Simple but Effective Heterogeneous Main Memory with On-Chip Memory Controller Support

Xiangyu Dong†‡, Yuan Xie†, Naveen Muralimanohar‡, Norman P. Jouppi‡
Computer Science and Engineering, Pennsylvania State University†, Exascale Computing Lab, Hewlett-Packard Labs‡
{xydong,yuanxie}@cse.psu.edu†, {xiangyu.dong,naveen.muralimanohar,norm.jouppi}@hp.com‡

***Abstract*—System-in-Package (SiP) and 3D integration are promising technologies to bring more memory onto a microprocessor package to mitigate the "memory wall" problem. In this paper, instead of using them to build caches, we study a heterogenous main memory using both on- and off-package memories providing both fast and high-bandwidth on-package accesses and expandable and low-cost commodity off-package memory capacity. We introduce another layer of address translation coupled with an on-chip memory controller that can dynamically migrate data between off-package and off-package memory either in hardware or with operating system assistance depending on the migration granularity. Our experimental results demonstrate that such design can achieve the average effectiveness of 83% of the ideal case where all memory can be placed in high-speed on-package memory for our simulated benchmarks[1].**

I. Introduction

Recent trends of multi/many core microprocessor design with increasing number of cores have accentuated the already daunting memory bandwidth problem. The traditional approach to mitigate the "memory wall" problem is to add more storage on-chip in the form of last-level cache (LLC). For example, IBM POWER7 microprocessor has a 32MB L3 cache built out of embedded-DRAM (eDRAM) technology. The decrease in miss-rate achieved by the extra cache size helps hide the latency gap between a processor and its memory.

Emerging System-in-Package (SiP) and three-dimensional (3D) integration technologies [1], [2] enable designers to integrate gigabytes of memory into the microprocessor package. However, using the on-package memory resources as a last-level cache (LLC) might not be the best solution. High-performance commodity DRAM dies such as GDDR are heavily optimized for cost and do not include specialized high performance tag arrays that can automatically determine a cache hit/miss and forward the request to the corresponding data array. Since the size of a tag array can be a hundred megabytes or more for a multi-gigabyte cache, storing tags in the microprocessor die is not feasible and the only alternative is to put the LLC tags in the on-package DRAM.[2] Moreover, it will dissipate too much power and waste too much bandwidth if all the ways are speculatively read from a highly associative on-package cache (e.g., 16-way) at the same time. Doing so would either increase the bandwidth and I/O requirements of the CPU chip by a factor of 16 and DRAM dynamic power by a factor of 16, or if the I/O count and DRAM power were kept constant would reduce the on-package memory bandwidth by a factor of 16. Instead we implement a 15-way set associative cache in the space of a 16-way set-associative data array, packing all the tags for a set into the 16th cache line for each set. We then access the tags first, and then access the data after a tag hit when the data way location is known. This makes the cache miss/hit determination time roughly equal to the on-package DRAM access time, and makes returning the data on a cache hit take approximately 2X of the time to access the on-package DRAM. Note that even if custom cache DRAM chips were developed, for similar reasons two sequential DRAM accesses would still be required for returning data on a cache hit.

Consequently, in this paper, instead of utilizing these on-package memory resources to augment existing caches or deepen the cache hierarchy, we propose a heterogeneous main memory architecture that consists of both on-package memories and off-package Dual Inline Memory Modules (DIMMs). To manage such a space and move frequently accessed data to fast regions, we propose two integrated memory controller schemes: a first technique handles everything in hardware and our second scheme takes assistance from the operating system. The effectiveness of each scheme depends on the memory management granularity. Through our evaluation, we show that our low-overhead solution can reduce off-package memory access traffic by 83% on average.

II. Heterogeneous Main Memory Space

In this work, we assume that for future microprocessor packages, DRAM dies are placed beside the microprocessor die using SiP as shown in Fig. 1. As the flip-chip SiP can provide die-to-die bandwidth of at least 2Tbps [3], the on-package high-performance DRAM chip is modified from existing commodity products to increase the number of banks and further increase the signal I/O speed to take advantage of the high-speed on-package interconnects. However, we do not assume a custom tag part and use only a single on-package DRAM design in order to reduce design cost and maximize the volume of the on-package DRAM parts. Although we leave the room for the potential through-silicon-vias-based (TSV-based) 3D stacking in the future, in this work, we assume the on-package memory size is around hundreds of megabytes due to the consideration that the capabilities of delivering power into

[1]X. Dong and Y. Xie were supported in part by NSF grants 0702617, 0903432, 0905365, and SRC grants.

[2]If there are 32 DRAM dies stacked in the package, and cache tags are 6.7% of the data size, this would still require the equivalent of 2.1 DRAM chips of area. Furthermore, the multi-core CPU chip is unlikely to support DRAM as dense as the commodity DRAM chips.

SC10 November 2010, New Orleans, Louisiana, USA 978-1-4244-7558-2/10/$26.00

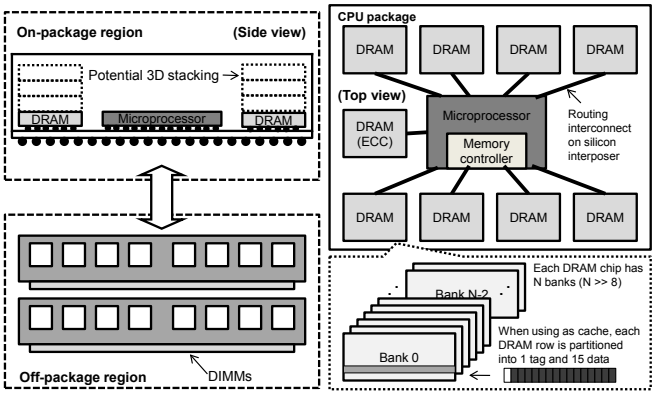

Fig. 1. The conceptual view of the System-in-Package (SiP) solution with 1 microprocessor die and 9 DRAM dies connecting off-package DIMMs (one on-package DRAM die is for ECC). The on-package DRAM chip is slightly different from the commodity off-package DRAM as it has a many-bank structure to reduce the memory access queuing delay. When the on-package DRAM dies are configured to be LLC, they function as 15-way associative cache since each DRAM row is partitioned into 1 tag and 15 data.

and dissipating heat out of the chip package are still limited. However, the available on-package memory capacity is much larger than the state-of-the-art LLC capacity.

Because the on-package DRAM can have fast access speed, it is ideal we can use them as the unified system main memory [4]–[7]. While some application domains do not require lots of main memory [4], generally the aggregate memory on-package, which is assumed to be a gigabyte in this work, is still not sufficient to hold the entire main memory space, which can be several gigabytes. For our system, in addition to the on-package DRAM, we model four DDR3 channels connected to traditional DRAM DIMMs. For the rest of the paper, we refer to these DIMMs as off-package DRAM.

A. On-Chip Memory Controller

It is not practical to implement the heterogeneous main memory space if the memory controller is off-chip and all the memory accesses have to leave the microprocessor package first. However, with the help of an on-chip memory controller, it becomes simple to form a heterogeneous memory space with both on- and off-package memories. As shown in Fig. 1, the memory controller on the microprocessor die is connected to off-package DIMMs with the conventional 64-bit DDRx bus and on-package memory with a customized memory bus. MSBs of physical memory addresses are used to decode the target location. For example, if 1GB of 32-bit memory space is on-package, *Addr[31..30]=00* is mapped to on-package memory while *Addr[31..30]=*{*01,10,11*} is mapped to off-package DIMMs. It is necessary to make several minor modifications so that the memory controller has the capability of mapping physical memory addresses to either on-package or off-package regions. Fig. 2 and Fig. 3 show the difference between a normal on-package DRAM memory controller and our proposed heterogeneity-aware one. The modifications are listed as follows,

- As shown in Fig. 2, the conventional memory controller first schedules memory transactions by combining and reordering to achieve performance optimization, and then the memory physical address is further resolved into indices in a DRAM memory system in terms of channel ID, rank ID, bank ID, row ID, and column ID. However, in our heterogeneity-aware memory controller, as shown in Fig. 3, the *Address Translation* stage is moved ahead so that each memory access is first routed to either the on-package region or the off-package region in addition to the DRAM indices such as channel ID, rank ID, *etc*. Then, transaction scheduling is performed for on-package and off-package regions separately, since the transaction-layer optimization for each region is independent of that for the other region.
- The removal of off-package electrical signaling for on-package memory is another minor change (see Fig. 3).
- The *Migration Controller* is another key component to make the memory controller "smart" in Fig. 3. It can be either pure hardware-based or OS-assisted depending on the migration granularity. The detailed discussion of its implementation and algorithm will be presented in Section III. In general, the migration controller monitors the recent memory access behavior, reconfigures the physical address routing, and sends out additional memory operations to swap the data across the package boundary.

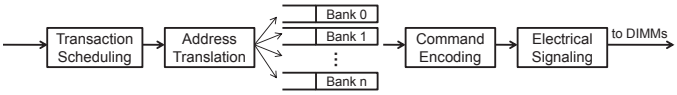

Fig. 2. Illustration of a conventional on-chip DRAM memory controller.

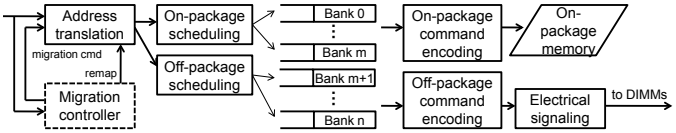

Fig. 3. Illustration of the proposed heterogeneity-aware on-chip DRAM memory controller with optional migration controller.

B. Performance Comparison to Larger LLCs

Rather than using on-package memory resources as a part of the main memory space, an alternative is to simply use them to expand the LLC capacity or deepen the cache hierarchy. Looking into the well-known average access latency approximation as quoted below,

$$\text{Average access time} = \text{Hit time} + \text{Miss rate} \times \text{Miss penalty}$$

the conventional cache hierarchy design only works effectively when the difference between the *Hit time* and the *Miss penalty* is large. When the LLC latency is approaching the off-package main memory latency (see Table II), the relatively small difference between *Hit time* and *Miss penalty* does not justify the use of the on-package memory as a cache. Furthermore, Fig. 4 shows that there is almost no benefit to enlarge the LLC capacity in terms of the cache miss rate. While accessing the LLC and the main memory in parallel can help hide the long LLC access latency, there is not enough off-package bandwidth to access the off-package memory speculatively and simultaneously with every reference to an on-package cache memory. Furthermore, off-package references consume

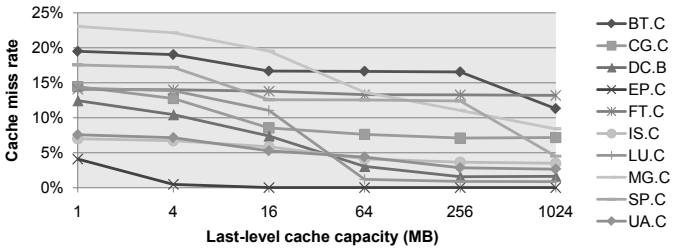

Fig. 4. The cache miss rate of different LLC capacities.

significantly more power and should generally be avoided when possible.

To validate the concept that it makes more sense to leverage the large on-package memory as a part of the main memory instead of an LLC, we run simulations on Simics [8] and compare the performance of different options using the metric of total IPC. For our quad-core system, our performance evaluation makes use of 4-thread OpenMP version of workloads from NAS Parallel Benchmark (NPB) 3.3. We use CLASS-C as the problem size so that the general memory footprint of the workload is sufficiently large for our purpose. The memory footprints of all the 10 workloads in NPB 3.3 suite are listed in Table I. Note that CLASS-C for the workload DC is not available in the benchmark package and therefore we use CLASS-B instead.

The simulation target is an Intel i7-like quad-core processor with private L1/L2 caches and a shared inclusive 8MB, 16-way L3 cache. The latencies of L1, L2, and L3 caches are computed by using CACTI [9] with 45nm technology. The main memory latency is modeled according to Micron DDR3-1866 datasheet [10]. The average random access memory latency depends on the actual memory reference timing and sequence. By assuming the SiP solution is used as shown in Fig. 1, we set the on-package memory capacity to be 1GB. The on-package and off-package memory latencies are modeled as follows,

- Off-package latency is the summation of the DRAM core access latency, the memory reference queuing delay, the memory controller traversal delay (including memory controller processing delay and the propagation delay between the CPU core and the memory controller), the package pin delay, and the PCB wiring delay;
- On-package latency is the summation of the DRAM core access latency, the memory controller traversal delay, the silicon interposer pin delay, and the intra-package wiring delay. Note that the memory reference queuing delay is not included in the on-package latency model since it is almost eliminated by the the increase in the number of on-package DRAM chip banks and their higher I/O speeds (our later trace-based simulation shows that accessing the off-package 8-bank DRAM causes 107 cycles of queuing delay while accessing the on-package 128-bank DRAM only causes less than 3 cycles on average).

In this Simics simulation, we simply model the DRAM core access latency as fixed numbers and set them to be 60-cycle and the queuing delay to be 120-cycle, respectively. In the trace-based simulation we later demonstrate in Section IV, a detailed DRAM timing model with FR-FCFS scheduling policy [11] will be used to obtain more accurate results.

TABLE I
THE MEMORY FOOTPRINTS OF NPB 3.3 BENCHMARK SUITE

Workload	Memory	Workload	Memory
BT.C	706MB	CG.C	920MB
DC.B	5876MB	EP.C	16MB
FT.C	5147MB	IS.C	1064MB
LU.C	615MB	MG.C	3426MB
SP.C	758MB	UA.C	510MB

TABLE II
THE SIMULATION CONFIGURATION OF THE BASELINE PROCESSOR AND THE OPTIONAL ENHANCEMENTS WITH ON-PACKAGE DRAM MODULES

Microprocessor	
Number of cores	4
Frequency	3.2GHz
Cache/Memory Hierarchy	
DL1 and IL1 caches	32KB, 8-way, 2-cycle, private
L2 cache	256KB, 8-way, 5-cycle, private
L3 cache	8MB, 16-way, 25-cycle, shared
Miscellaneous	
Memory controller	5-cycle for processing
Controller-to-core delay	4-cycle each way
Package pin delay	5-cycle each way
PCB wire delay	11-cycle round-trip
Interposer pin delay	3-cycle each way
Inter-package delay	1-cycle round-trip
DRAM core delay	50-cycle
Queuing delay	116-cycle
On-package LLC or memory	
L4 cache	1GB, 15-way, hit 140-cycle, miss 70-cycle
On-package memory	1GB, 70-cycle
Off-package memory	200-cycle
Simulation Cycles	
Fast-forward	Pre-defined breakpoint
Warm-up	1 billion instructions
Full simulation	10 billion cycles

The on-package memory is either employed as an L4 shared cache or the on-package memory region of a heterogenous memory space. The detailed architecture configuration is listed in Table II. Note that when using on-package DRAM as LLC, the cache hit time is 2X the DRAM access latency as the tag and the data are accessed sequentially.

As illustrated in Fig.5, the simulation result shows that using the extra 1GB on-package DRAM resources to add a new L4 cache can improve the IPC, but in some cases (e.g. CG.C) the performance improvement is limited to 0.1%. This is because cache capacity misses are not the performance bottleneck beyond a certain capacity threshold, while the increased cache latency starts to offset the benefit and degrade the performance as illustrated in Fig. 4.

On the other hand, directly mapping on-package DRAM resources into the main memory space can often achieve better performance. As shown in Fig. 5, for 7 out of the total 10 workloads that have memory footprint of less than 1GB, this strategy is equivalent to having all the memory on-package. For the other 3 workloads, MG.C has slightly better performance using the on-package memory as a heterogenous memory instead of an L4 cache, and DC.B and FT.C cannot compete against the L4 cache. However, note the simulation result shown in Fig. 5 is only the case of a static mapping. As we will show in the next section, a heterogeneous main memory with dynamic mapping that intelligently migrates

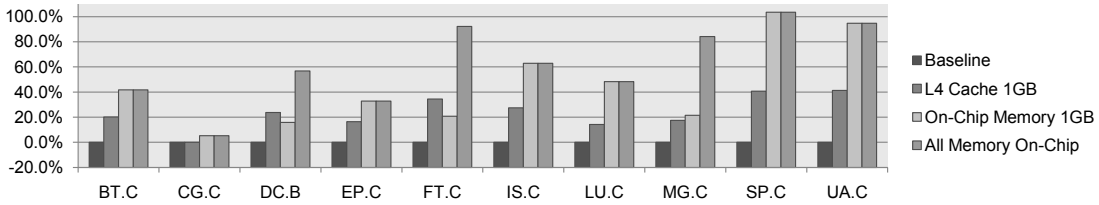

Fig. 5. The IPC comparison among different options of using on-package memory: (a) baseline; (b) Deepen the cache hierarchy with a 1GB DRAM L4 cache; (c) Statically map the first 1GB memory to the on-package DRAM; (d) The ideal case (all the DRAM resources are on-package).

data between on-package and off-package regions can further improve the performance and approach the ideal performance as described in Section III.

III. Data Migration Using On-Chip Memory Controller

As discussed in the previous section, the means of static memory mapping that always keeps the lowest memory address space on-chip only works effectively for the cases where the application memory footprints fit into the on-chip memory. For workloads that need much more memory resources, the performance improvement achieved by static mapping is trivial. For example, the performance improvement of DC.B is only 16% and that of FT.C is only 20.7%. Both of them are less than the ones achieved by using on-package DRAM as LLC. The major reason is the lack of the dynamic data mapping capability.

To solve this issue, we propose to add the data migration functionality into the memory controller so that frequently-used portions of memory can reside on-package with higher probability. Compared to other work on data migration [12]–[16]: (1) our data migration is implemented by introducing another layer of address translation; (2) depending on the data granularity, we propose either a pure-hardware implementation or an OS-assisted implementation; and (3) a novel migration algorithm is used to hide the data migration overhead.

In this paper, we use the term *macro page* as the data migration granularity, and the *macro page size* can be much larger than the *page size* of 4KB which is used in most operating systems.

A. Data Migration Algorithm

In this work, our data migration algorithm is based on *Hottest-coldest swapping* mechanism, which first monitors the LRU (least recently used) on-package macro page (*the coldest*) and the MRU (most recently used) off-package macro page (*the hottest*) during the last period of execution and then triggers the memory migration if the off-package MRU page is accessed more frequently than the on-package LRU page after each monitoring epoch.

As demonstrated later in the simulation results, it is a simple but effective method to implement the data migration. In this section, we focus on describing the migration algorithm in an incremental way starting from a relatively straightforward design.

Basic Design – N Mode

This is the basic design, in which the on-chip memory controller monitors the LRU on-package macro page and the MRU off-package macro page, and maintains a translation table. As shown in Fig. 6, assuming the memory space has 48 address bits and the macro page size is 4MB, the lowest 22 bits are the offset address of each macro page, and the highest 26 bits are the macro page IDs. The macro page ID is re-mapped to a new ID through a translation table. Considering the case that on-package memory has the capacity of 1GB, there are N=256 entries in the translation table. The left column in the translation table, as shown in Fig. 6, is actually the row address. Each row represents one on-package memory slot, while the right column represents which macro page is currently located in the on-package memory region. Note that the right column of the translation table is initialized to contain the same value as its left column counterpart so that the lowest 1GB main memory is initially located on-package. In addition, this translation table is bi-directional. For input macro page IDs smaller than N, the translation table works as a RAM; for IDs bigger than N, it works as a CAM. We call this design "N Mode" because all the N slots in the on-package memory region are utilized.

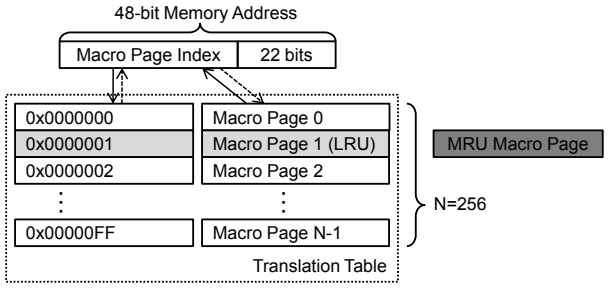

Fig. 6. The basic migration management design (N Mode).

The memory controller always monitors the LRU on-package macro page (for example, the 2nd row in Fig. 6) and the MRU off-package macro page. During the data migration procedure, the translation table is updated and the right column of the 2nd row in Fig. 6 will store the MRU macro page ID. However, the problem of this basic implementation is that data stored in LRU and MRU pages have to be swapped before the translation table is updated. Since the macro page size is usually large and the off-package DDRx bandwidth is limited, it will halt the execution and incur unacceptable performance overhead. Therefore, techniques to hide data movement latency is necessary.

N-1 Mode

As an improved design, the "N-1 Mode", in which one on-package memory slot is sacrificed, is proposed. As shown in Fig. 7, although there are still N slots, one of them (initially the last slot) does not map to any memory address and it is

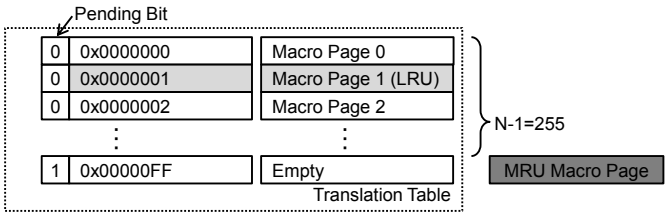

Fig. 7. The improved migration management design sacrificing one on-package memory slot for pending memory transactions (N-1 Mode).

marked as "empty". In practice, "empty" can be represented by a reserved macro page Ω (e.g., the highest 4MB macro page with ID of 0x800 in the 8GB memory space[3]). Therefore, the number of the effective on-package memory slots is N-1. Furthermore, each row in the translation table has an additional P bit, called "pending bit", which is initialized to be '0'. When P bit is set, the RAM function of the empty slot is bypassed and the left column is always translated to Ω instead, while the CAM function still works. The characteristics of the translation table are: *If macro page* $n(n < N)$ *is located in the on-package region, it can only be in the position of the n-th row.* Before we explain the swapping algorithm, we define all the macro pages into 5 categories:

1) *Original Fast (OF)*: Macro page whose ID is less than N, and its data is located in the on-package memory region without any address translation. Its address is in the left column of the address table but pointing to itself.
2) *Original Slow (OS)*: Macro page whose ID is greater than N, and its data is located in the off-package memory region without any address translation. Its address is not in the translation table.
3) *Migrated Fast (MF)*: Macro page whose ID is greater than N, but its data is migrated into the on-package memory region. Its address is in the right column of the translation table.
4) *Migrated Slow (MS)*: Macro page whose ID is less than N, but its data is migrated out of the on-package memory region. Its address is in the left column of the translation table and pointing to a new address.
5) *Ghost*: Macro page whose ID is less than N, but its data is migrated to a reserved macro page in the off-package memory region. Its address is in the left column of the translation table and pointing to the reserved macro page (i.e. 0x800).

The following algorithm describes how to performance a *hottest-coldest swap*.

- If the macro page ID of MRU is great than N and the LRU is less than N, it means that MRU is an *OS* macro page and LRU is an *OF* one. This is the simplest case. As shown in Fig. 8(a), the first step is to copy data C into the empty slot B. Since the MRU slot C is now in the on-package region, a new link, *B-to-C*, is updated in the translation table. However, because the link, *C-to-B*, is not ready yet, the P bit of this row is set to be '1'. The second step is to copy data B from the *Ghost* slot, Ω, to slot C. After this step finishes, the P bit is reset. Finally, after copying the LRU data, A, from slot A to slot Ω, slot A becomes the new empty slot.
- If the macro page ID of MRU is greater than N and the LRU is greater than N, it means that MRU is an *OS* macro page and LRU is an *MF* one. Fig. 8(b) shows the data movement in this case. The first two steps are the same as the ones in Fig. 8(a). In the third step, data A, which is currently stored in slot C, is copied to slot Ω, and after that row A in the translation table is remarked as "pending". Finally, data C is moved back to its original place, and after that the P bit is reset. During the last step, all the memory accesses to data A are no longer routed to slot C, but accesses to data C are still routed to slot A, because the P bit only prevents the address translation from A to C.
- If the macro page ID of MRU is less than N and the LRU is less than N, it means that MRU is an *MS* macro page and LRU is an *OF* one. In this case, as illustrated in Fig. 8(c), the first step is to copy data D into the empty slot C and update the translation table by adding a new link *C-to-D* but with the "pending" bit marked. The second step is to copy data B back to its original slot and set the entry in the translation table to be *B-to-B*. The third step is the same as the second step in Fig. 8(a) and Fig. 8(b). After that, the P bit is cleared. The fourth step is the same as the third step in Fig. 8(a).
- If the macro page ID of MRU is less than N and the LRU is greater than N, it means that MRU is an *MS* macro page and LRU is an *MF* one. This is the most complicated case because both the MRU and the LRU pages are migrated ones. Fig. 8(d) demonstrates the data movement in this case. Actually, the first 3 steps are the same as the ones in Fig. 8(c) and the last 2 steps are the same as the ones in Fig. 8(b).

Example

We use the case illustrated in Fig. 8(d) to describe how the *coldest* on-package data are swapped with the *hottest* off-package data. As shown in Fig. 8(d), before the swap is triggered, data A and B are already swapped with D and E, respectively. Consequently, A and B are *MS* pages, while D and E are *MF* pages. In addition, macro page C is the *Ghost* page since its data is actually stored in an off-package slot, called Ω. The MSR off-package page is B, and it should be moved back to its original location (slot B) in the on-package memory region. On the contrary, the LSR on-package page is D, and it should be moved back to its original location (slot D) in the off-package memory region. The entire swap procedure is performed in the following order:

1) Data E stored in slot B is moved to slot C, which is empty now.
2) Update the translation table. Change Row C from *C-to-empty* to *C-to-E*, and set the P bit of Row C.
3) Copy data B back to slot B. After this step, accesses to the MRU macro page, B, are already routed to on-

[3]This piece of macro page can be reserved by the hardware driver after booting the OS.

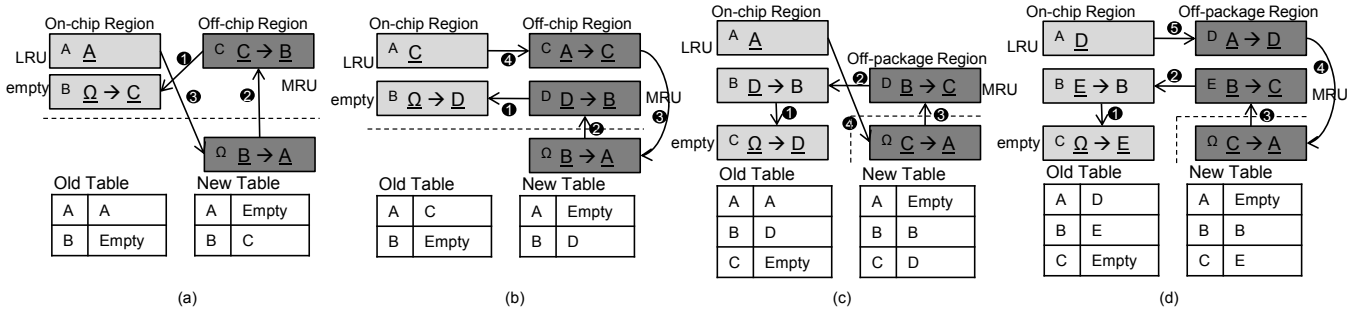

Fig. 8. Migration algorithm: (a) Data migration in the case that MRU$\geqslant N$ and LRU$< N$; (b) Data migration in the case that MRU$\geqslant N$ and LRU$\geqslant N$; (c) Data migration in the case that MRU$< N$ and LRU$< N$; (d) Data migration in the case that MRU$< N$ and LRU$\geqslant N$.

package regions.

4) Update the translation table. Change Row B from *B-to-E* to *B-to-B*.
5) Copy data C from slot Ω to slot E.
6) Update the translation table. Clear the P bit of Row C.
7) Copy data A from slot D to slot Ω.
8) Update the translation table. Set the P bit of Row A.
9) Copy data D from slot A to its original place – slot D.
10) Update the translation table. Change Row A from *A-to-D* to *A-to-empty*, and clear the P bit. Before this step is finished, accesses to data D are still routed to on-package regions.

In general, this algorithm makes sure that during the data migration procedure, the data under movement has two physical locations: one is in the on-package memory region and the other is in the off-package memory region. Thanks to the data duplication and the introduced P bit that blocks uncomplete bi-directional mapping, the program execution will not be halted since all the memory accesses are routed to an available physical location. By using this data movement algorithm, after the first step in Fig. 8(a) and Fig. 8(b) is completed or the first two steps in Fig. 8(c) and Fig. 8(d) are completed, the MRU macro page, which is previously stored in the off-package memory region, can start to experience fast access speed provided by the on-package memory. In addition, until the last step (in all four cases) is completed, the LRU macro page can still be fast accessed since it still has a copy in the on-package memory.

N-1 Mode with Live Migration

Although the *N-1 Mode* hides the migration latency by conservatively accessing the MRU macro page with off-package memory speed during the migration, we can further improve the algorithm by introducing the concept of *critical-data-first*. In the pure *N-1 Mode*, it takes time to bring the MRU page on chip. For example, when the macro page size is 4MB and the off-package interface is DDR3-1333, it takes $374\mu s$ to finish the first step as described in the algorithm of *N-1 Mode*. During this $374\mu s$, all the following accesses to the MRU page are still routed to off-package memory and slow. To address this issue, we divide the large data transaction into smaller chunks, such as 4KB. By doing so, the on-package memory can supply data to any request to the 4KB sub-blocks that have been already transferred while the rest of the swap is still ongoing in background.

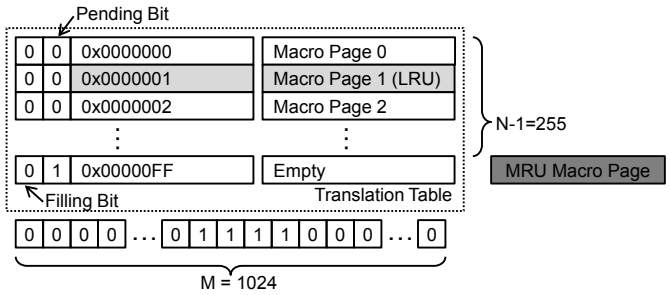

Fig. 9. The improved version of N-1 Mode with live migration support: The additional F bit indicates that the corresponding on-package slot is under data movement; the associated bit map indicates which sub-block is ready for accessing.

To implement this concept in hardware, we add another bit to each row in the translation table and a separated bit map. As shown in Fig. 9, the newly-added bit is called "F bit" (Filling bit). When the F bit is set, it means the corresponding on-package slot is loading data from off-package memory and this on-package slot is partially available. The additional bit map indicates which sub-block has already been moved to the on-package slot. When the F bit is set to '1', all the bits in the bit map are reset to '0'. However, when all the bits in the bit map become '1', the F bit is reset to '0' representing the data loading is completed.For example, if we use the macro page size of 4MB and sub-block size of 4KB, then there are 1,024 bits in the bit map. To support *critical-data-first*, the memory controller starts to copy the macro page from the position of the MRU sub-block and then wraps the address to the beginning.

By using this method, the migration overhead is further reduced. In Section IV, we compare the performance improvement achieved by the basic *N Mode* design, the *N-1 Mode* design, and this improved design, which we call *Live Migration*.

B. Data Migration Implementation

The migration algorithm can be implemented in either pure-hardware way or OS-assisted way depending on the migration granularity. Basically, the functionality of the data migration is achieved by keeping an extra layer of address translation that maps the physical address to the actual machine address. The pure-hardware scheme keeps the translation table in hardware while the OS-assisted scheme keeps it in software.

The pure-hardware solution is preferred when the macro page size is relatively large so that the scale of macro page

TABLE III
SIMULATION PARAMETERS AND WORKLOAD/TRACE DESCRIPTIONS

Memory system parameters			
Total memory capacity	4GB	Macro page size	from 4KB to 4MB
On-package memory capacity	512MB	Sub-block size	4KB
Workloads			
FT.C	FT contains the computational kernel of a 3D FFT-based spectral method.		
MG.C	MG uses a V-cycle MultiGrid method to compute the solution of a 3D scalar Poisson equation.		
SPEC2006 Mixture	The combination of four SPEC2006 workloads: gcc, mcf, perl, and zeusmp.		
pgbench	TPC-B like benchmark running PostgreSQL 8.3 with pgbench and a scaling factor of 100.		
Indexer	Nutch 0.9.1 indexer, Sun JDK 1.6.0 and HDFS hosted on one hard drive.		
SPECjbb	4 copies of SPECjbb 2005, each with 16 warehouses, using Sun JDK 1.6.0.		

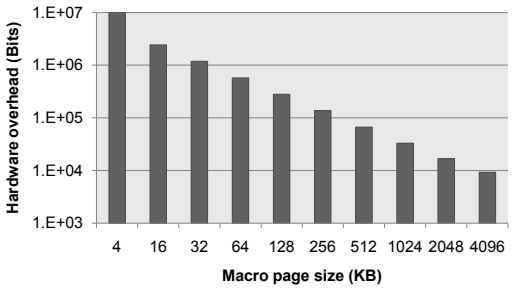

Fig. 10. The hardware overhead to manage 1GB on-package memory at different data migration granularity.

count is controllable within certain hardware resources. On the other hand, if finer granularity of data migration is required, the number of macro pages becomes too large for hardware to handle and OS-assisted scheme is used to track the access information of each macro page.

Pure-Hardware Solution

If the data migration is implemented by pure hardware, the major overhead comes from the translation table. In the case that the on-package memory size is 1GB and the macro page size is 4MB, the right column takes 26 bits per entry. Therefore, the total number of bits per entry is 28, including the P bit and the F bit. The cost of the entire translation table with 256 entries is 7,168 bits, and it is comparable to a TLB design in the conventional microprocessors. Since our proposed translation table has both RAM and CAM functions, we conservatively assume that it takes 2 additional clock cycles to complete one address translation.

Besides the translation table, there are two bit maps also incur the hardware overhead. The first bit map is used to store the filling status of each sub-block. In the 4MB macro page case, its size is 1,024 bits. The second bit map is used to record the LRU macro page with clock-based pseudo-LRU algorithm, which is used in real microprocessor implementation [17], and its size is 256 bits. The MRU policy is approximately implemented by multi-queue algorithm [18], by which we use three-level of queue with ten entries per level. Thus, the size of multi-queue is 780 bits. Hence, in total, the proposed pure-hardware scheme needs 9,228 bits to manage 1GB on-package memory at the 4MB granularity level, which is just a small overhead in today's microprocessor designs. Fig. 10 shows the number of bits required by the pure-hardware solution increases rapidly when reducing the macro page size. In this paper, we consider the pure-hardware solution is only feasible for the granularity larger than 1MB.

OS-Assisted Implementation

When the fined-granularity data migration is used, the physical-to-machine address translation has to be managed by software since the pure-hardware scheme has too much overhead as shown in Fig. 10. In the OS-assisted scheme, the data structure involved in the aforementioned algorithm is kept by the operating system, while the counters (i.e. the pseudo-LRU counter to determine the on-package LRU macro page and the multi-queue counter to determine the off-package MRU macro page) are still implemented in hardware. The translation table is updated by an OS periodical routine and the OS sends out data swapping instruction after each update. Besides the data movement latency, the extra overhead caused by OS involvement is on the same level of conventional TLB update overhead, which is mainly consumed by user-kernel mode switching and is about 127 cycles [19].

IV. TRACE-BASED ANALYSIS ON DYNAMIC MIGRATION

To evaluate the effectiveness of our proposed *hottest-coldest swap*, we compare the performance of *N Mode*, *N-1 Mode*, and *N-1 Mode with Live Migration* designs. We evaluate different methods in a conventional system primarily using trace-based simulation. In this section, we use traces rather than a detailed full-system simulator as we used in Section II because trace-based simulation makes it practical to process trillions of main memory accesses.

We collected the memory trace from a detailed full-system simulator [20] and the trace file records the physical address, CPU ID, time stamp, and read/write status of all main memory accesses. The workloads used to evaluate our designs includes FT.C and MG.C, which have large memory footprint as we have shown. Additionally, we developed a multi-programmed workload, *SPEC2006 Mixture*, by combining the traces from *gcc*, *mcf*, *perl*, and *zeusmp*. Furthermore, besides computation-intensive workloads, we also consider transaction-intensive workloads as well. Therefore, we add traditional server benchmarks (*pgbench* and *SPECjbb2005*) and a mixture of Web 2.0-based benchmarks (*indexer*). All of these workloads have memory footprints larger than 2GB.

In order to demonstrate how our proposed memory controller effectively leverage the on-package memory region, we limit the capacity of on-package memory space to be 512MB. As for the other parameters, we keep most of the assumptions that we have used in the previous sections. However, in this trace-based simulation, we model the detailed DRAM access latency by assuming FR-FCFS [11] scheduling policy and

TABLE IV
THE EFFECTIVENESS OF THE PROPOSED MEMORY CONTROLLER-BASED DATA MIGRATION IN REDUCING THE AVERAGE MEMORY ACCESS LATENCY.

Workload	FT.C	MG.C	pgbench	indexer	SPECjbb	SPEC2006 Mixture
DRAM core latency (cycles)	85.18	42.29	121.85	103.45	126.45	80.67
Latency w/o migration (cycles)	86.02	49.16	152.83	118.52	159.40	288.10
Best latency w/ migration (cycles)	83.20	42.23	124.27	105.55	135.60	82.45
Effectiveness	69.1%	84.3%	92.2%	86.1%	72.2%	99.1%

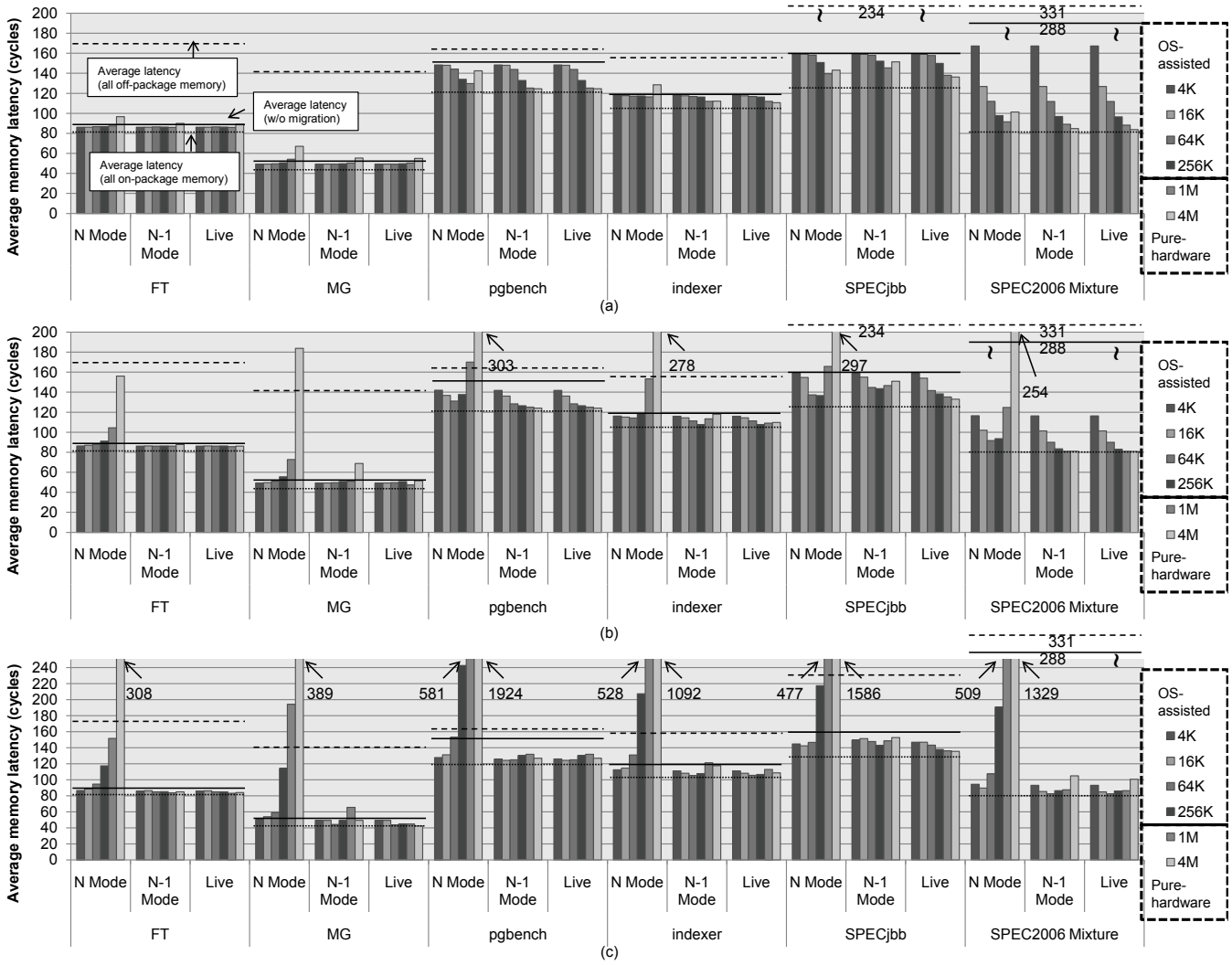

Fig. 11. The average memory access latency of various workloads by using different methods of data swapping: (a) swapping interval = 100K memory accesses; (b) swapping interval = 10K memory accesses; (c) the swapping interval = 1K memory accesses.

open page access. We use 8-bank structure for the off-package DRAM and 128-bank structure for the on-package DRAM. Therefore, the simulated DRAM access latency depends on the memory reference sequence, read/write distribution, row/bank locality, and whether the access is routed to the on-package or off-package region. In addition, the macro page granularity under the memory controller management ranges from 4KB to 4MB, and the sub-block size for live migration is 4KB. Due to the hardware overhead concern, OS-assisted scheme is used for macro pages smaller than 1MB and pure-hardware scheme is used for macro pages larger than 1MB (including 1MB).

Swap interval is another parameter that has a large impact on the effectiveness of our proposed hardware-based data migration scheme. Therefore, we run the trace-based simulation by varying the swapping interval number. We compare the random memory access latency on average by triggering the swap operation after each 100,000, 10,000, and 1,000 memory accesses, respectively. When the data swap activity becomes too frequent, the existence of P bit and F bit prevents triggering another swap if the previous swap is not complete yet.

A. N Mode, N-1 Mode, and Live Migration

Fig. 11 shows the effective memory access latencies of using different swapping intervals and the comparison among different migration algorithms. Observing the simulation result when the data granularity is large (i.e. 4MB), it is obvious to see that the straightforward method, *N Mode*, is not practical because it needs to move a large chunk of data without hiding

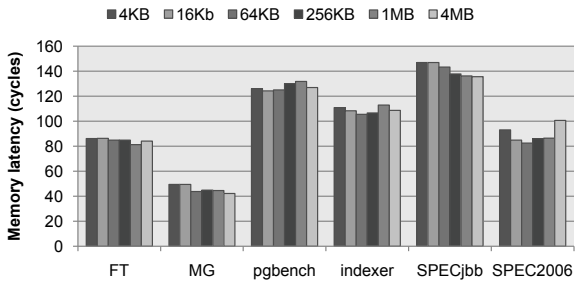

Fig. 12. The average memory latency (swapping interval = 1K memory accesses).

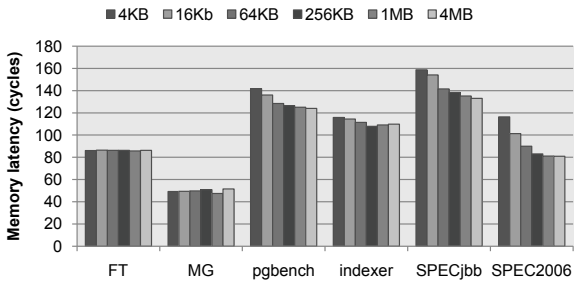

Fig. 13. The average memory latency (swapping interval = 10K memory accesses).

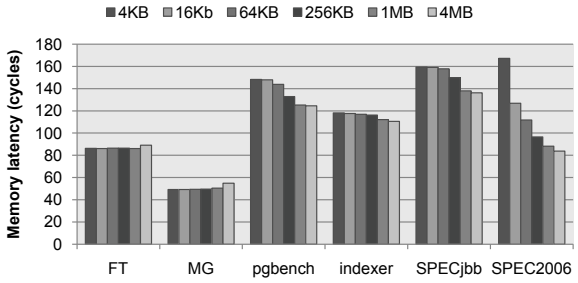

Fig. 14. The average memory latency (swapping interval = 100K memory accesses).

any latency. Hence, in this design, when the swap frequency is high (i.e. once per 1K or 10K memory accesses), the migration overhead offsets the benefit. Therefore, in order to hide the migration overhead, *N-1 Mode*, which sacrifices one slot in the translation table, should be used to overlap data migration with computation. In addition, *live migration*, which cuts the macro page into smaller pieces, can further hide the migration overhead by finer-granularity overlapping and reduce the average memory access latency by 5.2%.

However, when the data migration granularity is small (i.e. 4KB), the difference among *N Mode*, *N-1 Mode*, and *live migration* becomes negligible. This is because the overhead of swapping two small macro pages is trivial and N Mode has one more slots in the translation table as its name implies.

B. Migration Granularity and Frequency

The migration granularity and the migration frequency are the two key factors affecting the effectiveness of the proposed heterogenous main memory space. To study the impact of these two factors, we use *live migration* algorithm with different macro page sizes and swapping intervals, and Fig. 12 to Fig. 14 illustrate the result. The comparison shows that the migration frequency is more important since the minimum memory latencies in Fig. 12 are smaller than those in Fig. 13 and Fig. 14. Another observation is that different types of workloads have different favorites on the migration granularity and the optimal migration granularity also depends on the migration frequency. While the *live migration* can make it feasible to migrate data across the package boundary without incurring too much overhead, it is necessary for the memory controller to adaptively change the migration granularity according to different types of workloads. In this work, we propose to use an OS-assisted approach for fine-granularity and a pure-hardware approach for coarse-granularity as described in the previous sections.

In order to demonstrate the effectiveness of the proposed heterogenous memory space and its management algorithms in a more generalized way, we define the effectiveness, η, as follows,

$$\eta = \frac{\text{Latency w/o migration} - \text{Latency w/ migration}}{\text{Latency w/o migration} - \text{DRAM core latency}} \times 100\%$$

And, this metric approximately reflects how many memory accesses are routed to the on-package memory region.

Note that the average effectiveness is 83% as listed in Table IV, while the on-package memory size is 512MB, which is only 12.5% of the 4GB memory space in this case.

C. Sensitivity Analysis on On-Package Memory Capacity

In order to further demonstrate how our proposed heterogeneous main memory with migration feature enabled can efficiently leverage the on-package memory resources, we conduct a sensitivity analysis by reducing the on-package memory capacity from 512MB to 128MB. As expected, the average memory access latency is increased because it becomes harder to keep as many as data on package when the on-package memory size is reduced. However, as illustrated in Fig. 15, the average latency is still much shorter than the latency without dynamic data migration, while the on-package memory capacity is reduced from 512MB to 128MB.

D. Power Evaluation

Fig. 16 shows the total memory power comparison between using hybrid on- and off-package DRAM with dynamic migration and only using off-package DRAM. We assume 5pJ/bit for both on- and off-package DRAM core access, 1.66pJ/bit for on-package interconnect, and 13pJ/bit for off-package interconnect [21]. The memory power overhead caused by crossing-package migration depends on the migration interval. The minimum power overhead we observe is about 2X, which occurs when the migration interval is once per 100K memory accesses and the migration granularity is 4KB.

V. Related Work

A large body of prior work has examined how to use on-chip memory controllers and data migration schemes to improve the system performance.

A. On-Chip Memory Controller

The recent trends with the inclusion of on-chip memory controllers in modern microprocessor designs help eliminate

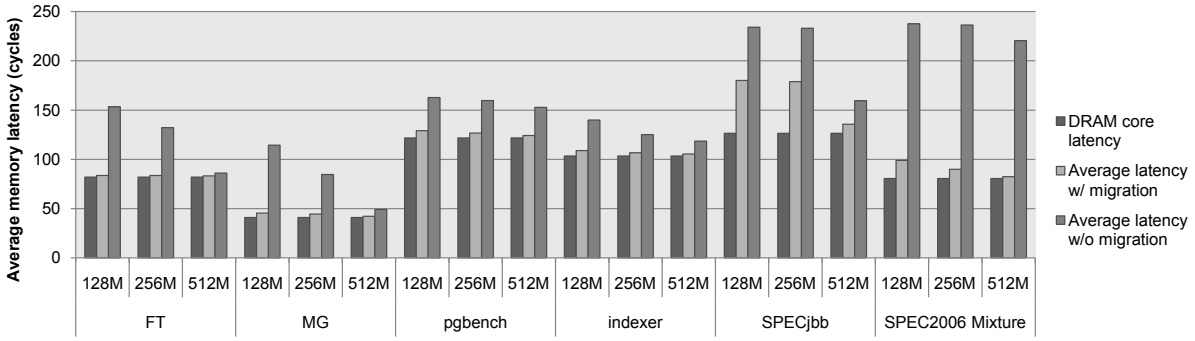

Fig. 15. The average memory access latency of the heterogeneous main memory under different sizes of on-package memory.

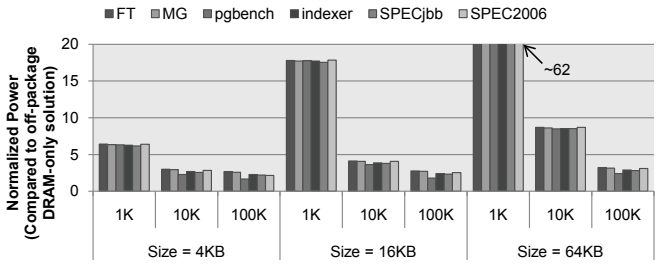

Fig. 16. The relative memory power when using heterogenous on- and off-package DRAM with dynamic migration and only using off-package DRAM.

the system-bus bottleneck and enable high-performance interface between a CPU and its DRAM DIMMs [22]. In addition, integrating the memory controller on chip also makes it possible to add an extra level of address indirection to remap the data structure in memory and to improve cache and bus utilization [23]. However, our work proposes a system architecture where the memory controller is tightly integrated with TLBs, which is seldom available in the contemporary computer market.

Many previous DRAM scheduling policies were proposed to improve the memory controller throughput [24], [25]. Stream pre-fetchers are also commonly used in many processors [26], [27] since they do not require significant hardware cost and work well for a larger number of applications. In addition, prefetch-aware DRAM controller was proposed [28] to maximize the benefit and minimize the harm of prefetching. Our work is orthogonal to these mechanisms: while aggressive prefetching is effective in tolerating memory latency by moving more useful data to caches, our work focuses on leveraging fast regions in the heterogeneous memory space to hide the impact of low performance regions.

B. Non-Uniform Access

NUMA (non-uniform memory access) architecture is used in many commercial settings of SMP (symmetric multiprocessing) clusters, such as those based on AMD Opteron and Alpha EV7 processors. In the ccNUMA architecture [29], each processor or processor cluster maintains a cache system to reduce internode traffic and average latency to non-local data. The Fully-Buffered DIMM [30] is another example of NUMA which can be configured to behave as a uniform architecture by forcing all DIMMs to stall and emulate the slowest one.

For large caches where interconnect latency dominates the total access latency, a partitioned Non-Uniform Cache Access (NUCA) architecture is adopted [31]. To improve the proximity of data and computation, dynamic-NUCA policies have been actively studied in recent years [12]–[14], [32]. Although dynamic-NUCA method helps performance, the complexity of the search mechanism together with the high cost of communication makes it not scalable.

Our proposed heterogeneous main memory can also be treated as a NUMA system. Although on-chip and off-chip memory regions have different access speeds, both of them are local to the accessing processor, hence cache coherence is not an issue in this scenario. The previous research closest to our work is from Ekman and Stenstrom [33], in which a multi-level exclusive main memory subsystem is proposed. In their work, pages can be gradually paged out to disk through multiple levels of memory regions, but proactive data migration from lower-tier to higher-tier memory regions is not allowed. Different from most of the previous work, our approach introduces a new layer of address translation that is handled by the on-chip memory controller.

VI. Conclusion

SiP and 3D integration are promising to bring more memory cells onto microprocessor package to mitigate the "memory wall" problem. In this paper, instead of using them as caches, we studied the architecture of using the on-package memory cells as a portion of the main memory working in partnership with a conventional main memory implemented with DIMMs. To manage this heterogeneous main memory containing data on both on-package and off-package regions, we introduced another layer of address translation. While the conventional physical address makes the OS paging system keep intact, the newly-added machine address represents the actual data location on the DRAM chips. By manipulating the physical-to-machine translation, our proposed on-package memory controller design can dynamically migrate data across the chip boundary. Compared to using on-package memory as caches, the heterogenous main memory approach does not pay an extra penalty for tag access before data access and for cache misses. The evaluation results demonstrate how the heterogeneous main memory can use the on-package memory efficiently and achieve the effectiveness of 83% on average.

REFERENCES

[1] B. Black, M. Annavaram, N. Brekelbaum, J. DeVale, L. Jiang *et al.*, "Die Stacking (3D) Microarchitecture," in *MICRO '06*, 2006, pp. 469–479.

[2] G. H. Loh, Y. Xie, and B. Black, "Processor Design in 3D Die-Stacking Technologies," *IEEE Micro*, vol. 27, no. 3, pp. 31–48, 2007.

[3] International Technology Roadmap for Semiconductors, "ITRS 2009 Edition," http://www.itrs.net/.

[4] T. Kgil, A. Saidi, N. Binkert, S. Reinhardt, K. Flautner *et al.*, "PicoServer: Using 3D stacking technology to build energy efficient servers," *ACM Journal on Emerging Technologies in Computing Systems*, vol. 4, no. 4, pp. 1–34, 2008.

[5] G. H. Loh, "3D-Stacked Memory Architectures for Multi-core Processors," in *ISCA '08*, 2008, pp. 453–464.

[6] G. L. Loi, B. Agrawal, N. Srivastava, S.-C. Lin, T. Sherwood *et al.*, "A Thermally-Aware Performance Analysis of Vertically Integrated (3-D) Processor-Memory Hierarchy," in *DAC '06*, 2006, pp. 991–996.

[7] M. Ghosh and H.-H. S. Lee, "Smart Refresh: An Enhanced Memory Controller Design for Reducing Energy in Conventional and 3D Die-Stacked DRAMs," in *MICRO '07*, 2007, pp. 134–145.

[8] P. S. Magnusson, M. Christensson, J. Eskilson, D. Forsgren, G. Hållberg *et al.*, "Simics: A Full System Simulation Platform," *Computer*, vol. 35, no. 2, pp. 50–58, 2002.

[9] S. Thoziyoor, J. H. Ahn, M. Monchiero, J. B. Brockman, and N. P. Jouppi, "A Comprehensive Memory Modeling Tool and Its Application to the Design and Analysis of Future Memory Hierarchies," in *ISCA '08*. IEEE Computer Society, 2008, pp. 51–62.

[10] Micron, "2Gb: x4, x8, x16 DDR3 SDRAM," http://www.micron.net/.

[11] S. Rixner, W. J. Dally, U. J. Kapasi, P. Mattson, and J. D. Owens, "Memory Access Scheduling," in *ISCA '00*, 2000, pp. 128–138.

[12] B. M. Beckmann, M. R. Marty, and D. A. Wood, "ASR: Adaptive Selective Replication for CMP Caches," in *MICRO '06*, 2006, pp. 443–454.

[13] Z. Chishti, M. D. Powell, and T. N. Vijaykumar, "Optimizing Replication, Communication, and Capacity Allocation in CMPs," in *ISCA '05*, 2005, pp. 357–368.

[14] J. Huh, C. Kim, H. Shafi, L. Zhang, D. Burger *et al.*, "A NUCA Substrate for Flexible CMP Cache Sharing," in *ICS '05*, 2005, pp. 31–40.

[15] N. Rafique, W.-T. Lim, and M. Thottethodi, "Architectural Support for Operating System-Driven CMP Cache Management," in *PACT '06*, 2006, pp. 2–12.

[16] S. Cho and L. Jin, "Managing Distributed, Shared L2 Caches through OS-Level Page Allocation," in *MICRO '06*, 2006, pp. 455–468.

[17] "UltraSPARC T2. Supplement to the UltraSPARC Architecture 2007," Sun Microsystems, Inc., Tech. Rep. 950-5556-01, 2007.

[18] G. Loh, "Extending the Effectiveness of 3D-Stacked DRAM Caches with an Adaptive Multi-Queue Policy," in *MICRO '09*, 2009, pp. 201–212.

[19] J. Liedtke, "Improving IPC by kernel design," in *SOSP '93*, 1993, pp. 175–188.

[20] E. Argollo, A. Falcón, P. Faraboschi, M. Monchiero, and D. Ortega, "COTSon: Infrastructure for Full System Simulation," *ACM SIGOPS Operating Systems Review*, vol. 43, no. 1, pp. 52–61, 2009.

[21] K. Fukuda, H. Yamashita, G. Ono, R. Nemoto, E. Suzuki, T. Takemoto, F. Yuki, and T. Saito, "A 12.3mW 12.5Gb/s Complete Transceiver in 65nm CMOS," in *ISSCC '10*, 2010, pp. 368–369.

[22] W.-F. Lin, S. K. Reinhardt, and D. Burger, "Reducing DRAM Latencies with an Integrated Memory Hierarchy Design," in *HPCA '01*, 2001, pp. 301–312.

[23] L. Zhang, Z. Fang, M. Parker, B. K. Mathew, L. Schaelicke *et al.*, "The Impulse Memory Controller," *IEEE Transactions on Computers*, vol. 50, no. 11, pp. 1117–1132, 2001.

[24] Z. Zhu and Z. Zhang, "A Performance Comparison of DRAM Memory System Optimizations for SMT Processors," in *HPCA '05*, 2005, pp. 213–224.

[25] E. Ipek, O. Mutlu, J. F. Martínez, and R. Caruana, "Self-Optimizing Memory Controllers: A Reinforcement Learning Approach," in *ISCA '08*, 2008, pp. 39–50.

[26] H. Q. Le, W. J. Starke, J. S. Fields, F. O'Connell *et al.*, "IBM POWER6 Microarchitecture," *IBM Journal of Research and Developement*, vol. 51, no. 6, pp. 639–662, 2007.

[27] S. Sharma, J. G. Beu, and T. M. Conte, "Spectral Prefetcher: An Effective Mechanism for L2 Cache Prefetching," *ACM Transactions on Architecture and Code Optimization*, vol. 2, no. 4, pp. 423–450, 2005.

[28] C. J. Lee, O. Mutlu, V. Narasiman, and Y. N. Patt, "Prefetch-Aware DRAM Controllers," in *MICRO '08*, 2008, pp. 200–209.

[29] J. Laudon and D. Lenoski, "The SGI Origin: A ccNUMA Highly Scalable Server," in *ISCA '97*, 1997, pp. 241–251.

[30] B. Ganesh, A. Jaleel, D. Wang, and B. Jacob, "Fully-Buffered DIMM Memory Architectures: Understanding Mechanisms, Overheads and Scaling," in *HPCA' 07*, 2007, pp. 109–120.

[31] C. Kim, D. Burger, and S. W. Keckler, "An Adaptive, Non-Uniform Cache Structure for Wire-Delay Dominated On-Chip Caches," in *ASPLOS '02*, 2002, pp. 211–222.

[32] J. Chang and G. S. Sohi, "Cooperative Caching for Chip Multiprocessors," in *ISCA '06*, 2006, pp. 264–276.

[33] M. Ekman and P. Stenstrom, "A Cost-Effective Main Memory Organization for Future Servers," in *IPDPS '05*, 2005, pp. 45–54.