Insights on Memory Controller Scaling in Multi-core Embedded Systems

Mario Donato Marino
Independent Researcher
Piazzale Umbria 15, Sanfatucchio (PG), 06060, Italy
Email: mdmarino@ieee.org

Kuan-Ching Li*
Dept. of Computer Science and Information Engineering
*Corresponding author
Providence University
Taichung 43301 Taiwan
Email: kuancli@gm.pu.edu.tw

Abstract:
In recent years, the growth on the number of cores as well as the frequency of cores along different processor generations has proportionally increased bandwidth needs simultaneously in both CPU and GPU systems. In order to address the communication latency between CPU and GPU memories in recent implementation of heterogeneous mobile embedded systems with hard or firm real-time requirements, sharing the same address space adds significant levels of contention. In addition, when heterogeneous cores are simultaneously present in a single system, memory parallelism is significantly restricted by a small amount of memory controllers (MCs). As a strategy to approach these significant levels of memory pressure, it is proposed in this paper evaluations of the impact of scaling MCs up to 4-8 units - limited by motherboard size for embedded purposes. Our findings show that performance is enhanced by a factor of 4x when employing only CPU cores, 4.6x when only GPU cores and finally, 2x when both CPU and GPU cores are simultaneously considered.

Keywords:
embedded; heterogeneous; memory; controller; scaling; bandwidth; performance.

Reference to this paper should be made as follows: Marino, M.D. and Li, K-C. (xxxx) Insights on memory controller scaling in multi-core embedded systems, Int. J. Embedded Systems, Vol. X, No. Y, pp.xxxxxx.

Biographical notes:
Mario Donato Marino is currently an independent researcher in Italy. He has received a best paper award on an international top conference and has co-authored 38 international articles in journals, conferences, and workshops which include computer architecture, microprocessor evaluation, systems, high-performance computing, distributed computing, parallel computing, and performance evaluation. He has served as organization chairman in one international conference. He serves/has served a number of committees in conferences, workshops, and one journal editorial board. He has received an award on teaching. He is a member of the Institute of Electrical and Electronics Engineers (IEEE) and a member Association of Computer Machinery (ACM).

Kuan-Ching Li is currently a Professor in the Department of Computer Science and Information Engineering and the Special Assistant to the University President at the Providence University, Taiwan. He is recipient of awards from Nvidia, investigator on several MOST awards (Taiwan), as also guest professorship from universities in China and Brazil. He serves/has served on the chairmanship positions of several conferences and workshops, he has organized numerous conferences related to high-performance computing and computational science & engineering, as also serving a number of journal's editorial boards and guest editorship. He has co-authored over 100 articles in peer-reviewed journals and conferences on topics that include networked computing, GPU computing, parallel software design, and performance evaluation and benchmarking. He is a Fellow of the Institution of Engineering and Technology (IET), a senior member of Institute of Electrical and Electronics Engineers (IEEE) and a member of Taiwan Association for Cloud Computing (TACC).

1 Introduction

In mobile heterogeneous embedded systems - particularly considered in this study as a set composed by CPU and GPU cores typically employed in cellphones, tablets, and notebooks - larger clock frequencies and larger number of embedded cores have been employed along each processor generation. For example, recent cellphone processors have started to achieve high frequencies such as 2.5 GHz [9]. Furthermore, cellphones and tablets are expected [4] to be fabricated with 16 cores soon, whilst there are other examples of embedded systems which present significantly larger number of cores such as the Cisco-IBM CRS-1 router [8] with 192 cores, Tilera Tile 64 [27] with just 64 cores, and the embedded NVidia Tegra4 GPU processors [12] with 72 cores.

Higher frequencies combined to larger number of cores under the intense use of bandwidth-bound applications such as the one related to video processing, gaming, and graphical environments, even with the presence of larger caches, these restrictions have been further pushing the levels of memory pressure.

In heterogeneous systems, namely systems which contain both CPU and GPU processors, the typical communication between these units is done via PCI Express bus [1]. Although the speed and bandwidth over PCI Express bus have been significantly increased, a number of techniques such as software pipelining [30], buffering, as well as overlapping of communication and computation [19] are employed to minimize PCI express overhead, yet the speed and contention of this bus is the communication bottleneck among CPU and GPU cores.

In order to eliminate this overhead, designers have designed CPU and GPU cores to share the same physical memory. In this case, data can be passed via by exchanging addresses, instead of transferring contents via PCI express bus, thus notably reducing the communication latencies, i.e., improving performance. However, this solution brings CPU and GPU cores to one single address space, which significantly leverages the pressure on the memory system.

Recently, most of heterogeneous embedded cores has incorporated out of order (OOO) techniques into their cores. For example, the incorporation of a reorder-buffer (ROB) in ARM A9 [6] architecture improves the throughput in regards of the number of instructions, which is likely to demand higher data throughput from the memory as high bandwidth-bound applications - such as graphic-oriented traffic programs - are executed. Furthermore, Intel Atom [15] processor already incorporated a ROB and the recent Intel Haswell processor [16] - a traditional OOO-microprocessor that employes a ROB - designed with low-energy techniques to be also employed on embedded systems. This technique is naturally going to increase memory pressure, when compared to traditional in-order cores present in current embedded mobile systems. Therefore, in systems with combination of both types of cores to form an heterogeneous multicore system the number of simultaneous memory requsts is going to significantly increase, therefore pushing further the levels of contention, represented by larger transaction queues and/or larger duration of the transactions [20] at the MCs.

The natural solution to provide more bandwidth as the number of heterogeneous cores scales is to increase memory parallelism. One straightforward solution to augment memory parallelism is by scaling MCs. However, as reported in [21], the scalability of MCs in DDR-systems is restricted by the scalability of I/O pins. As a consequence, typical DDR-systems employed as memory solutions in mobile embedded systems, present a low amount of MCs. For example, typical cellphones or tablets present 1-2 MCs whilst about 8 MCs in GPUs or router processors.

In order to leverage the area of heterogeneous embedded core systems by evaluating the impact of improving memory parallelism in these embedded systems, we investigate in this paper the effects of scaling the number of MCs, respecting the limits imposed by the I/O pin scaling in current DDR systems. By considering that cache addresses are interleaved among ranks, and each rank is independently connected to a different MC to benefit the extraction of its maximum bandwidth, we create a multi-core model and assess it in terms of MC scalability, evaluating the bandwidth and performance benefits, by using detailed and accurate simulation tools combined to several intense and medium intense memory bandwidth-bound benchmarks. As a result of this study, we envision the following contributions:

- Current CPU and GPU cores share the same physical address space via L3 sharing [16]. We assume that future memory systems are likely to allow the isolation of individual CPU or GPU address spaces and investigate the performance benefits of scaling the number of MCs within core:MC ratios limits of current heterogeneous embedded systems. To the best of our knowledge, through extensive analysis of related researches, this is the first work which MC scalability investigation and analysis are performed in embedded systems.
- The investigation is performed aiming to determine the performance benefits of MC scaling along each individual (i) CPU address space, (ii) GPU address space, and when (iii) combining both.
- Given that in (iii) the likely memory contention is further larger than when either CPU or GPU cores are individually using the memory address space, we propose a methodology that combines the performance of (i) and (ii) to obtain (iii), assuming the same number of MCs and benchmarks with the same number of memory requests.

Although memory power/energy are of fundamental importance in mobile embedded systems, unfortunately we leave the evaluation of these aspects for a further study. Section 2 describes the background and motivation for researching memory systems in order to provide larger bandwidth in embedded systems. Section 3 describes the benefits of scaling MCs and analysis when having them allocated to CPU/GPU with separate or combined address spaces. In Section 4 we discuss the experimental methodology as also to present respective results, and finally Section 5 describes the related work and Section 6 presents the conclusions and items to be developed as future work.

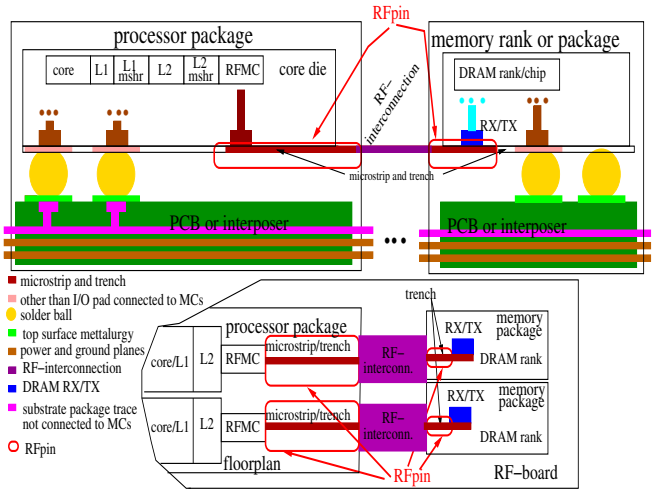

Figure 1: traditional memory path;

2 Background and Motivation

We start in this section by describing the role of MCs in current memory systems. Next, an overview of the I/O pin problem restrictions on MC scaling is discussed.

2.1 The role of MCs

In this subsection we describe the role of MCs in current memory systems. Before we describe the role of MCs, we assume that these systems are based on DDR-memories in terms of timings, protocols, control-data signal separations, and organization - ranks, banks, rows, and columns. Furthermore, ranks are assumed to be manufactured as single module chip packages in order to minimize its total occupied area so that, when scaled together with the on-chip MCs, they do not turn into a space restriction. We observe that the assumption regarding rank manufacturing has its viability relying on similar fabrication of ranks in this form, such as in embedded systems or for servers with HMC [14] memory system.

The role of the MC in typical memory systems is illustrated in Figure 1. At the flip-chip package interface, when a cache request is received at the MC, signals traverse the following path from the MC to the rank: MC, package trace, package via, repeaters, the structures which form the pins - such as pad, solder balls, and PCB-pad - and finally the signal reaches the PCB trace, and PCB via, followed by the same sequence in the opposite order when these signals reach the rank. The same path in the reverse order happens when the rank response happens from the rank to the MC. Similarly, in embedded systems, an interposer is employed instead of a PCB, and the path of these signals is changed to: MC, package trace, package via, repeaters, pad, solder balls, interposer trace, and the reverse order when the response of from the rank is performed.

2.2 The I/O pin problem

The I/O pin problem is characterized by a set of physical restrictions which are likely to happen as the number of pins increases, larger pin-densities are employed, and larger clocks along the processor-to-memory channel are scaled. These pin restrictions involve electro-migration and crosstalk effects among pins, as well as implementing a reliable connection between the motherboard PCB or alternatively the interposer, and the processor pads [20]. In addition, as pins are scaled, area costs are like wisely to increase.

We illustrate the effect of these restrictions (a) in current embedded and typical microprocessors in terms of cores versus MC counts and (b) the effects of pin-counts on bandwidth and MC counts.

Aiming to illustrate (a), we show the effects of the increase of the number of cores versus MC counts in typical microprocessors which have similar features to the most advanced employed in embedded systems. In Figure 2a, for purposes of reference, we show the red-line where core:MC-count ratio has magnitude 1:1. All the examples in this figure are placed on the right of the 1:1 magnitude, which means that most of the systems - embedded and traditional - have more cores than MCs, and which reflects the current imbalance between MC counts and cores. Another example is the 192-core Cisco-IBM CRS-1 router, which has 16 MCs [5][8][27].

In Figure 2a, the total bandwidth magnitudes achieved are still at lower-sides when compared to core-growth, even counting the largest rank bandwidths over each MC since, since MC counts are found at lower ranges. Figure 2b illustrates (b) how significant are pin-count magnitudes in current systems. It also shows how bandwidth is restricted in terms of number of MCs and pins in Intel systems according to Polka [26]. Furthermore, it shows larger MC- and pin-counts of GPUs such as NVIDIA GPU GT200 (8 MCs, 2500 pins) as well as embedded Tilera Tile 64 (4 MCs, 1500 pins).

As a motivation, Figures (a) and (b) demonstrate the need of focusing on larger MC counts to approach the bandwidth demands due to the core growth. Given these motivations, we proceed to analyze the impact of MC scalability towards bandwidth and performance.

3 MC scalability

Equation 1 shows how rank bandwidth and the number of I/O pins are combined to obtain the bandwidth per pin:

$$bw\_pin = bandwidth\_rank/number\_of\_IO\_pins, \quad (1)$$

To understand the effects of MC scaling towards improving bandwidth, we observe in this equation that as the number of pins is reduced, bandwidth per pin is increased.

Before we define bandwidth as a function of rank frequency and width, as previously mentioned it is important to note that we are assuming address interleaving among different ranks. Assuming a typical configuration where the MC clocked at half of the processor frequency, rank frequency is the dominant factor in terms of performance. To understand the effects of scaling MC counts towards bandwidth, we define the peak bandwidth supplied by one rank as a function of its frequency and width as follows:

$$peak\ bandwidth = rank\ frequency * width \quad (2)$$

According to the memory path previously described, data stored in the rank is forwarded to or comes from the MC, which itself forwards to the cache(s) attached to it (them). Assuming we have multiple MCs, each MC independently connected to one rank, we can model the total peak bandwidth as in equation 3. Since MCs are independently controlling the ranks attached, the peak bandwidth is proportional to the number of MCs. For instance, for a typical rank data frequency or data rate of 1333MT/s in a system with only 1

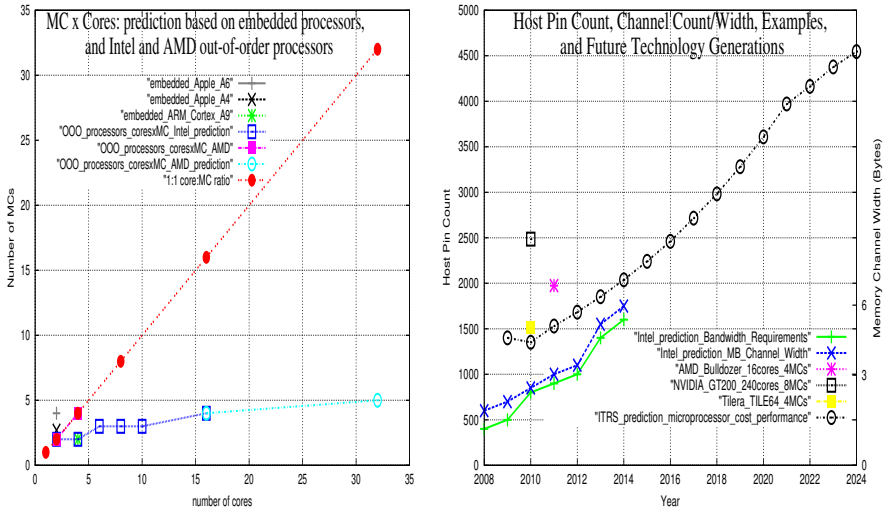

Figure 2: left to right (a) examples cores versus MC counts ; (b) ITRS pin limits, MC versus pin-count: Intel predictions and other examples - repeated from [20]

MC can have a peak bandwidth of 10.664 GB/s, whilst with 2 MCs, the peak bandwidth achieves the double, i.e., 21.328 GB/s. and with 4 MCs, peak bandwidth is improved fourfold, and finally about 8x for 8 MCs.

$$peak\ bandwidth = rank\ frequency * width * MCcounts \quad (3)$$

These peak bandwidth levels when performing read or write operations are reduced due to contention in the crossbar or buses that are employed along the interconnection from the MC to the rank, ranks, and caches.

3.1 Dedicating MCs to CPU-, GPU-based, and combined CPU-GPU address spaces

In order to scale MCs and allocate them to each individual memory space - formed by only CPUs, GPUs, and combined CPUs and GPUs, we propose the following mechanisms:

- The number of MCs to be scaled is dedicated to each of these memory address spaces (CPUs/GPUs/combined CPU/GPU) can be selected via an operating system (OS) interface combined to a crossbar (or similar dedicated hardware), or by disabling L3 sharing among the different sets of cores.
- Upon the creation of the isolated address spaces and the configuration of the number of MCs allocated to each of them - both further discussed - in case independent address spaces are needed, each proceeds its memory access as an independent one, with different degrees of memory parallelism given by the number of MCs available to that space. We leave the discussion of the identification, allocation, and reconfiguration of these address spaces as a future effort.

In the text that follows next, we define each of the address spaces by exemplifying their functionality. Figure 3 illustrates the case with CPU, GPU, and combined heterogeneous (CPU/GPU) region. Each region, CPU, GPU, or heterogeneous has a certain number of MCs allocated to it. To exemplify the functionality of the mechanisms proposed we illustrate with the following examples: (i) previous configuration: address space shared by both CPUs and GPUs and an interleaved memory addressing along 8 MCs (which we assume as the upper limit within the restriction of current pin scaling technologies); assuming that after a reconfiguration - further described, addresses generated by the CPUs are interleaved only among 6 MCs, which form a CPU address space, while, on the second address space (isolated as well) there are only GPUs, which are able to utilize the remaining 2 MCs.

Another example (ii), with the previous configuration as the last one (i), after a first reconfiguration of example (i); assuming as a motivation that CPUs are executing cache-intensive programs (and not bandwidth-bound), we propose

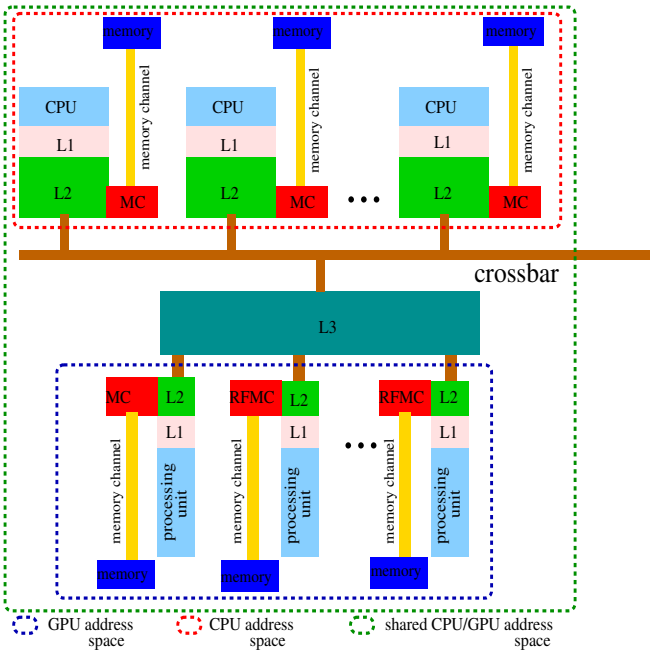

Figure 3: Example on the creation of different regions: one CPU address space, one GPU address space, or combined (heterogeneous shared CPU/GPU) address space

CPU Core	16 cores, 4.0 GHz, OOO-Core, 4-wide issue, tournament branch predictor
GPU Core	256 cores, based on Fermi architecture [2], 0.325 GHz,
technology	22 nm
L1 cache	32kB dcache + 32 kB icache; associativity = 2 MSHR = 8, latency = 0.25 ns
L2 CPU cache	1MB/per core ; associativity = 8 MSHR = 16; latency = 2.0 ns
L2 GPU cache	32kB/MC MSHR = 16; latency = 2.0 ns
crossbar (CPU region)	latency = 1 cycle
GPU interconnection	0.325 GHz,
MCs trans. queue	1 to 8 MCs; 1 MC/core, 2.0GHz for CPUs, on-chip, close page mode
trans. queue	2.0GHz for GPUs, on-chip, close page mode buffer size = 32/MC
Memory rank	DDR3 1333MT/s, 1 rank/RFMC, 1GB, 8 banks, 16384 rows, 1024 columns, 64 bits, Micron MT41K128M8 [23] tras=26.7cycles, tcas=trcd=8cycles

Table 1 Modeled architecture parameters

the creation of an address space with just 2 MCs dedicated to the CPUs, while the remaining 6 MCs, can be allocated to the GPUs, which form a second address space.

As a final example of (iii), we consider the previous configuration (ii): after performing (ii) we propose to form a single address space, where all the 8 MCs available and can be used towards heterogeneous programs where CPUs and GPUs simultaneously have their respective slice of computation, and where variables and dataflow of each computation unit (CPU/GPU) interact (exchange via passing addresses among them, where OpenCL and CUDA paradigms are employed as programming paradigms).

The reconfiguration operations assumed are responsible for the reconfiguration itself of the created address spaces and the allocation of the MCs to the processing elements according to the selected goal. After these steps, the new regions, address space, and MC allocation are performed. Due to the design and evaluation complexities involved, the investigation of the creation and reconfiguration of these address spaces and its properties are left as future researches.

4 Experimental Section

In this section we present the methodology employed first, followed by the bandwidth/speedup results obtained in our experimental infrastructure.

4.1 Methodology

To model a CPU-based address space with a set of allocated MCs we employ a combined integration between M5 [24] and DRAMsim [11] simulators. In this integration, memory transactions are generated by M5 and sent to DRAMsim, which configured with multiple MCs, yet responding to M5 with the result of the memory transactions. We further describe how timings involved in the interconnection (crossbar) are incorporated in these combined simulators.

Benchmark	Input Size	read:write, MPKI
Copy, Add, Scale, Triad (STREAM)	4Mdoubles per core, 2 iterations	2.54:1 , 54.3
pChase	64MB/thread, 3 iterations, random	158:1 , 116.7
Hotspot,	6000 x 6000, 3 iter.	2.5:1 , 12.5
Pathfinder	65536, 2 iter.	- , -
Backprop	2 iter.	- , -
Srad, (Rodinia)	2 iter.	- , -

Table 2 benchmarks and input sizes

To model a GPU-based address space with an allocated set of MCs, we employ GPGPUsim [3] simulator, which already contains a module that implement multiple DDR-based MC-system. Memory transactions are generated by the multiple GPU caches and and treated in the memory module of GPGPUsim. Similarly to the CPU-based case, we further describe how interconnection timings are incorporated in this simulator.

In order to model a heterogeneous address space, also taking into consideration the same simulators previously mentioned, the integration of the GPGPUsim into GemM5 CPU simulator is already implemented as indicated in [13]. Nevertheless, this combined infrastructure further increases simulation complexity, and therefore, significantly increases simulation times. To address this restriction, we propose a simpler methodology: (i) we first determine the maximum bandwidth of each CPU and GPU address spaces. (ii) We obtain the bandwidth of the heterogeneous address space by assuming that the bandwidth of the heterogeneous address space as a fraction of the bandwidth of the CPU or GPU address spaces, given that a larger number of requests is present in the heterogeneous ones. This fraction corresponds to the memory access component on the CPU or GPU over the total memory accesses. In this study, given the difference in terms of number of cores and operating frequencies, we make the simplest assumption of having the number of GPU memory requests of the same order of CPU memory requests. Therefore, the bandwidth obtained in the heterogeneous address space is halved.

To determine the behavior of programs where tasks are scheduled between CPU and GPUs, it is necessary to perform an individual program analysis which depends on the program parallelization in order to identify the composition of these memory accesses in each CPU or GPU spaces. We leave this investigative analysis as a future effort.

For the CPU regions, we employ different MC counts in the 16:1 to 16:8 range (i.e., up to 8 MCs for 16 cores), and the baseline with 2 MCs, to reflect typical configurations found in tablets and cellphones [9][16]. For the GPU regions, we utilize similar methodology, varying MCs in the 1-8 range and apply it in Nvidia Fermi architecture [2]. In order to evaluate MC scalability, memory timing parameters are based on 1GB DDR3 rank, based on Micron model MT41K128M8 [23].

Regarding validating MC scalability to other rank parameters such as other rank clock frequencies, it is important to mention that this type of parallelism was

previously explored [20][21] not only in off-chip memories but also for on on-package memory configurations which employ other different ranges of frequencies, which further demonstrate the validity and the coverage of the proposed technique, when high-bandwidth applications executed on multi-cores demand high memory bandwidth.

As to guarantee pressure on the memory system in the CPU address space, we have utilized an OOO embedded core. The CPU processor modeled follows a clustered architecture, where we have one core per L2 slice, i.e., private L2 slices in order to avoid cache sharing effects. The CPU ISA employed is based on Alpha processor, configured as a 4-way issue OOO core similar to Intel Haswell [16]. Furthermore, we presumed a banked and scalable L2 MSHR structure [29] and assumed 1MB/core as an L2 (CPU) cache slice size to reflect current OOO embedded cores. Similarly, GPU processor utilized in the GPU-based regions follow Nvidia Fermi architecture [2] which itself, given its larger number of cores, yet typical employed applications, demands significant memory bandwidth.

CPU and GPU L2 slices are interconnected through an 1-cycle crossbar (optimistic assumption to elucidate the noticeability of the memory transfers). We obtained cache latencies from Cacti [7] with energy optimizations and adopted MSHR counts for each L2 slice of as in typical multi-cores [16].

PCB delays are not included in the baseline modeling since we found a broad variety of magnitudes; due to that, the baseline measurements, such as bandwidth / speedups, are closer to the ideal case, i.e., the likely bandwidth results are better than ones achieved in this experimentation.

STREAM suite applications [22] are specifically designed to evaluate bandwidth, whilst pChase is designed to evaluate both bandwidth and latency [25], and Hotspot, Pathfinder, Backpropagation, and Srad are some of the bandwidth-bound applications in heterogeneous Rodinia suite [28]. Using Loh's criteria [18] to select memory bandwidth-bound benchmarks, however with focus on the ones with medium or high number of misses per kiloinstructions (MPKI) to stress the memory system. the following benchmarks have been selected for CPU regions: STREAM [22] suite, which we decompose in its four sub-benchmarks (Copy, Add, Scale, and Triad); pChase [25] benchmark with pointer chase sequences randomly accessed. to evaluate combined heterogeneous CPU and GPU regions spaces, Hotspot and Pathfinder from Rodinia suite [28] were selected, and finally, Backpropagation and Srad applications from Rodinia suite to evaluate GPU regions.

Table 2 summarizes the benchmarks experimented, input sizes, read-to-write rate, and L2 MPKI obtained in the experiments. In all benchmarks, the parallel regions of interest were executed until completion. All the input sizes are larger than the total rank memory size, which guarantees that all the memory spaces are stressed. The average results were calculated based on harmonic average.

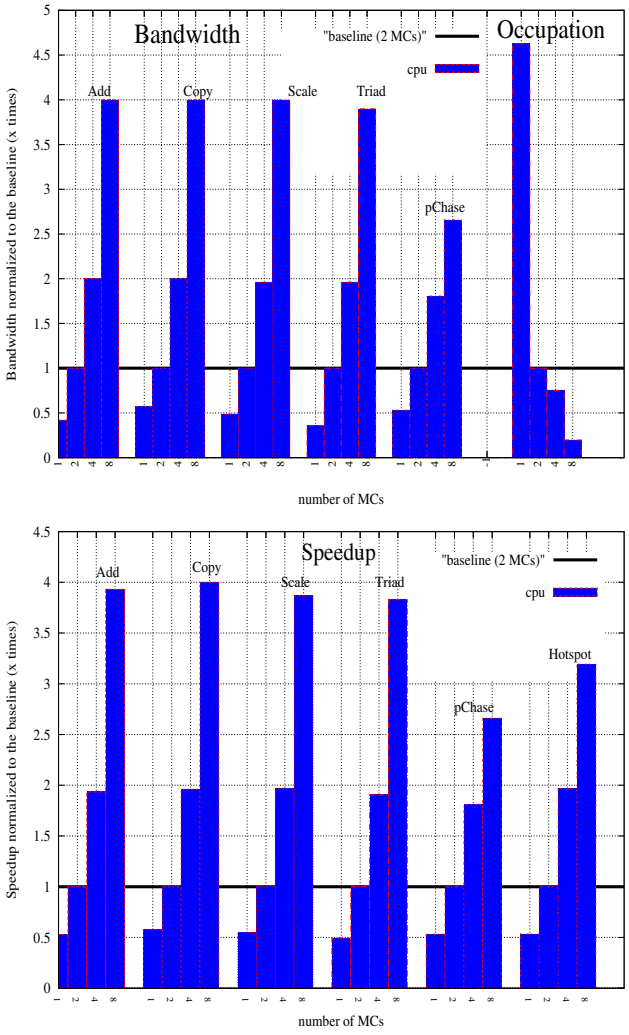

Figure 4: top to bottom: (a) bandwidth versus number of MCs for CPU regions; (b) speedup versus number of MCs for CPU regions.

4.2 *Results: Bandwidth and Speedups*

Figure 4a shows the bandwidth results obtained for the CPU region. In all STREAM benchmarks and pChase, which were designed to measure bandwidth magnitudes, a remarkable bandwidth improvement factor of 4x more bandwidth than the baseline for the CPU space was obtained. As a result, significant larger number of memory transactions are simultaneously processed. Therefore, the memory parallelism obtained through MC-scaling also reduces the size of the transaction queues and time transactions occupied in the queue.

Alternatively, since bandwidth and latency are related, bandwidth increase is followed by a latency reduction. To understand the benefits of the lower latency obtained, we have measured the transaction queue average occupancy and duration in the CPU spaces. Compared to the baseline, transaction queue occupancy is respectively reduced about 90%, as shown on the right side of Figure 4a.

The speedups obtained across the benchmarks for the CPU regions are illustrated on Figure 4b. In this figure, for all benchmarks, we observe that speedups increase in the same proportion as a result of the larger scalability. For STREAM, speedups of CPU spaces are up to 4x faster than the baseline, therefore noticeably faster. Similar scaling trends are obtained for pChase as well. Furthermore, it is important to highlight that significant results obtained in

pChase in regarding to speedups, bandwidth, and latency given that they are obtained with random accesses, they also demonstrate the generality of the solution when high-bandwidth is required by the application, as demonstrated from the diversity of benchmarks employed in this evaluation either in CPU or GPU spaces.

As mentioned before the method to obtain the bandwidth of the heterogeneous region, given that we have arbitrarily selected the CPU bandwidth to be used in this method, so bandwidth results for the GPU spaces are not shown though concentrating on their speedups. The correspondent speedups are illustrated in Figure 5a. For all benchmarks programs we observe similar behavior to the CPU spaces regarding MC scaling, that is, speedups proportionally increases as the number of MCs increases. For Hotspot, Pathfinder, Backpropagation, and Srad, similar improvement trends are obtained. The largest bandwidth/speedup improvements occur for Pathfinder, achieving about 4.5x faster than the baseline, due to its access pattern and MPKI magnitudes (refer to Table 2). Moreover, these significant speedup results show that memory traffic is contained in the CPU spaces.

The same bandwidth trends happen regarding the heterogeneous CPU/GPU region, i.e., MCs proportionally scales with bandwidth as shown in Figure 5b. With the assumption of equivalent number of memory requests on the CPU and GPU individual spaces, we have obtained up to 2x bandwidth, assuming the baseline described as follows. Instead of having the previous baseline reference (2 MCs), we have preferred to present the above results having the CPU region as a general baseline in order to demonstrate their smaller bandwidth comparatively to the CPU regions obtained in Figure 4a.

As a general conclusion, we observe that for individual CPU or GPU spaces, as well as for the heterogeneous one, MC scaling benefits performance by improving memory parallelism. Combined CPU/GPU sets need more bandwidth to achieve the same levels of speedup of independent ones.

Finally, although it is not our aim to compare parallelization techniques and parallel architectures, it is interesting to observe that, using the same inputs for CPU and GPU spaces, the behavior of the performance of the benchmarks experimented in these two platforms is different. Hotspot presents a better performance on the CPU regions, whilst Pathfinder on the GPU ones. In Rodinia, these GPU programs were parallelized using CUDA, while on the CPU versions, the same Rodinia applications utilized OpenMP. This apparent inconsistency happens due to the fact that in order to have a fair performance comparison between programs executed on different architectures, as indicated in [17], we should have both programs parallelized in both platforms using similar techniques to achieve the best possible performance on that architecture. For example, if parallelized in CUDA, the program indirectly control GPU caches, and similar techniques should be employed for the CPU version aiming to have the fairest comparison; therefore it is not possible to have a fair comparison under these circumstances without developing all the steps as target.

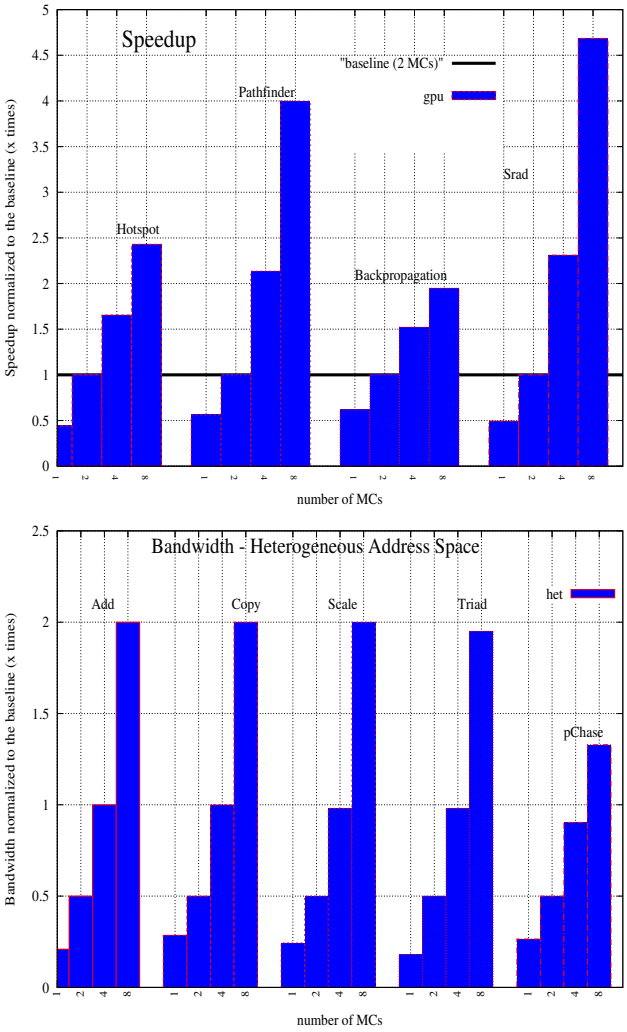

Figure 5: top to bottom: (a) Speedup versus number of MCs for GPU regions; (b) bandwidth results for heterogeneous region (CPUs and GPU shared region); the baseline for the heterogeneous region is assumed as the same of Figure 4a

5 Related work

10 TB/s-bandwidth Corona [10] optical memory system (160 GB/s/MC) was designed aiming low energy levels (7.8 nJ/bit) per memory channel access, and most importantly, with only 2 optical I/O-pins per optical memory. In this study, we employ traditional digital MC organization and electrical and therefore, MC scaling limits are significantly limited.

HMC [14] is a recent memory solution designed to target 3Dstacking and for off-chip memory systems. In the case of off-chip memories, either HMC or this study use an external memory package as memory ranks. HMC organizes its memory package by employing sets of banks of the memory dies, and processor/memory communication is done via serial/deserial, with 10-Gbit/s-I/O-links. To contrast with HMC, in this study, we follow a typical DDR memory organization, however, we share with this technology the use of external memory packages.

The levels of MC scaling employed in this study are significantly lower than the employed in RF-memory systems [21], given the latter approaches MC scalability in a

similar way to optics (RFpins and optical-pins to favor MC scalability).

6 Conclusions

In this investigation, we proposed a strategy to increase the memory bandwidth of heterogeneous embedded mobile systems, consisting of having individual address spaces for CPUs and GPUs, or for heterogeneous shared CPU/GPU spaces, whereas on each one, we evaluate the effects of scaling MCs on the memory bandwidth. The result of this investigation indicates significant bandwidth and speedup improvements in each type of address space listed.

As a further effort, we plan to implement and evaluate the larger MC scalabilities via optical and RF techniques as well as investigate the benefits of these techniques in terms of energy benefits. Furthermore, we also aim to investigate the benefits of programs where tasks are divided among CPUs and GPUs using CUDA/OpenCL programming APIs. We also intend to proper identify, propose allocation and reconfiguration mechanisms of these address spaces as a future effort.

7 Acknowledgements

Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the National Science Council or Nvidia. This research is based upon work partially supported by Nvidia, National Science Council (NSC), Taiwan, and Providence University, Taiwan. The authors appreciate and thank Maria A. Guitti and anonymous reviewers for their reviews and valuable suggestions.

References

[1] PCI Express* Architecture. Accessed date: 04/10/2013 ; http://www.intel.com/content/www/us/en/io/pci-express/pci-express-architecture-devnet-resources.html.

[2] . Fermi Architecture White Paper - Nvidia. In *White Paper*, 2009.

[3] A. Bakhoda, G. L. Yuan, W. W. L. Fung, H. Wong, T. M. Aamodt. Analyzing CUDA workloads using a detailed GPU simulator. In *International Symposium on Performance Analysis of Systems and Software*, pages 163–174, Boston, Massachusetts, USA, 2009. IEEE.

[4] AMD details first ARM-based server chip: up to 16 helpings of Cortex-A57 clocked at 2GHz, 2013. accessed date: 08/15/2013 - http://www.engadget.com/2013/06/18/amd-seattle-arm-server-chip/.

[5] AMD Reveals Details About Bulldozer Microprocessors, 2011. accessed date: 11/10/2012 - http://www.xbitlabs.com/news/cpu/display/20100824154814_AMD_Unveils_Details_About_Bulldozer_Microprocessors.html.

[6] Comparison of CPU architectures. Accessed date: 06/05/2013 - http://en.wikipedia.org/wiki/Comparison_of_CPU_architectures.

[7] CACTI 5.1. Accessed Date: 10/20/2012; http://www.hpl.hp.com/techreports/2008/HPL200820.html.

[8] The Push of Network Processing to the Top of the Pyramid. Accessed date: 07/21/2012 ; http://www.cesr.ncsu.edu/ancs/slides/eathertonKeynote.pdf.

[9] Cortex-A15 Processor, 2012. accessed date: 07/02/2013 - http://en.wikipedia.org/wiki/ARM_CortexA15_MPCore.

[10] Dana Vantrease et al. Corona: System Implications of Emerging Nanophotonic Technology. In *ISCA '08*, pages 153–164, Washington, DC, USA, 2008. IEEE Computer Society.

[11] David Wang et al. DRAMsim: a memory system simulator. *SIGARCH Comput. Archit. News*, 33(4):100–107, 2005.

[12] Nvidias Tegra 4 demystified: 28nm, 72-core GPU, integrated LTE, and questionable power consumption, 2013. accessed date: 05/25/2013 - http://www.extremetech.com/computing/ 144942-nvidias-tegra-4-demystified-28nm-72-core-gpu-integrated-lte-and-questionable-power-consumption.

[13] Hao Wang, Vijay Sathish, Ripudaman Singh, Michael Schulte, Nam Sung Kim. Workload and Power Budget Partitioning for Single-Chip Heterogeneous Processors. In *Int. Conf. on Parallel Architecture and Compilation Techniques (PACT)*, Minneapolis, USA, 2012. IEEE/ACM.

[14] Hybrid Memory Cube Specification 1.0. Accessed date: 07/08/2013 ; http://www.hybridmemorycube.org/.

[15] Intel Atom Processor, 2013. accessed date: 04/24/2013 - http://www.intel.com/content/www/us/en/processors/atom/atom-processor.html.

[16] Haswell (microarchitecture), 2013. accessed date: 08/03/2013 - http://en.wikipedia.org/wiki/Haswell_(microarchitecture).

[17] Lee, Victor et al. Debunking the 100X GPU vs. CPU myth: an evaluation of throughput computing on CPU and GPU. In *ISCA*, pages 451–460. ACM, 2010.

[18] Loh, Gabriel H. 3D-Stacked Memory Architectures for Multi-core Processors. In *ISCA'08*, pages 453–464, Washington, DC, USA, 2008. IEEE Computer Society.

[19] Lustig, D. and Martonosi, M. Reducing GPU Offload Latency via Fine-Grained CPU-GPU Synchronization. In *19th IEEE International Symposium on High Performance Computer Architecture (HPCA)*, 2013.

[20] Marino, M. D. RFiop: RF-Memory Path To Address On-package I/O Pad And Memory Controller Scalability. In *ICCD, 2012, Montreal, Quebec, Canada*. IEEE, 2012.

[21] Marino, M. D. RFiof: An RF approach to the I/O-pin and Memory Controller Scalability for Off-chip Memories. In *CF, May 14-16 , Ischia, Italy*. ACM, 2013.

[22] McCalpin, John D. Memory Bandwidth and Machine Balance in Current High Performance Computers. *IEEE TCCA Newsletter*, pages 19–25, Dec. 1995.

[23] Micron manufactures DRAM components and modules and NAND Flash. Accessed date: 02/28/2013 ; http://www.micron.com/.

[24] Nathan L. Binkert et al. The M5 Simulator: Modeling Networked Systems. *IEEE Micro*, 26(4):52–60, 2006.

[25] The pChase Memory Benchmark Page. Accessed date: 05/22/2011 ; http://pchase.org/.

[26] Polka L. A. et al. Package Technology to Address the Memory Bandwidth Challenge for Tera-scale Computing. *Intel Technology Journal*, 11(3):197–206, 2007.

[27] Shane Bell et al. TILE64TM Processor: A 64-Core SoC with Mesh Interconnect. pages 88–90. IEEE, 2008.

[28] Shuai Che et al . Rodinia: A benchmark suite for heterogeneous computing. In *IISWC*, pages 44–54. IEEE, 2009.

[29] Tuck, James et al. Scalable Cache Miss Handling for High Memory-Level Parallelism. In *MICRO'06*, pages 409–422, Washington, DC, USA, 2006. IEEE Computer Society.

[30] Udupa, Abhishek and Govindarajan, R. and Thazhuthaveetil, Matthew J. Software pipelined execution of stream programs on gpus. In *Proceedings of the 7th annual IEEE/ACM International Symposium on Code Generation and Optimization*, CGO '09, pages 200–209, Washington, DC, USA, 2009. IEEE Computer Society.