be transferred to or from memory over a period of time. As we described in Section IV-B, we allocate pages of a bandwidth sensitive application across the banks in all the modules to harness the benefit of bank-level parallelism. With bank-level parallelism, it is possible to issue simultaneous read/write commands to different banks in a pipelined fashion and achieve higher bandwidth. To further improve bandwidth, we employ a bandwidth-optimized module with relatively a large number of banks (16 banks in a module) and a large row buffer size (512 bytes/row). It uses DRAM multiplexed addressing scheme (instead of SRAM scheme) to improve effective pin bandwidth. We set the prefetch buffer depth to be `8n` which is the maximum supported by the DDR3 interface. All these design choices could lead to higher latency and power, but provide higher bandwidth.

3) Power Optimized Memory Module (M_P): We employ two techniques used in Micron's LPDRAM to reduce refresh power [4]. One is called Partial Array Self Refresh (PASR) where a memory controller selects a portion of memory to refresh. Another is called Temperature Compensated Self Refresh (TCSR) where an on-chip temperature sensor adapts the refresh interval based on the device temperature. In addition, we consider four key design choices to arrive at a low power design. First, we use a small number of banks (two banks per module). Second, we use small row size (32 bytes per row). We choose the smallest possible prefetch buffer size (`1n`) and restrict it to perform only one read/write per cycle. Finally, operation voltage is selected to be the minimum possible (1.5V). All these design choices reduce power but increases memory latency and reduces bandwidth.

V. Profiling and Operating System Support

We propose an efficient page allocation policy(V-B) using which an operating system can allocate pages of an application to an appropriate memory module without significantly increasing the page fault rate.

A. Classifying Applications

An application's type could be determined either at compile-time using profiling or at runtime. We employ offline profiling, but the mechanisms that we discuss can be adapted to engineer a dynamic profiler using processor's performance counters. In our study, we used execution-driven simulations to profile the memory access behavior of applications.

Our offline algorithm classifies applications based on its memory demand which is determined by profiling its L2 miss-rate and Memory Level Parallelism (MLP). MLP is a measure of how well we can hide the latency penalty of consecutive long latency loads. To measure the MLP of an application, we profile the average number of cycles a memory read instruction is stalled at the head of the re-order buffer (ROB) due to an L2 miss [16]. Higher this stall time, lower the MLP for that application.

Algorithm 1 describes our algorithm. If an application has an L2 MPKI (Misses Per Kilo Instructions) higher than a threshold and high MLP, we classify that application as bandwidth sensitive. If an application is not bandwidth intensive, then we check if the application has L2 MPKI higher than another threshold *and* whether it has low memory-level parallelism (MLP). If an application is neither bandwidth nor latency sensitive, then it is classified as a candidate for power optimization.

Algorithm 1 Benchmark Classification Algorithm

```
for all benchmarks do
  if (L2 MPKI ≥ thr_1b) AND (ROB stall time ≤ thr_mlp) then
    Type B: optimize for bandwidth
  else if (L2 MPKI ≥ thr_1l) AND (ROB stall time ≥ thr_mlp)
  then
    Type L: optimize for latency
  else
    Type P: optimize for power
  end if
end for
```

thr_{1b} = 10, thr_{1l} = 0.5, thr_{mlp} = 5

B. Page Allocation and Page Fault Mechanism

The operating system (OS) selects physical pages to store an application's virtual page. We propose a slightly modified LRU page replacement policy to ensure that an application's page is stored in a memory module that meets its demand as much as possible. For `type L` and `type P` applications, the pages should be mapped to their respective modules. For `type B` workloads, its pages should be allocated across all the modules in a round-robin fashion to maximize module- and bank-level parallelism.

Our modified LRU (Least Recently Used) policy works as follows. On encountering a page fault, the OS selects *m* least recently used pages as potential candidates for replacement. From these *m* pages, it picks the best victim based on whether it is stored in a module that would benefit the new page's application's memory demand. If none of the *m* pages are stored in the required memory type, then the OS picks the closest match. If a page in the latency optimized module is not available among the *m* pages, we select a page in the bandwidth optimized module. If a page in power optimized module is not available, we try to find a page in the latency optimized module first. If both of these attempts fail, then the most LRU page is replaced, irrespective of its type. In our study, we configured *m* to be 10. This page replacement policy ensures that most pages are allocated on the desired memory module while not significantly increasing the page fault rate when compared to our baseline LRU policy.

VI. Results

This section evaluates the performance and power benefits of our heterogeneous memory architecture.

A. Experimental Setup

We use the M5 [8] execution-driven simulator for modeling the performance of the multi-core processor and the system bus. The detailed configuration is presented in Table II. The simulator was augmented with DRAMSim [20] for detailed DRAM timing. It models the DRAM system in detail for each type of memory we studied, by keeping track of the internal

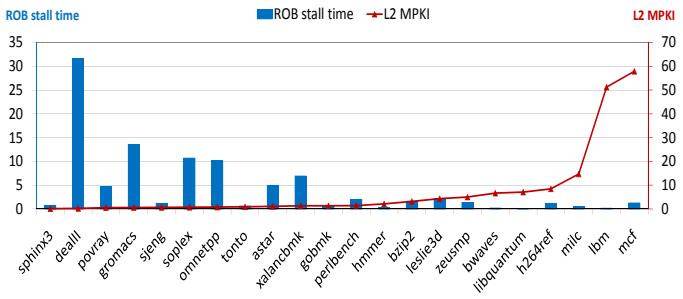

Fig. 4. Bar graph: Number of machine cycles spent stalling at head of the ROB per L2 read miss. Line Graph: L2 MPKI

states of each DRAM rank, bank and channel. We model memory, bus and memory controller delays and contention as well as ability to service multiple outstanding memory requests. We use an open-page policy in the SDRAMs. Power modeling was done by using activity factors from the simulation and feeding them into the Micron DRAM power calculator [5]. To model the power saving techniques used in M_P, we scaled the worst-case power figures in the DRAM calculator by the potential savings that can be obtained.

TABLE II
SYSTEM CONFIGURATION

Execution core	1 GHz Alpha ISA out-of-order, Dispatch/Issue/Commit width 8, 80 entry ROB, 32 entry LSQ, 4K entry gshare branch predictor
On-chip caches	64KB split L1, 2-way, 2 cycles, 1 read/write port, Unified L2 (2-8 MB), 4-way, 15 cycles, 1 read/write port, 64B line size
Pre-fetcher	distance of 64, tagged, degree of 2, 4k entry gshare BP
Mem-core interface	DDR3, 128-bit channel
Main Memory	Baseline: Three 1GB DDR3-1600 modules

We used the SPEC CPU2006 V1.1 suite of benchmarks [7]. We evaluated 24 of the 31 benchmarks which includes all the benchmarks that we could run on our simulator. We used the available Simpoints [17] to select phases of the workloads. After fast-forwarding to the relevant simpoint and warming up for 1 million cycles, we simulated 100 million instructions in detailed out-of-order mode for each core. For a multi-core, multi-workload simulation, we simulated until every core has executed at least 100 million instructions.

B. Benchmark Classification

Using the profiling algorithm described in Section V-A, we classified benchmarks into 3 types as shown in Table III. Figure 4 shows the L2 Misses-Per-Kilo-Instructions (MPKI) and the average number of processor cycles that a memory operation spends at the head of the ROB. Three benchmarks (*lbm, mcf and milc*) are sensitive to bandwidth due to high L2 MPKI and high MLP (as seen from the low ROB stall time in Fig. 4). Six applications (e.g. `dealII`) with high ROB stall time and adequate L2 MPKI are classified as latency sensitive. The rest are classified as candidates for power optimization.

TABLE III
BENCHMARKS CLASSIFICATION

Type	Benchmarks
L (latency)	dealII, gromacs, soplex,omnetpp, xalanbmk, astar
P (power)	povray, leslie3d, perlbench,bzip2, zeusmp, libquantum, h264ref, sjeng, sphinx3, tonto, hmmer, gobmk, bwaves
B (bandwidth)	lbm, mcf, milc

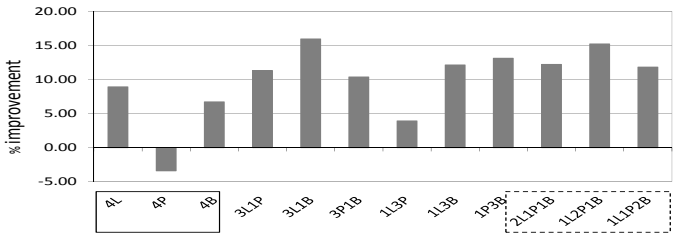

Fig. 5. Performance of heterogeneous memory system over DDR3 homogeneous memory for a 4-core system.

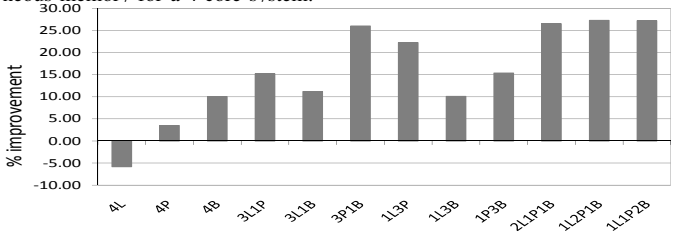

Fig. 6. Power improvements heterogeneous memory system over DDR3 homogeneous memory for a 4-core system.

C. Workload Mixes and System Setup

We evaluate for 4, 8 and 16 core configurations, each with 12 different workload *types*. A workload *type* is determined by the number of Latency (L), Power (P) and Bandwidth (B) applications in the workload. `1L4P4B` stands for a workload type with 1 latency sensitive benchmark, 4 power optimized benchmarks and 4 bandwidth bounded benchmarks (determined in Section VI-B). For each workload type, we consider 10 different workloads (mixture of different benchmarks selected based on their type), thus yielding a total of 120 workloads for each of the three processor configuration.

We use weighted speedup to measure the performance of a multicore system. $W.Speedup = \sum_n \left(\frac{IPC^i_{shared}}{IPC^i_{alone}}\right)$, where IPC^i_{shared} and IPC^i_{alone} denote the IPC of the i^{th} application when running in a shared multi-core environment with others, and running alone respectively.

D. Benefits of Heterogeneous Memory

Fig. 5 and Fig. 6 shows the improvement in weighted speedup and power for a 4-core system using a heterogeneous memory (M_L, M_P, M_B) when compared to the DDR3-1600 homogeneous memory. We show results for 12 different workload types. For each workload type, we present the average improvement observed for the 10 workload mixtures.

We observe improved efficiency for most workload types. Maximum gains are observed for types with roughly equal distribution among application types (last three on the right-hand side) – 16% performance, 26% memory's power. Our worst case is when application mixture is heavily imbalanced (leftmost 3 workload types). Even so, the maximum performance penalty noticed is only 3% (`4P`). We believe such imbalance in workload mixture would be rare in most user scenarios. On average our system improves performance by 13.5%, memory's power by 20.01% averaged across 120 different workloads for a 4-core configuration.

Fig. 7 shows results for 4, 8 and 16 cores (average over 120 workloads for each configuration). We find that heterogeneous memory retains its advantages as the number of cores scale.

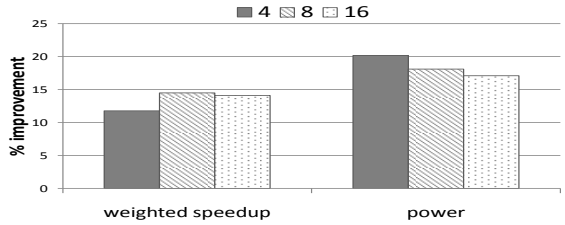

Fig. 7. Improvements in weighted speedup (WS) and memory power for 4, 8 and 16-core system due to Heterogeneous Memory.

We find on average performance improves by 14% (16-core) and memory's power reduces by 17% (16-core).

We measured the efficiency of our page allocation method (Section V-B) for a 4-core system. For only about 1% increase in page fault rate, our policy was able to allocate pages on the required memory type for most pages. Only 2.35% memory accesses were to a page that was allocated on a memory type that was not optimal.

VII. Related Work

To our knowledge, we are the first to make a case for heterogeneity in memory modules to cater to the varying memory demands of applications concurrently running on a multi-core processor. Given the space limitation, here we touch upon only a few closely related prior works.

Hybrid memory systems with a high performance DRAM as the first level main memory and a secondary large low power non-volatile memory has been proposed [12], [18] to reduce the long latency in accessing disks. These earlier works studied a hierarchical organization based on the latency of a memory type. Whereas, our design goal is to employ memories with different characteristics to meet diverse memory demand across applications. As the performance of a non-volatile memory scales to get closer to that of DRAM's, it could serve as one of the power optimized module in our heterogeneous design.

Previous works like Mini-rank [21] describe techniques that can improve bandwidth and power efficiency, but for some latency cost. They do not consider the advantage of heterogeneous memory modules, but their technique could be one of the ways to heavily optimize a memory module for one parameter (bandwidth) at cost of sacrificing the other (latency). Several prior works proposed techniques to optimize DRAM energy [19] and power [14]. All these ideas are complementary to ours as these techniques can be used to further improve the efficiency of modules in a heterogeneous memory.

Microarchitecture designs to expose and exploit MLP to reduce the impact of memory [16] and network latency [9] on an application's performance have been proposed. While the advantages of heterogeneous cores on a chip have been studied earlier [13], our work makes a case for heterogeneous main memory.

VIII. Conclusions

Most computer systems already use multi-core processors. This paper demonstrates that different applications that could run simultaneously on multi-core systems have varied needs from the main memory system. We proposed a heterogeneous memory system to provide better latency and bandwidth based on an application's memory demand. We also discussed a profiling method to classify applications, and a page allocation policy that can take advantage of a heterogeneous memory. Through cycle accurate simulations, we demonstrated significant performance and power benefits of our heterogeneous memory.

Acknowledgment

We thank Felix Loh and Chaz Whiting for their help with the simulation infrastructure. We thank Reetuparna Das and the anonymous reviewers for providing their valuable feedback.

References

[1] AMD Fusion™ Family of APUs: Enabling a Superior, Immersive PC Experience. http://sites.amd.com/us/Documents/48423B_fusion_whitepaper_WEB.pdd.

[2] DDR3 SDRAM Part Catalog. http://www.micron.com/partscatalog.html?categoryPath=products/parametri%c/dram/ddr3_sdram.

[3] External Memory Interface Handbook Volume 3. http://www.altera.com/literature/hb/external-memory/emi_ddr3up_ug.pdf.

[4] Low-power versus standard ddr sdram. http://download.micron.com/pdf/technotes/DDR/tn4615.pdf.

[5] The micron system-power calculator. http://www.micron.com/support/dram/power_calc.html.

[6] Reduced latency dram (rldram). http://www.micron.com/products/ProductDetails.html?product=products/dra%m/MT49H16M16FM-33.

[7] SPEC CPU2006. http://www.spec.org/cpu2006.

[8] N. L. Binkert, R. G. Dreslinski, L. R. Hsu, K. T. Lim, A. G. Saidi, and S. K. Reinhardt. The m5 simulator: Modeling networked systems. *IEEE Micro*, 26(4):52–60, 2006.

[9] R. Das, O. Mutlu, T. Moscibroda, and C. R. Das. Application-aware prioritization mechanisms for on-chip networks. In *MICRO*, pages 280–291, 2009.

[10] A. Glew. MLP yes! ILP no! In *Wild and Crazy Idea Session, ASPLOS*, 1998.

[11] B. Jacob, S. Ng, and D. Wang. Memory systems: Cache, dram, disk. In *Elsevier*, 2008.

[12] T. Kgil, D. Roberts, and T. Mudge. Improving nand flash based disk caches. In *Proceedings of the 35th International Symposium on Computer Architecture*, pages 327–338, 2008.

[13] R. Kumar, K. I. Farkas, N. P. Jouppi, P. Ranganathan, and D. M. Tullsen. Single-isa heterogeneous multi-core architectures: The potential for processor power reduction. In *MICRO*, pages 81–92, 2003.

[14] A. R. Lebeck, X. Fan, H. Zeng, and C. Ellis. Power aware page allocation. In *Ninth Int'l Conf. on Arch. Support for Programming Languages and Operating Systems*, pages 105–116, 2000.

[15] B. C. Lee, E. Ipek, O. Mutlu, and D. Burger. Architecting phase change memory as a scalable dram alternative. In *Proceedings of the 36th International Symposium on Computer Architecture*, 2009.

[16] O. Mutlu, H. Kim, and Y. N. Patt. Efficient runahead execution: Power-efficient memory latency tolerance. *IEEE Micro*, 26(1):10–20, 2006.

[17] A. Nair and L. John. Simulation points for spec cpu 2006. In *Proceedings of the 26th International Conference on Computer Design, ICCD*, pages 397–403, 2008.

[18] M. K. Qureshi, V. Srinivasan, and J. A. Rivers. Scalable high performance main memory system using phase-change memory technology. In *ISCA*, pages 24–33, 2009.

[19] 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 *ISCA '10*, pages 175–186, 2010.

[20] D. Wang, B. Ganesh, N. Tuaycharoen, K. Baynes, A. Jaleel, and B. Jacob. Dramsim: A memory-system simulator. *SIGARCH Computer Architecture News*, 33(4):100–107, 2005.

[21] H. Zheng, J. Lin, Z. Zhang, E. Gorbatov, H. David, and Z. Zhu. Mini-rank: Adaptive dram architecture for improving memory power efficiency. In *Proceedings of the 2008 41st IEEE/ACM International Symposium on Microarchitecture*, pages 210–221, 2008.

[22] H. Zheng, J. Lin, Z. Zhang, and Z. Zhu. Decoupled dimm: Building high-bandwidth memory system using low-speed dram devices. In *Int'l Symp. on Computer Arch.*, June 2009.