DMR3D: Dynamic Memory Relocation in 3D Multicore Systems

Dean Michael Ancajas Koushik Chakraborty Sanghamitra Roy

USU BRIDGE LAB, Electrical and Computer Engineering, Utah State University
dbancajas@gmail.com, {koushik.chakraborty, sanghamitra.roy}@usu.edu

ABSTRACT

Three-dimensional Multicore Systems present unique opportunities for proximity driven data placement in the memory banks. Coupled with distributed memory controllers, a design trend seen in recent systems, we propose a Dynamic Memory Relocator for 3D Multicores (DMR3D) to dynamically migrate physical pages among different memory controllers. Our proposed technique avoids long interconnect delays, and increases the use of vertical interconnect, thereby substantially reducing memory access latency and communication energy. Our techniques show 30% and 25% average performance and communication energy improvement on real world applications.

1. INTRODUCTION

A key factor limiting the system performance stems from the increasing gap between the processor and memory speed. Recent fabrication techniques such as 3D die stacking seek to solve this problem by bringing the memory physically closer to the processor. 3D die-stacking is used to bond multiple wafers in a vertical manner using low-latency and high bandwidth vertical interconnects [5, 7], allowing faster communication between the processor and the memory.

Several previous works evaluate the potential benefits of 3D-stacking in system performance. Loh studied aggressive memory configurations that take advantage of a 3D processor-memory setup [12]. Other studies evaluated the benefit of placing memory directly on top of the processor and have reported huge performance speedups [11, 13]. However, many of these studies considered systems with a centralized memory controller (MC). In a traditional 2D system, this setup was suitable because of the limited pin count and the slow off-chip communication interface, which dominates the memory latency. On the contrary, such design limitations are not present in 3D systems because of the abundance of low latency Through Silicon Via (TSV) interconnects. Moreover, with the increasing core count in multicore systems, we can expect MCs to grow in number to accommodate the bandwidth needed. This idea has been recently incorporated in several traditional 2D systems [2, 21].

DAC '13, May 29 - June 07 2013, Austin, TX, USA.

In this work, we identify a new problem that arises in 3D systems with multiple MCs: the problem of suboptimal data placement that precludes the performance brought about by 3D die stacking. In a 3D system with distributed MCs, data located on a memory bank directly above the requesting processor will experience a smaller delay as compared to data that needs to traverse an interconnect system. A typical modern on-die interconnect such as HyperTransport [9] still requires 100+ ns in order to send 8 bytes of data. Meanwhile, vertical interconnect delays using TSVs and 3D dram access times are just a small fraction of this at 12ps [13] and 7ns [19], respectively. Therefore, *we can achieve 10x access latency improvement if data is directly accessed through TSVs rather than being transferred using the interconnect.*

In a 3D Multicore, the placement of data in a memory bank and the location of the processing core determines if that data can be accessed through the TSV or if it must traverse the interconnect. However, current system designs are agnostic about this critical consideration, thereby substantially undermining the potential performance and energy efficiency in a 3D Multicore. In this work, we study hardware mechanisms that alleviate the interconnect problem by remapping data to an optimal location in a 3D Multicore System. Our contributions are as follows:

- We show that the data mapping policy of a modern OS can hurt performance of 3D multicore systems with multiple MCs. Our analysis with a detailed full system simulation for real world applications shows 117–280% (average 170%) performance improvement if all non-local accesses are transformed into local accesses as a result of efficient data placement (Section 3).
- We propose DMR3D, a hardware technique that minimizes non-local accesses by identifying critical memory pages and migrating them to a closer memory bank. We present two algorithms for DMR3D: a *Global Scheme* that dynamically allocates migration slots to different threads and a *Thread On-Demand Scheme* that statically allocates equal migration slots to individual threads (Section 4).
- We perform a thorough evaluation of our two DMR3D schemes using a state-of-the-art full system simulator (Section 5). Our best performing scheme increases performance by 7–72% (ave: 30%), increases local accesses by 9–95% (ave: 50%) and improves communication energy by up to 48% (ave: 25%) (Section 6) compared to the baseline. We also compare our schemes with a representative scheme for NUCA caches (Victim Replication [22]), and observe an average of 33% improved

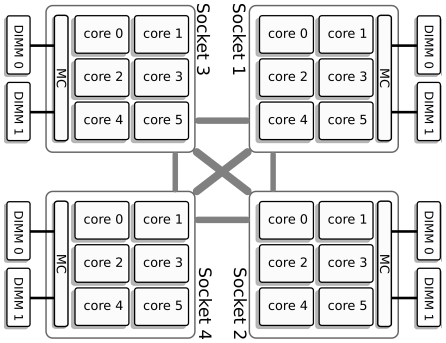


Figure 1: Logical organization of HP G7 series composed of AMD Istanbul processors. The processors communicate through point-to-point Direct Connect Architecture. DIMMs attached to each socket are local memory.

performance.

2. BACKGROUND

In this section, we first describe a modern Non-Uniform Memory Access (NUMA) system. We then introduce our 3D NUMA processor-memory system based on recent work on 3D integration.

Modern server machines typically distribute work to several chip multiprocessors. Because each CMP is composed of multiple aggressive cores, recent designs [2,21] have included a memory controller (MC) on-chip in order to satisfy increasing bandwidth demands. In this setup, the entire memory address space is equally divided among all MCs such that independent requests to different memory sections can be serviced simultaneously. Figure 1 shows an example of this setup. In the figure, processors accessing memory locations residing in a different socket will experience a longer delay compared to the ones which reside in the same socket (also known as local access). Previous studies estimate the non-local access to be 33%-100% longer than the local access [15, 20].

2.1 Baseline System Organization

In our simulation setup, we model a 64-bit processor system with 8GB of total addressable main memory. As depicted in Figure 2, we have a 3D system with the first layer containing all 8 processors along with four memory controllers connected by a ring network interconnect. The second and third layers contain the DRAM cells. Each node in our NUMA system is composed of two processors, one MC and two memory ranks in the topmost two layers.

We recognize three different access types based on source-destination proximity: *local*, *neighbor* and *remote*. Figure 3 shows an example of how different nodes interact with each other. The most basic type of access is when a processor requests data that belongs to the same node (Figure 3a). In Figure 3b, if a processor in node 1 requests data which resides in local memory of node 2, it requires a single hop in the interconnect network. We refer to this request as *neighbor*. Consequently, a processor in node 1 requesting data situated in the local memory of node 3 would now require 2 hops (Figure 3c). We refer to this type of access as *remote*.

2.2 DRAM organization

The 8GB main memory is composed of eight 1 GB ranks. Data bus width is 64-bits. Each rank is constructed using

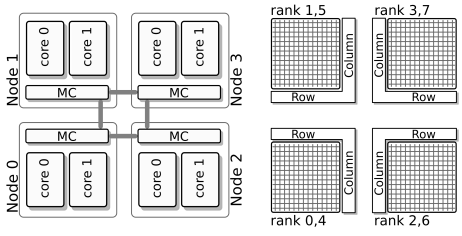


Figure 2: 3D setup used in this study.

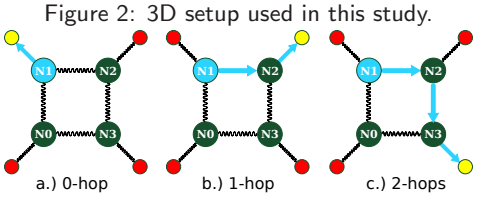


Figure 3: Different Request Distances.

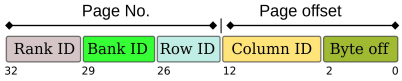


Figure 4: DRAM Address Mapping Policy.

4 128 Mbit, x16 DDR3 devices. We based our DRAM device model on Micron MT41J128M16 [1]. Timing details are indicated in Table 2 of Section 5. The address mapping policy that we used is shown in Figure 4 and is similar to Intel 845G Memory Controller. With this mapping, all data from a memory page being worked by a particular thread is always located in the same *rank*, *bank* and *row* which allows easier migration of data from one bank to another.

3. MOTIVATION

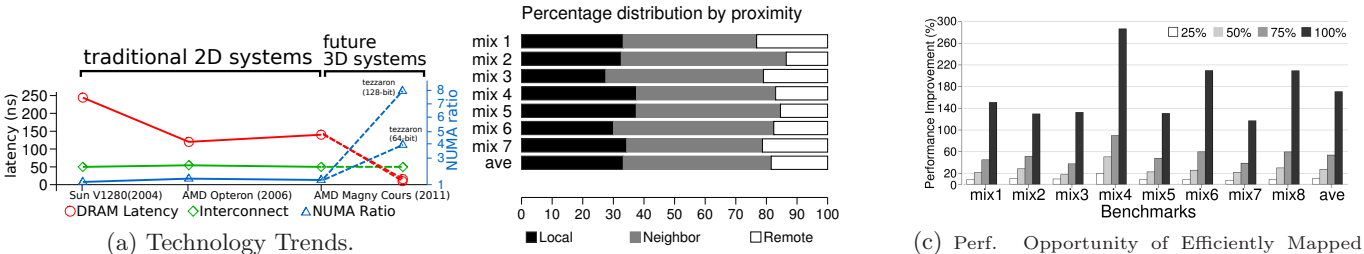
In this section, we first discuss why current technological trends demand a more efficient mapping of data. Then, we analyze how data from different programs are mapped across the memory banks. Along with the results of our motivational study, we will argue why such intelligent mapping policies are desired.

3.1 Effects of 3D integration on NUMA ratio

In current multicore systems, the NUMA ratio[1] is around 1.5 [15]. With 3D integration, this ratio will further increase because on-chip 3D DRAMs will be naturally faster while interconnect latency in the processor layer will remain the same. To explain this point clearly, we gathered values of different NUMA system parameters from three machines manufactured in the last decade (2004, 2006, 2011). We combine data from the work of [3], [16] and Tezzaron 3D DRAMs [19] to estimate NUMA ratios of future 3D systems. Both [3] and [16] use the same benchmarking methodology so we were able to combine their data with little effort.

To get the main memory latency values, both studies ran a program that strides through an array, the stride size is then increased until all accesses effectively miss the last level of cache. The interconnect latency is then the difference between the remote and local access latency. We consolidate all this data and show it in Figure 5(a). The first two data points from the graph (V1280 and Opteron) are taken from [3]'s work, while the third one (MagnyCours) is taken from [16]'s study. Our estimates for these parameters in a 3D

[1]NUMA ratio is the ratio between the remote and local memory latency



(a) Technology Trends.

(b) Breakdown of Mem Access Types.

(c) Perf. Opportunity of Efficiently Mapped Data.

Figure 5: Technology Trends, Access Profile and Performance Opportunity in 3D Systems.

multicore system are sketched in broken lines.

3.1.1 NUMA ratio

The NUMA ratio can be calculated using two parameters: interconnect and DRAM access latency. Local memory requests only experience DRAM latency while non-local requests include interconnect latency. The increase in NUMA ratio in 3D systems is largely driven by the inability of interconnects to scale well with 3D DRAM latency. We show dramatic leaps in the value of this parameter in Figure 5(a). Going from a 2D system to a 3D, the penalty of a non-local access increases by 700%, or to a ratio of 8. Next, we discuss how trends in interconnects and DRAMs drive this parameter to enormous values.

3.1.2 Interconnect

One notable difference from the interconnect used by the three machines in Fig. 5(a) is that both Opteron and V1280 use **off-chip** communication while MagnyCours uses **on-chip** communication. However, looking at the interconnect delay in Figure 5(a), the values are almost the same. This poor scaling of interconnect delay primarily stems from the long RC delays within a given chip, which dominates most of the communication through the interconnect fabric.

3.1.3 DRAM Latency

3D DRAM modules use TSVs to reduce latency and increase bandwidth. As such, the latency of 3D DRAM modules are substantially smaller compared to that of its 2D counterparts. Prototypes of 3D DRAMs from Tezzaron have access times of 7ns [19]. We used data sheets from Tezzaron to calculate latency to get a single cache line, both for 64 and 128-bit versions of DRAM. These values are shown as two estimations in the DRAM latency curve in Figure 5(a).

3.2 Workload Access Patterns

In the light of the trends shown above, we perform workload characterization of modern benchmarks on our setup (discussed in Section 5) to determine the amount of non-local accesses in each benchmark. Figure 5(b) shows the distribution of memory accesses of programs composed of high memory bandwidth SPEC 2006 programs. From the figure, workload 3 (*mix3*), about 25% of its accesses are local while 75% of its memory accesses incur large interconnect latencies. On an average, for all workloads, about 68% of the memory accesses are to non-local nodes.

3.3 Performance Opportunity in 3D Systems

From these results, we also conducted another experiment to see the performance improvement if fractions of the non-local requests were transformed to local requests as a result of a more efficient data mapping. For this experiment, we assume the NUMA ratio to be 8. Figure 5(c) shows the result of this study. The four bars represent the performance improvement if fractions of non-local memory requests were routed locally. For instance, if all non-local accesses of *mix4* were routed to the local memory as a result of migration, its performance would improve by 280%. As a whole, there is an average improvement of 10%, 27%, 54% and 171% if we can transform 25%, 50%, 75% and 100% of non-local accesses into local accesses, respectively.

The results of our motivational study show a tremendous opportunity for performance improvement. We now discuss our proposed techniques to realize this opportunity in a system design.

4. DESIGN OVERVIEW

In this section, we discuss the design of *Dynamic Memory Relocator for 3D Multicores* (DMR3D). We first give a brief overview of our design, discuss two schemes of implementing DMR3D, and then explain hardware implementation overheads.

The key principle of DMR3D is identifying and managing data that needs to be swapped to increase the relative percentage of local requests in the whole system. To this end, DMR3D creates an online access profile of data from different memory banks to determine which data needs to be migrated. At certain time intervals (hereafter referred to as *epoch*), a DMA copy is issued to swap data between memory banks. We propose two DMR3D schemes that differ in the way the profiling and migration of data is done. The first scheme (*Global*) relies on a global epoch to synchronize profiling and migration of data, while the second (*Thread On-Demand*) uses a per-thread epoch.

We now discuss the basic working principle of our proposed DMR3D (Section 4.1), two proposed schemes (Sections 4.2 and 4.3), and the corresponding overheads (Section 4.5).

4.1 Hardware Structures in DMR3D

DMR3D is composed of two major hardware structures: the *Profile Table* (PT) and the *Address Remapping Table* (ART). Both of these tables are added to each memory controller. The PT is used to identify access patterns of threads while the ART is used as a layer of indirection for memory pages that were remapped.

To explain DMR3D in detail, we provide an example of the sequence of events that happen during a memory request in Figure 6. In this example, it is assumed that the system

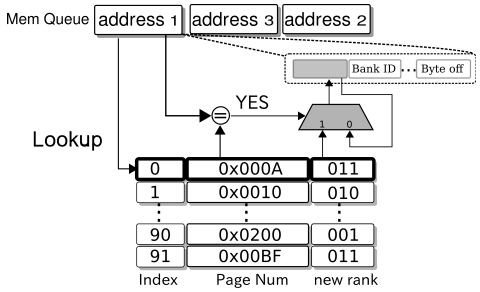


Figure 6: Mechanism of Address Remapping Table.

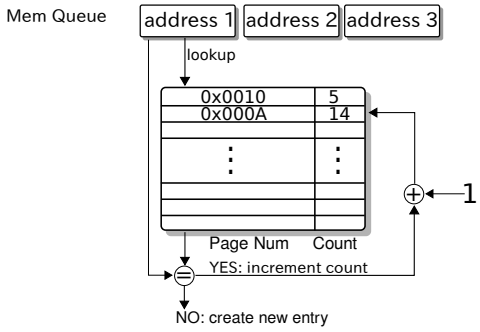


Figure 7: Profile Table Mechanism.

has already swapped some data in different memory banks. When a memory request is added in the memory queue, its page number is scanned in the ART. If there is a hit, the address in the memory queue is modified to reflect the local location of the data. Otherwise, the request is immediately redirected to a non-local MC (not shown in figure) or stays in the queue if it is local to the node.

There are also cases when an originally local address is now mapped to a non-local bank because of a data migration initiated by a non-local MC. For correctness issue, we handle these cases by storing such mappings in ARTs of MCs involved in the swap.

DMR3D reshuffles data between different nodes by changing certain sections of the memory address. Since the memory *ranks* are distributed uniformly across all nodes, DMR3D relies on this information to construct the new address of memory pages. The exact mechanisms for address translation and migration are discussed in Section S5.

The next key structure in DMR3D is the PT that is used to create an online access profile of memory addresses requested by the running threads. As the whole execution is divided into several *epochs*, DMR3D uses the profile of a current epoch to anticipate memory accesses in the next *epoch* by exploiting the temporal and spatial locality property of programs. Shown in Figure 7 is the mechanism behind the use of the PT. For each request received in the memory controller, the address is analyzed using the PT to get insight on which memory pages are accessed the most by the running programs. The PT contains the page numbers and its access count that are used to guide data migration at the end of every *epoch*.

While data migration is being done, accesses to these remote data will still be serviced by the non-local MC. Once migration finishes, the ART table is then populated with new address mappings which are then used by their respective memory controllers.

4.2 Proposal 1: Global Scheme (GS)

In the Global scheme, the migration of data is done at constant time intervals. The main motivation of this scheme is the fact that different programs running on a multicore present different demands to the memory system. This scheme gives more priority to threads that access the memory more and less to threads that do not, particularly CPU-bound threads. GS improves the performance by devoting more migration opportunities to memory-intensive threads that are severely crippled by the interconnect latency.

At the start of each *epoch*, GS will migrate the top 200 pages servicing non-local requests and migrate them at appropriate locations. In this work, we chose a 1 million cycle *epoch* size and 200-page size migration in each epoch. We also vary these two parameters to see their effects on DMR3D (See Section S6 for our design space exploration).

Parameter	Value
MP Size and Freq	8-core, 2Ghz
Re-order Buffer	64 entries
Fetch/Dispatch	4/cycle
Exec/Commit	4/cycle
L1 I-cache	32 KB/4-way, private, 2-cycle
L1 D-cache	48 KB/4-way, private, 2-cycle
L2 cache	256 KB/8-way, shared, 16-cycle.
Cache Line	64 Bytes

Table 1: NUMA Parameters

4.3 Proposal 2: Thread on-demand Scheme (TODS)

While GS focuses on improving the overall system locality by giving more migration opportunities to memory-intensive threads, TODS allocates equal migration opportunities for each running thread in the system. In TODS, the threads strive to maintain a specific ratio (threshold) between their local accesses and non-local accesses. Once this threshold has been crossed, the MC then triggers migration in order to satisfy the threshold requirement. We discuss the motivation behind TODS in more details in Section S2.

4.4 Cache Coherency Issues

During page relocation, coherent data could be moved around and might be corrupted if not synchronized properly. As such, any pending writes to a memory location being migrated are stalled in the remote MC until the data has been completely transferred. The write requests, along with succeeding queued requests are then forwarded to the new MC. Read requests on a memory location being transferred are serviced by the original MC.

4.5 Migration Overhead

The performance increase resulting from the use of DMR3D is based on transferring memory pages to a physically closer bank when they are likely to be referenced again soon, thus avoiding the repeated cost of interconnects. We assume the presence of an overlaid DMA channel responsible for moving data around DRAM banks, similar to [6] and [18].

Other pertinent overheads such as the sizing of PT, ART and the latencies are discussed in Section S4.

5. METHODOLOGY

Our detailed full system simulator is built upon the Windriver Simics [14] platform. Important parameters of our system are shown in Table 1. We modeled a modern multicore system composed of 8 superscalar processors with a

Parameter	Value
Device	Micron MT41J128M16 [1] DDR3
Configuration	1 rank/DIMM, 64-bit channel, 4 devices/DIMM
Clock	2 Ghz
Timings	$t_{CCD} = t_{RCD} = 20$ns
DRAM Capacity	8 GB
NUMA ratio	4 (350 cpu cycles)

Table 2: Memory System

Name	Threads
Mix 1	bwaves-cactus-mcf-gems-lbm-zeusmp-sjeng-leslie
Mix 2	bwaves-bwaves-cactus-cactus-mcf-mcf-gems-gems
Mix 3	leslie-leslie3d-sjeng-sjeng-zeusmp-zeusmp-lbm-lbm
Mix 4	namd-gromacs-leslie-bwaves-cactus-sjeng-mcf-namd
Mix 5	milc-zeusmp-sjeng-sjeng-leslie-leslie-namd-namd
Mix 6	milc-milc-namd-namd-lbm-lbm-mcf-mcf
Mix 7	zeus-zeus-milc-gems-gems-mcf-gromacs-bwaves
Mix 8	namd-namd-gromacs-gromacs-milc-milc-sjeng-sjeng

Table 3: Workload Mix

detailed memory system, DRAMSim2 [17]. All aspects of DRAM device operation are accurately modeled using state machines. Since the main memory is now placed on chip, we run it at a higher clock speed but all timing parameters remain the same. The parameters of our memory system are shown in Table 2.

Our two schemes are evaluated using full system simulation of modern benchmark programs. We combined individual programs from SPEC 2006 suite to construct memory-intensive multiprogram workloads. The threads were chosen according to the profile of [8]. We ran all benchmarks to approximately 1 billion instructions before checkpointing. Each workload has eight threads running on the *reference* input set. The thread composition of the workloads is shown in Table 3.

Before doing detailed simulation for 100 million instructions, our simulator fast forwards the simulation by 25 million instructions to warm-up the caches. Doing so averts bias in our results as streaming accesses to fill-in cold caches could disrupt DMR3D algorithms. Additionally, we scale down cache sizes to optimize simulation time, as done in [18].

For performance evaluation we used the Fair-Speedup (FS) [4] metric as it provides a better measure of the overall system improvement. FS is essentially the harmonic mean of speedups seen in individual threads. More details about the FS metric are discussed in Section S1. We also evaluate communication energy improvement using the McPAT [10] framework.

6. RESULTS

In this section, we present the results from our experiments. We show the increase in local accesses, performance and communication energy improvement of DMR3D against a baseline configuration.

6.1 DMR3D Schemes

We present results for four different schemes: baseline, GS, TODS, and Perfect. Perfect is the same as GS except that there are no restrictions on the number of page migrations per *epoch* and migrations incur no overhead. This is equivalent to an OS that can almost perfectly predict forthcoming memory accesses and bring those data closer to the processor. Our results show that this is enough to establish an upper-bound performance on our schemes. We use an epoch length of 1 million cycles and migrate 200 pages per epoch. We also compare our technique with Victim Replication (VR) [22], which attempts to improve cache performance using a data placement technique similar to DMR3D.

Since the goal of DMR3D is to increase the overall system locality, we first present results on the improvement of the number of local accesses achieved by GS and TODS. Figure 8(a) shows that both GS and TODS can increase memory access locality significantly across a range of benchmarks. Except for *mix 2* and *mix 7*, our schemes achieve at least 60% of the locality compared with the Perfect scheme. *Mix 8* has the most locality increase at 97%, while *mix 6* has the lowest at 14%. *Mix 6* has poor temporal and spatial locality as even an ideal scheme would only obtain a 20% increase. We omit VR in Fig. 8(a) as it does not affect page placement.

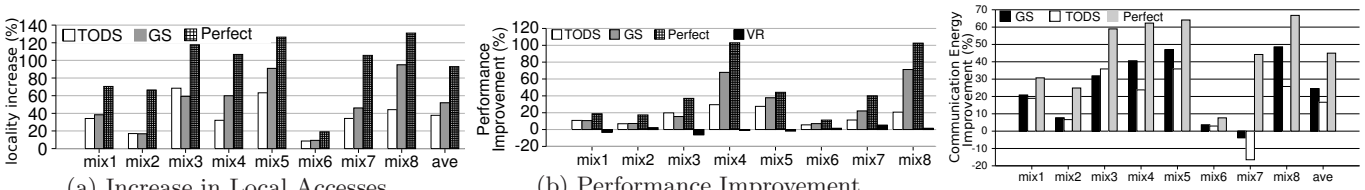
Figure 8(b) shows the collective performance improvement at the application level resulting from our proposed techniques. We estimate the performance of these workloads using Fair Speedup (see Section 5). Except on *mix 3* and *mix 1*, GS consistently outperforms TODS. GS performs better because migration slots are appropriately allocated based on global demand of the thread, while in TODS, slots can go unused if a thread does not need a non-local memory page. The best performance for GS and TODS are 71% in *mix 8* and 29% in *mix 4*, respectively. It is worth noting that *mix 8* and *mix 4* are among the four workloads with the most increase in locality. On an average, GS and TODS show 29% and 16% performance improvement in all benchmarks, respectively.

VR's performance improvement ranges from -6% to 5%. Performance degradation happens when some threads are favored at the expense of others. In some benchmarks, the memory access pattern allows VR to speed up some threads. These threads in turn can starve other slow-running threads in the benchmark. The primary difference between VR (and similar schemes) and DMR3D is that VR works at the cache latency level, and is unable to optimize speedy access of memory pages through the vertical interconnect.

6.2 Communication Energy

We show in Figure 8(c) the percentage improvement in communication energy when using the DMR3D schemes. Except for *mix 7*, all schemes show good improvements. The maximum improvement is achieved by GS on *mix 5* at 48%. Benchmark *mix 7* shows a degradation on GS and TODS. This degradation can be attributed to the program behavior changing its access patterns because the Perfect scheme shows a good improvement, increasing its total energy. On an average, we see an improvement of 25%, 17% and 45% for GS, TODS and Perfect, respectively.

Figure 9 shows the breakdown of the energy spent on memory access. The energy consumption was obtained using both the DRAMSim2 and McPat tools. The bars in each cluster represent the schemes evaluated. From left to right, they are the baseline, GS, TODS and Perfect schemes. The interconnect energy, which is used to send data across the network, accounts for more than 50% of the total energy. Note that since we are showing the breakdown in percentages, our schemes show smaller reduction compared with the baseline even though we have lowered the overall energy consumption. Benchmarks *mix 6* and *mix 7* have higher or



(a) Increase in Local Accesses. (b) Performance Improvement. (c) Comm. Energy Improvement.

Figure 8: Locality, Performance and Communication Energy Improvements (Higher is better).

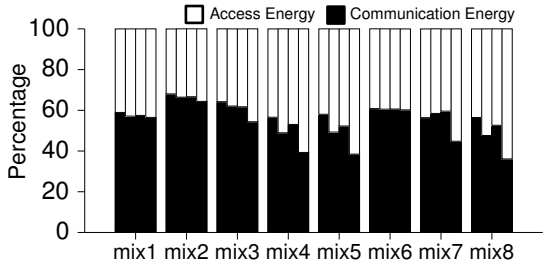


Figure 9: Breakdown of Energy Consumption.

almost equal interconnect energy compared to the baseline. This is consistent with our observation in Figure 8(c), where the unpredictable access patterns from programs may occasionally hurt the performance when using DMR3D schemes.

7. RELATED WORK

3D processor-memory integration has received a lot focus in industry and academia recently. Loi et. al showed that vertically integrating the processor and memory can achieve an impressive 65% performance speedup [13]. Liu's [11] work obtained a 90+% speedup but focused mostly on different cache-memory configurations for 3D architectures. However, they do not consider accesses to the main memory, which can be the source of a lot of traffic. In this context, Loh's work analyzed the performance improvement of aggressive configurations of 3D DRAM systems [12]. None of these works fully explore the context of a 3D NUMA multicore system, where memory latency is significantly smaller than the interconnect latency. We leveraged data from recent literature [3, 16, 19] to accurately model NUMA parameters for a 3D multicore system. The results of our work show that there is a significant performance difference between a system that is aware of data placement and one that is not.

8. CONCLUSION

In this work, we investigate the unique opportunities of exploiting the short vertical interconnect delay instead of long horizontal interconnect delays in a 3D Multicore System. We propose two schemes based on a hardware mechanism to dynamically migrate pages between distributed memory controllers: GS and TODS. GS allocates migration slots based on global demand of threads, while TODS insures fairness by equally distributing migration slots. Our techniques show 30% and 25% average performance and communication energy improvements on real world applications.

Acknowledgments

This work was supported in part by National Science Foundation grants CNS-1117425 and CAREER-1253024, and donation from the Micron Foundation.

9. REFERENCES

[1] Micron Technology Inc. Micron DDR3 SDRAM Part MT41J128M16, 2006.

[2] Abts, D. and others Achieving predictable performance through better memory controller placement in many-core CMPs. In *Proc. of ISCA* (2009), pp. 451–461.

[3] Antony, J. and others Exploring Thread and Memory Placement on NUMA Architectures: Solaris and Linux, UltraSPARC-FirePlane and Opteron-HyperTransport. In *HIPC* (2006), pp. 338–352.

[4] Chang, J., and Sohi, G. S. Cooperative cache partitioning for chip multiprocessors. In *ICS* (2007), pp. 242–252.

[5] Das, S. and others Technology, performance, and computer-aided design of three-dimensional integrated circuits. In *Proc. of ISPD* (2004), pp. 108–115.

[6] Dong, X. and others Simple but Effective Heterogeneous Main Memory with On-Chip Memory Controller Support. In *Proceedings of HPCNSA* (2010), SC '10, pp. 1–11.

[7] Gupta, S. and others Techniques for Producing 3D ICs with High-Density Interconnect. VMIC '04, pp. 1–5.

[8] Henning, J. L. SPEC CPU2006 benchmark descriptions. *SIGARCH Comput. Archit. News 34*, 4 (2006), 1–17.

[9] Holden, B. Latency Comparison Between HyperTransport and PCI-Express In Communications Systems. White paper, HyperTransport Consortium Technical Group, November 2006. Available online (11 pages).

[10] Li, S. and others McPAT: An integrated power, area, and timing modeling framework for multicore and manycore architectures. In *Proc. of MICRO* (2009), pp. 469 –480.

[11] Liu, C. and others Bridging the processor-memory performance gap with 3D IC technology. *Design Test of Computers, IEEE 22*, 6 (nov.-dec. 2005), 556 – 564.

[12] Loh, G. H. 3D-Stacked Memory Architectures for Multi-core Processors. In *Proc. of ISCA* (2008), pp. 453–464.

[13] Loi, G. L. and others A thermally-aware performance analysis of vertically integrated (3-D) processor-memory hierarchy. In *Proc. of DAC* (2006), pp. 991–996.

[14] Magnusson, P. S. and others Simics: A Full System Simulation Platform. *IEEE Computer 35*, 2 (Feb 2002), 50–58.

[15] Marathe, J., and Mueller, F. Hardware profile-guided automatic page placement for ccNUMA systems. In *PPOPP* (2006), pp. 90–99.

[16] McCormick, P. and others Empirical Memory-Access Cost Models in Multicore NUMA Architectures. In *ICPP* (September 2011).

[17] Rosenfeld, P. and others DRAMSim2: A Cycle Accurate Memory System Simulator. *Computer Architecture Letters* (jan. 2011), 16–19.

[18] Sudan, K. and others Micro-pages: increasing DRAM efficiency with locality-aware data placement. *SIGPLAN Not. 45* (March 2010), 219–230.

[19] Tezzaron. FaStackÂő Creates 3D Integrated Circuits @ONLINE, May 2011.

[20] Tikir, M. M., and Hollingsworth, J. K. Using Hardware Counters to Automatically Improve Memory Performance. In *SC* (2004), IEEE Computer Society, p. 46.

[21] Wentzlaff, D. and others On-Chip Interconnection Architecture of the Tile Processor. *Proc. of MICRO* (2007), 15–31.

[22] Zhang, M., and Asanovic, K. Victim Replication: Maximizing Capacity while Hiding Wire Delay in Tiled Chip Multiprocessors. In *Proc. of ISCA* (2005), pp. 336–345.

Supplemental Materials

S1. FAIR SPEEDUP

To measure the performance in our multiprogrammed workload, we use a metric called *Fair Speedup* (FS) [SR1]. FS is defined as the harmonic mean of per-thread Speedups. We use FS as it follows the Pareto efficiency principle where improvements on the system should increase the performance of all running threads.

$$FS(scheme) = \frac{n}{\sum \frac{IPC_i(base)}{IPC_i(scheme)}}$$

Additionally, FS is a fair metric because the harmonic mean of speedup rewards uniform improvements across all threads while at the same time penalizing slow downs. Such fairness is not exhibited by other metrics such as speedup of aggregate IPCs (Agg_{ipc}).

We give an example below to illustrate this point clearly. In this example, there are 4 threads running with IPC values indicated in Table 4. Suppose we introduce a scheme that degrades the performance of the first three threads by 10% while increasing the last one by 100%.

Setup	IPC1	IPC2	IPC3	IPC4	Agg_{ipc}	FS
Baseline	1	2	3	4	-	-
NewScheme	0.9	1.8	2.7	8	**1.17**	**1.04**

Table 4: Sample IPC values

Both performance improvements from Agg_{ipc} and FS are shown in the table (last two columns). Agg_{ipc} reports this as an impressive 17% performance increase while FS will report this at a modest 4% improvement. We believe FS gives a more accurate picture of the actual improvement because outlier values cannot heavily influence the final speedup as compared to other metrics based on arithmetic averages.

S2. PROFILING RESULTS

We present results for profiling of pages in Figure 10. We show both the average and the maximum unique pages accessed in each *epoch*, where the trend in the maximum value is 10-15× larger than the average value. Although the Profile Table needs to accommodate the maximum number of pages, on an average, it only uses a small fraction of its entire capacity. This translates to a low overhead hardware logic required to search for an entry in the table.

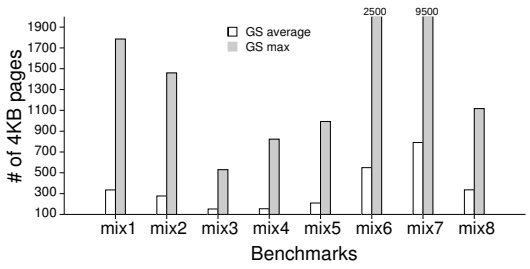


Figure 10: Page reach of benchmarks using the Global scheme. Both the average and maximum unique pages per epoch are shown.

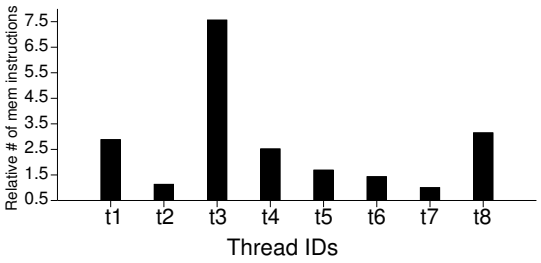


Figure 11: Breakdown of # of mem instructions executed for w1. Values relative to thread 7.

S2.1 TODS Rationale

TODS is largely motivated by the natural imbalance of the thread execution speed present in different applications. We show an example of this execution speed asymmetry in Figure 11, where thread 3 has executed 7.5× more memory instructions than thread 7. Other threads also exhibit varying number of memory instructions executed. With such diverse demands on the memory, thread fairness can suffer if less memory-demanding threads are not given migration slots by DMR3D.

S3. SIZING PT AND ART

To estimate the practical size of PT, we obtained profiles on how many unique pages are accessed by different threads in each *epoch*. We show these data in Figure 10, where on an average, most threads access less than 700+ unique pages in a 1 million cycle *epoch* size. There are also outlier workloads such as *w7*, which access at most 9500 unique pages in 1 epoch. To accommodate all workloads, we size the PT to be 10000 entries.

For the sizing of the ART, there are two factors to be considered. First, the table must be as minimum as possible to get less latency. Second, the size chosen must be enough to hold address mappings of frequently accessed data, in order to justify the cost of migration. We have experimented many parameters and found that for most benchmarks, only 200 pages need to be migrated in each new epoch to saturate the performance of DMR3D. We explore the design space of DMR3D to arrive at this choice (see Section S6).

S4. LATENCIES OF PT AND ART

Since the PT and ART are accessed for every memory request, the latency to query these tables must not add significant overhead to the overall memory access time. Using CACTI 6.0 [SR2] at 45nm, we find that associative lookups of these tables take 0.52ns and 0.30ns for PT and ART, respectively. These overheads are negligible considering that the average memory access times can reach 100ns.

S4.1 Memory Thrashing

In the context of DMR3D, thrashing can occur if data are constantly being swapped between two same memory banks such that the performance improvement is nullified by the overhead of constantly migrating data. This is usually caused by threads that share memory pages. However, modern parallel programs are designed to share data minimally in order to harness multicore performance. Since thrashing can degrade performance, we profile our benchmarks and find that sharing of pages occurs less than 0.45% of the time.

S5. PAGE REMAPPING TECHNIQUE

In a system with multiple MCs, the whole address space is divided into smaller contiguous address spaces such that each memory controller services independent requests to different memory sections simultaneously. Figure 12 shows the distribution of the memory address space across four MCs. Memory rank 0 is managed by MC0, rank 1 by MC1 and so forth. Note that we ignore other fields (Bank, Column Id etc.) in the address to simplify our discussion. The two MSBs in the address determine which MC manages its data. As such, for DMR3D to migrate and move around memory pages across memory controllers, it only needs to keep track of the previous and new rank locations of the data.

We next show the sequence of events during the migration of memory pages. The first step in migrating is to determine the source MC where the page belongs. This is taken as the two MSB in the address. The second step is to obtain the new page number by replacing the rank field with the one from the destination MC. All this information is then added to the ART of the destination MC after data migration has been done.

Once a page is listed for migration (i.e. *alien* page), the original data (i.e. *victim* page) residing at the new location must be evicted and moved to the previous address of the *alien* page, effectively swapping the two pages. The *victim* page undergoes the same process as the *alien* page but with opposite source-destination locations. Requests for the *victim* page arriving at the original MC will now be detected by ART and then routed to the remote MC. Accordingly, request for *victim* page from the remote MC will be detected by its own ART and treated as a local request.

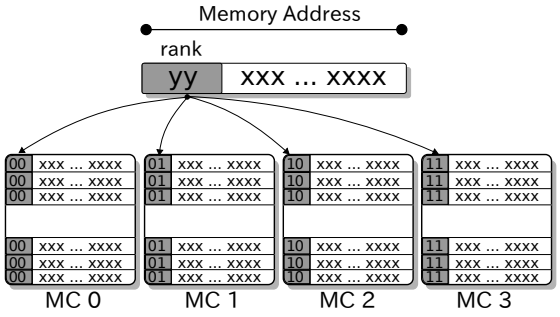


Figure 12: Memory Mapping across different MCs.

S6. DESIGN SPACE EXPLORATION

We performed two experiments: varying the number of pages migrated per epoch and changing the epoch length. To simplify our discussion, we use the term ***Migration Slot Size*** (MSS) to refer to the total number of pages migrated per epoch. The MSS values are chosen as 200, 400 and 800. These are taken from the range of values in our page reach profile experiment (Fig. 10). The *epoch* lengths are then chosen such that the migration overhead for the page is around 10% of the whole *epoch*.

S6.1 Increasing Epoch Lengths

Figures 14 and 15 show the results for the design exploration of GS and TODS. For an MSS of 200 and 400, increasing the epoch length to 2M and 4M slightly decreases the

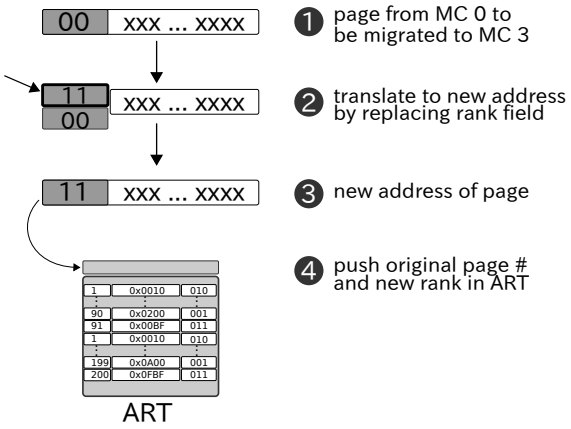


Figure 13: Sequence of events for Address Translation.

performance by 0.8% and 2.5%, respectively. This degradation trend is also exhibited on TODS but with smaller values (0.2 and 0.5%) on all MSS. The decrease in performance can be caused by two things. First, the increase in *epoch* length could increase the page reach of the program and can cause interference on the profiles generated by the PT. This can do more harm than good. Second, due to the dynamic nature of programs, the profile used to predict accesses to locations might only be useful at a fraction of the epoch length. As such, when the MSS of GS is 800, there is now an average of 4% improvement because it can handle the large page reach of threads. Hence, increasing epoch lengths does not automatically mean improved performance as it has to be matched with a much larger MSS to accommodate the corresponding large page reach of running programs.

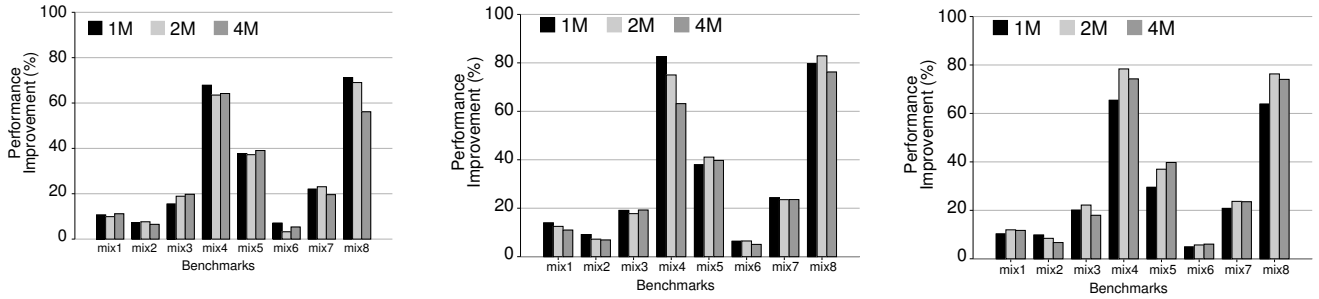
S6.2 Increasing Migration Slot Size

The same set of figures (Figures 14 and 15) show the results for increasing the MSS. We first discuss its effect on GS and then on TODS. For *epoch* lengths of 1M and 2M, using an MSS of 400 instead of 200, increases the performance by 4.25% on an average. For a 4M *epoch* length, the performance improvement drops to 3%. However, further increasing the MSS to 800 pages hurts the performance because of the migration overhead. The performance drops for 1M and 2M *epoch* lengths are 6% and 0.35%, respectively. The former has more degradation because a shorter *epoch* length translates to less chances of using migrated data before another *epoch* is reached. To overcome the migration overhead, we can use a 4M *epoch* length that increases performance by 1.2%.

While GS experiences degradation on some configurations, TODS consistently increases its performance with an increase in MSS. On an average, it improves by 3% going from an MSS of 200 to 400-pages and 400 to 800-pages. This improvement is the same for all *epoch* lengths. Although the MSS for TODS is increased, each thread will use only the migration slots if its ratio of local to non-local accesses fall below the threshold. As such, the migration overhead is reduced to threads that need to migrate and performance is realized.

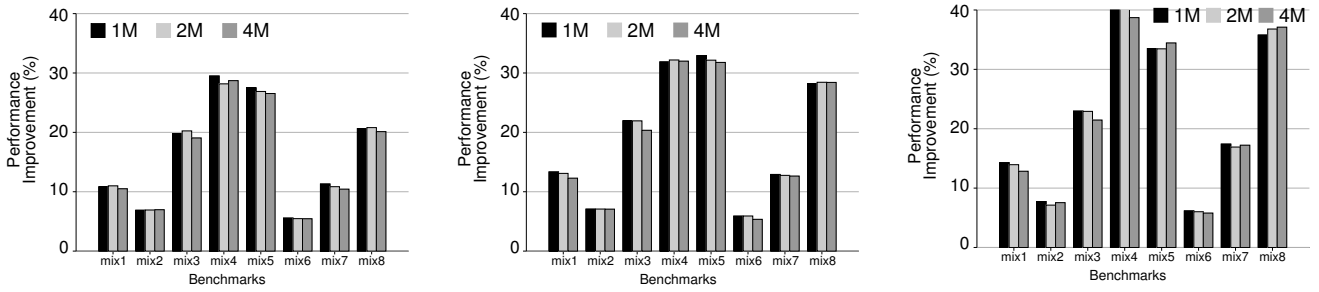
S6.3 DMR3D Design Choice

From the results of our design space exploration, we chose MSS to be 200-pages and epoch length to be 1 Million cy-



(a) Using 200-page MSS (b) Using 400-page MSS (c) Using 800-page MSS

Figure 14: Varying Epoch Sizes for Global Scheme under different Migration Slot Sizes



(a) Using 200-page MSS (b) Using 400-page MSS (c) Using 800-page MSS

Figure 15: Varying Epoch Sizes for TODS under different Migration Slot Sizes

cles. Although there is at most 5% average performance improvement using larger MSS, we cannot justify its hardware overhead on the ART size. For instance, using an MSS of 800 can improve the GS by 5% but needs an ART that is 300% larger. Furthermore, using a larger ART could lead to longer latency in table lookup. Thus, the best configuration for DMR3D is one that has fewer MSS/ART size.

S7. REFERENCES

[SR1] Chang, J., and Sohi, G. S. Cooperative cache partitioning for chip multiprocessors. In *International Conference on Supercomputing* (2007).

[SR2] Muralimanohar AND OTHERS CACTI 6.0: A tool to model large caches. University of Utah, School of Computing, Technical Report (2007).