WCET Analysis of the Shared Data Cache in Integrated CPU-GPU Architectures

Yijie Huangfu
Virginia Commonwealth University
huangfuy2@vcu.edu

Wei Zhang
Virginia Commonwealth University
wzhang4@vcu.edu

***Abstract*—By taking the advantages of both CPU and GPU as well as the shared DRAM and cache, the integrated CPU-GPU architecture has the potential to boost the performance for a variety of applications, including real-time applications as well. However, before being applied to the hard real-time and safety-critical applications, the time-predictability of the integrated CPU-GPU architecture needs to be studied and improved. In this work, we study the shared data Last Level Cache (LLC) in the integrated CPU-GPU architecture and propose to use an access interval based method to improve the time-predictability of the LLC. The results show that the proposed technique can effectively improve the accuracy of the miss rate estimation in the LLC. We also find that the improved LLC miss rate estimations can be used to further improve the WCET estimations of GPU kernels running on such an architecture.**

I. Introduction

While GPUs are used as powerful accelerators, CPUs are also increasingly considered and used as coprocessors, rather than just the host cores that simply launch tasks to GPUs and are not involved in computing. Such integrated CPU-GPU architectures exploit the unique strengths of both types of Processing Units (PUs) as well as the shared resources to further improve the performance, compared to a GPU- or CPU-only system. For instance, seven out of the top ten Green500 supercomputers use both CPUs and GPUs[1], i.e., Heterogeneous Computing Systems.

In a heterogeneous architecture, CPUs and GPUs can have separate memory spaces and can be connected together through a Peripheral Component Interconnect Express (PCIe) bus, which is referred as a *discrete* system. GPUs and CPUs transfer data back and forth through the PCIe bus in such a system, which requires programmers to manage the data needed by both CPUs and GPUs and can introduce performance overheads. As a result, the *integrated* CPU-GPU architecture is proposed and implemented to allow the CPUs and GPUs to share the same memory space and avoid such data transfer, e.g., AMD's Accelerated Processing Units (APUs)[2].

In embedded applications, heterogeneous architectures have become increasingly popular as well. For instance, the *big.LITTLE* technology [3] combines high-performance cores and energy-efficient cores to achieve power-optimization while delivering peak-performance capability. Also, by the integration of GPU and CPU architectures, the Tegra [4] processors bring the general-purpose GPU computing power to the embedded systems.

In real-time systems, it is very promising to exploit the computing power of the integrated CPU-GPU architecture as well. However, the issues of the time-predictability in such systems need to be addressed first. One of the problems is the estimation of the behavior of the shared Last Level Cache (LLC) in such an integrated architecture, since it is shared by both the CPU and GPU and can affect the WCETs of both. Therefore, we propose to first explore the WCET analysis of the shared data LLC in the integrated architecture, the analysis results of which are then used in estimating the WCET of the GPU kernels.

II. Related Work

For WCET analysis of the multicore architecture, page coloring and locking techniques are studied and used to reduce or remove the conflicts between different cores in the LLC[5][6][7], so that the time-predictability can be improved. Hardware supports are proposed in [8] to guarantee an upper bound delay for hard real-time tasks in multicore systems, while the Time Division Multiple Access shared bus access scheme is proposed in [9] to enable the static shared bus scheduling and shared cache conflict analysis.

Research efforts have been made on partitioning and/or scheduling tasks or specific algorithms on heterogeneous architectures, based on the relative performance of different processing units and/or the characteristics of different subtasks [10][11][12][13]. Some studies focus on the compiler-level methods to automatically generate the programs for heterogeneous systems[14][15], while others propose programming frameworks to utilize the resources[16][17]. Comparisons between the discrete and integrated CPU-GPU architectures show that the integrated architecture can help to reduce the performance and/or energy overheads [18][19][20]. However, few studies focus on the time-predictability issues of the integrated CPU-GPU architectures.

The measurement-based methods [21][22] are proposed for GPU WCET analysis, while the proposed method in this work is based on static timing analysis. A WCET analysis method for GPU L1 data cache is proposed in [27]. A timing model for GPU WCET analysis is proposed in [28] for a GPU system without L1 or L2 caches. By comparison, this work studies

978-1-5386-3472-1/17/$31.00 ©2017 IEEE

the timing analysis of the shared data LLC in the integrated architecture, which can lead to a more realistic and tight timing model.

III. Reuse Distance

The analysis method proposed in this work is based on the *Reuse Distance* theory. The metric of *Reuse Distance* [23] can be used to analyze the cache behaviors in CPU or GPU programs[24][25]. For set-associative cache memories, the reuse distance of a cache access *A* can be defined as the number of unique cache accesses that are mapped to the same cache set with *A* but with different tag values from *A* since the last access of *A*. For the very first access to a certain address, the reuse distance is infinity. Assuming the associativity is *N*, in an LRU cache, a cache access with the reuse distance less than *N* will be a hit, otherwise it will be a miss.

TABLE I: An Example of Reuse Distance.[23]

Access	0	1	2	3	4	5	6	7
Address	A	B	C	D	A	C	E	B
Reuse Distance	∞	∞	∞	∞	3	2	∞	4

For instance, Table I shows a sequence of memory accesses with the addresses of *A* to *E*, which map to the same cache set but with different tag values. The reuse distance values of each access are shown in the table. Accesses *0* to *3* with addresses *A, B, C and D* have the reuse distance of infinity, since they are all the very first accesses to those addresses. So is the access *6* to the address *E*. Accesses *4* with addresses *A* has the reuse distance of 3, since there are accesses to 3 unique different addresses (*B, C, D*) between access *0* and access *4*. Similarly, access *5 and 7* have the reuse distance values of 2 and 4 respectively.

IV. Shared Data LLC Analysis

A. The CPU-GPU Architecture Under Analysis

In this work, the gem5-gpu [26] simulator is used as the target architecture under analysis. In the default architecture of the gem5-gpu simulator, the CPU and GPU both have their own LLC and then connect to the off-chip memory. To support the shared LLC between GPU and CPU, the memory system in the simulator is extended to include LLCs for instructions and data before going out to the off-chip memory. It should be noted that the LLC is usually used for both instruction and data. However, since the focus of this work is to analyze the shared last level data cache, the LLCs in the target architecture is separated into instruction and data LLCs.Conducting timing analysis of the hybrid LLC for the integrated architecture will be part of our future work.

B. Simple Shared Data LLC Analysis Method

Knowing the order of the memory accesses to the cache is important in using the reuse distance to predict cache hit and miss. In the example of the sequence of memory access addresses in Table I, if the access order between access *3* and *4* is not for sure, i.e. access *4* with address *A* can possibly be either before or after access *3* with address *D*, the reuse distance of access *4* with address *A* then can be either 2 or 3. If there are many accesses whose access order to the cache can not be known for sure, there can be many possibilities in the reuse distance results.

In the worst-case timing analysis for caches, the maximum reuse distance of each access to the cache needs to be estimated, so that it can be compared with the associativity of the cache to predict whether the access is a hit or not. For instance, in Table I if the access order of the accesses *4* to *7* is not known, access *4* in the table can become the last access in the sequence, in which case the reuse distance of this access will be 4 rather than 3 of its current position. If the associativity of the cache under analysis is 4, the change of the reuse distance calculation from 3 to 4 will make the prediction of this access from hit to miss. This shows how the uncertainty in the access order can lead to overestimation in cache miss rates.

Unfortunately, for the shared data LLC in the integrated CPU-GPU architecture, the order of accesses from different CPU and GPU cores to the shared LLC is hard to predict statically. This is because the executions of the GPU kernels and the CPU programs are independent of each other. In other words, while the order of the accesses from the same GPU or CPU core can be analyzed and predicted statically based on the content of the code, the orders of the accesses among different cores are mostly based on the run-time execution and warp/thread scheduling.

	Core 0	Core 1	Core 2
	C0_A	C1_A	C2_E
	C0_B	C1_D	C2_B
T0	**C0_C**	C1_C	C2_D
	C0_D	C1_E	C2_A
	C0_A	C1_A	C2_C
T1	**C0_C**	C1_B	C2_B
	C0_B	C1_D	C2_A
	C0_E	C1_C	C2_D
	.....	.....	.....

Fig. 1: An Example of Accesses From Different Cores

Fig. 1 shows an example of how the accesses to shared LLC from different cores can affect the estimation of the reuse distance of one access. There are three cores *0* to *2*, each of which has a sequence of accesses to the shared LLC as shown in the figure. The reuse distance, for instance, of the access $C0\_C$ on *Core 0* at the time point *T1* depends on the accesses that happen between the time point *T0* and *T1*. If there is only one core, then the accesses in the gray area in the column *Core 0* are enough to predict the reuse distance and the hit/miss results. However, there are 2 other cores which access the shared LLC simultaneously and independently. In this case, to find the safe upper bound of the cache miss rate, all the accesses in the gray area under the three columns need to be considered as the possible accesses to the shared LLC

978-1-5386-3472-1/17/$31.00 ©2017 IEEE

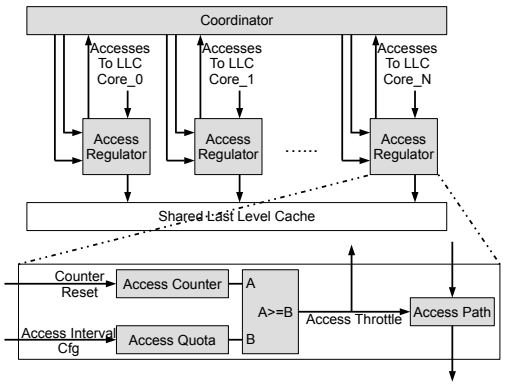

Fig. 2: Architectural Extensions for Access Interval Regulation

between time point *T0* and *T1*. Obviously, this analysis method simply takes all the accesses from other cores and the accesses in between from the current core to estimate the worst-case reuse distance. Therefore, it is referred as the *Simple* method.

Table II shows the miss rate estimation results using the *Simple* method. These results are from 8 GPU kernel benchmarks running on the gem5-gpu simulator with a shared data LLC of 512 KB. The cache line size is 128B and the associativity is 32. The simulator is configured to have 15 GPU SMs (Streaming Multiprocessors) and 1 CPU core. The results show that, except for the first two GPU kernels, the overestimation in the miss rate is very high. This is because all the accesses from other cores are considered as possible conflicting accesses in estimating the reuse distance. We also find that the first two benchmarks have less overestimation because they have much less total numbers of accesses than the others.

C. Access Interval Based Shared Data LLC Analysis Method

Although the *Simple* method introduced in SectionIV-B is straightforward and easy to implement, the overestimation can be very high. The major reason is that too many accesses from other cores are considered as possible conflicts. Based on the comparison between the results of first two kernels and the others, the results indicate that limiting the number of total accesses in reuse distance and hit/miss estimation may help to reduce the overestimation.

Due to the large number of GPU SMs in the integrated architecture, the number of possible conflicting LLC accesses can be significantly overestimated. To address this problem, we propose the *Access Interval* based analysis method to enable tighter WCET analysis of the data LLC in the integrated CPU-GPU without significant impact on the average-case performance. Some architectural extensions are needed in this *Access Interval* based method, as shown in Fig. 2. Specifically, each core in the system will be assigned with a quota of the number of accesses that this core is allowed to send to the shared LLC during each access interval. If the quota is reached, the path of sending accesses to the shared LLC is throttled. When all the active cores have reached the quota, the coordinator resets the access counter and the next interval begins.

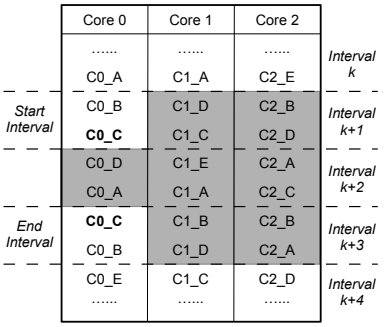

Fig. 3: An Example of Access Interval Based Method

Fig. 3 shows a simple example to illustrate the access interval based method. In this example, the quota of each access interval is set to 2 accesses. Then, for the estimation of the access $C0\_C$ in the interval *k+3*, the interval that this access belongs to is set as the *End Interval*. The interval that has the latest previous access to the same cache line is set as the *Start Interval*, e.g. interval *k+1* in this example. Then the possible conflicting accesses are the accesses from the *Start Interval* and *End Interval* from all the cores, except (1) the accesses from the core that has the latest previous access and that are also earlier than the latest previous access in the *Start Interval* and (2) the accesses from the core that has the access under analysis and that are also later than the access under analysis in the *End Interval*. In this example they are the accesses in the gray area.

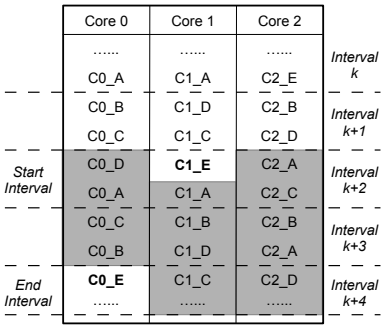

Fig. 4: An Example of Access Interval Based Method

It should be noted that the latest previous access to the same cache line can be from other cores, and the *Start Interval* should be set accordingly, as shown in Fig. 4. This example assumes that the access $C1\_E$ is the latest previous access of the access $C0\_E$. Then the possible conflicting accesses are as shown in the gray area in the figure.

The comparison between Fig. 1, 3 and 4 shows that the number of possible conflicting accesses is largely reduced by the *Access Interval* based method. Therefore, this *Access Interval* based method is likely to lead to a much tighter WCET estimation for the data LLC of the integrated CPU-GPU. Also, since different SMs execute the same GPU kernel

978-1-5386-3472-1/17/$31.00 ©2017 IEEE

TABLE II: Shared Data LLC (512KB) Miss Rate Estimations of the *Simple* Method

GPU Kernels	1	2	3	4	5	6	7	8
Actual Number of Misses	304	608	643	2311	521	2084	3179	24816
Estimated Number of Misses	372	737	2330	12499	1641	7610	15514	55524
Total Number of Access	670	1292	3228	12499	2609	11363	15514	55524
Actual Miss Rate	45.4%	47.1%	19.9%	18.5%	20.0%	18.3%	20.5%	44.7%
Estimated Miss Rate	55.5%	57.0%	72.2%	100.0%	62.9%	67.0%	100.0%	100.0%

code and, thus, generally have similar access patterns to the memory system (e.g. when memory access happens along the kernel execution), the overhead introduced by this *Access Interval* based method is expected to be small, i.e. it does not significantly impact the performance of the system, as shown by the evaluation results.

V. WCET Analysis of GPU Kernels with Shared Data LLC Estimation Results

A timing model for WCET analysis of GPU kernels was proposed in [28]. In this timing model, the latency of each instruction is divided into two parts, the issuing latency *LI* and the execution latency *LE*. The issuing latencies can not overlap with each other, while the execution latencies can overlap with the issuing and execution latencies of other warps, as shown in Fig. 5. Based on this timing model, the WCET of a GPU kernel can be derived using Equation 1, 2 and 3 (see [28] for more details). However, in [28], the timing model assumes that there is no L1 or L2 caches and all memory accesses go to the off-chip memory directly.

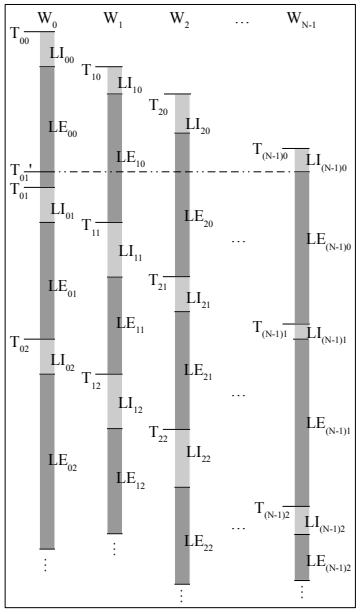

Fig. 5: Timing Model for WCET Analysis[28]

With the shared data LLC analysis method based on the *Access Interval* technique proposed in this work, this WCET timing model can be improved to analyze the GPU system with L1 and L2 data caches, which is a more realistic system compared to the system without caches. In this timing model, the latency to access the memory system for global load/store instructions needs to be set. Under the assumption that there is no L1 or L2 data cache, this latency needs to cover the latency of accessing the off-chip memory and the stall latency caused by the interconnection of the network-on-chip (NoC) for every instruction.

$$\begin{aligned} &T_{00} \Leftarrow 0 \\ &T_{i0} \Leftarrow T_{i'0} + LI_{i'0} (i > 0) \\ &i' = (i-1) \end{aligned} \tag{1}$$

$$\begin{aligned} &T_{ij} \Leftarrow MAX(T_{i'k} + LI_{i'k}, T_{ij'} + LI_{ij'} + LE_{ij'}) \\ &k = (i == 0)\ ?\ (j-1)\ :\ j \\ &i' = (i == 0)\ ?\ (N-1)\ :\ (i-1) \\ &j' = j - 1 \\ &N : Number\ of\ Warps \end{aligned} \tag{2}$$

$$\begin{aligned} &T_{i(end)} = T_{ij\_last} + LI_{ij\_last} + LE_{ij\_last} \\ &WCET = MAX(T_{0(end)}, T_{1(end)}, ..., T_{N-1(end)}) \end{aligned} \tag{3}$$

However, based on the worst-case timing analysis of the data LLC, the latency of each memory instruction can be set with different values, according to whether it is predicted to be a hit or miss. Specifically, in the memory system with L1 and L2 data caches, the value can be set to the latency of an L1 hit, an L2 hit or an L2 miss. It should be noted that, besides the latencies of accessing the different levels of caches, there are still some latencies caused by the NoC in the system. However, since the focus of this work is the shared data LLC, it is assumed that the latency of the NoC is known, which will be explained in details in Section VI-A. It also should be noted that other parts of this timing model are not affected by the integration of the cache hit/miss estimations.

VI. Evaluation Results

A. Experimental Methodology

1) Simulator: As mentioned in Section IV-A, the gem5-gpu [26] simulator is used to implement and evaluate the proposed methods. The gem5-gpu simulator integrates the simulators of GPGPU-Sim [29], which simulates the GPU cores and executes the GPU kernels, and the gem5 [30], which simulates the CPU cores, executes the CPU code and launches the GPU kernels to the GPGPU-Sim simulator.

Table III shows some of the basic configuration values of the gem5-gpu simulator. Since the focus of this work is the analysis of the shared data LLC and its impact on GPU kernel analysis, the CPU part in the system is relatively simple with

978-1-5386-3472-1/17/$31.00 ©2017 IEEE

TABLE III: Configurations of the gem5-gpu Simulator

Number of SMs	15
Number of CPU Cores	1
GPU SM Clock Cycle	500 Ticks
CPU Core Clock Cycle	500 Ticks
L1 Data Cache Size	64KB
L1 Cache Line Size	128B
L1 Cache Associativity	4
L2 Data Cache Size	256KB/512KB/1024KB
L2 Cache Line Size	128B
L2 Cache Associativity	32
L1/L2 Cache Replacement Policy	LRU
GPU Warp Size	32
GPU Warp Scheduling Policy	Pure Round-Robin
Max Number of Active Warps	48
Max Number of Active Blocks	8

1 CPU core, while there are 15 GPU SMs. The periods of one clock cycle for the GPU SM and GPU core are set to 500 ticks. One tick is the basic cycle at which the whole simulator cycles. There is an L1 data cache for each GPU SM and CPU core, with the size, the cache line size and associativity as shown in the table. There are separate instruction caches, which are modified and configured as perfect caches (as this work focuses on analyzing the data LLC). All the caches use the LRU replacement policy. To enable the static timing analysis, the Pure Round-Robin warp scheduling policy is used. The other basic configurations for the GPU SMs are shown in the rest of the table, which basically follows the configuration for the Fermi architecture [31] in the GPGPU-Sim simulator.

2) Benchmarks: The GPU kernels used in the evaluations are from the Rodinia [32] benchmark suite. Table IV shows the names of the GPU kernel benchmarks and the sizes of the inputs to the kernels. The names *k1-10* are used in Section VI-B to refer to these benchmarks.

The *gaussian* benchmark solves all the variables in a linear system, computing the results row by row. The *nw* benchmark implements a nonlinear global optimization algorithm of DNA sequence alignment. The *cfd* benchmark implements a solver algorithm for 3D Euler equations. The *lud* benchmark calculates the solution of a set of linear equations by decomposing a matrix as the product of a lower triangular and an upper triangular matrix. The *srad* benchmark is a diffusion algorithm for radar and ultrasonic images.

With the access interval method, extra delays can be introduced in accessing the LLC, which can lead to performance overhead. To measure this, each benchmark is executed without the access interval regulation first to get the baseline performance results. Then, with the access interval enabled, each benchmark is executed again to get the performance results with possible performance overhead and the results of the actual miss rate in the shared data LLC, to which the estimated LLC miss rate is compared.

B. Experiment Results

1) Shared Data LLC Miss Rate Estimation Results: Fig. 6 shows the actual and estimated miss rate results of a 512KB LLC. The results show that, for different actual miss rates across the benchmarks, the proposed estimation method can provide a safe upper bound, among which only *k6* has relatively higher overestimation.

TABLE IV: Benchmarks

	Benchmark Name	Input Size
k1	cfd1	4096
k2	cfd2	4096
k3	gaussian	128
k4	gaussian	256
k5	lud	128
k6	lud	256
k7	nw	1024
k8	nw	2048
k9	srad	128
k10	srad	256

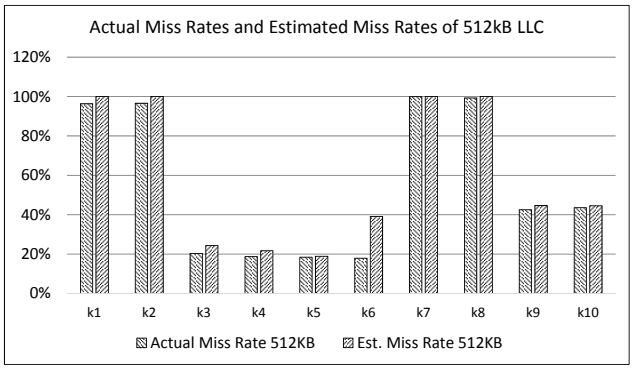

Fig. 6: Miss Rate Estimation Results of a 512KB LLC

Fig. 7 shows the actual and estimated miss rate results of 3 different LLC sizes, including 256KB, 512KB and 1024KB. As shown in the figure, for most of the kernels the overestimation reduces as the LLC size increases. For example, the overestimation in *k3* reduces from over 100% with 256KB LLC to less than 1% with 1024KB LLC. This is because that a larger LLC has more cache sets and hence the number of possible conflicting accesses that are mapped to the same set is reduced, which leads to a tighter estimation of reuse distance values.

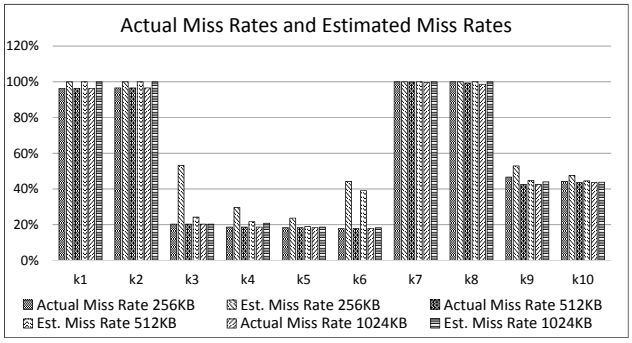

Fig. 7: Miss Rate Estimation Results of Different LLC Sizes

978-1-5386-3472-1/17/$31.00 ©2017 IEEE

2) WCET Estimation Results of GPU Kernels: Fig. 8 shows the normalized performance results of the benchmarks with different shared data LLC caches. The numbers of execution cycles are normalized to those with a 256KB LLC for each benchmark. As shown in the figure, some benchmarks benefit from larger cache sizes, while others do not. Part of the reason is that for some benchmarks, a larger LLC does not necessarily result in a lower miss rate. For those that have smaller LLC miss rates with larger LLC sizes, e.g. *k7, k8 and k9*, the performance is well improved.

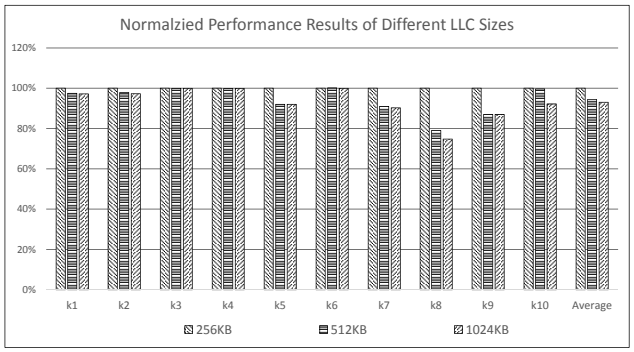

Fig. 8: Normalized Performance Results of Different LLC Sizes

Fig. 9 shows the normalized performance and WCET estimation results with different shard LLC sizes. The performance and estimation results in the figure are normalized to the actual performance results with a 256KB shared data LLC for each benchmark. The results show that high overestimation in the LLC miss rate can result in high overestimated WCET result, such as *k6* with more than 140% in overestimation of LLC miss rate and more than 35% overestimation in WCET with a 256KB LLC.

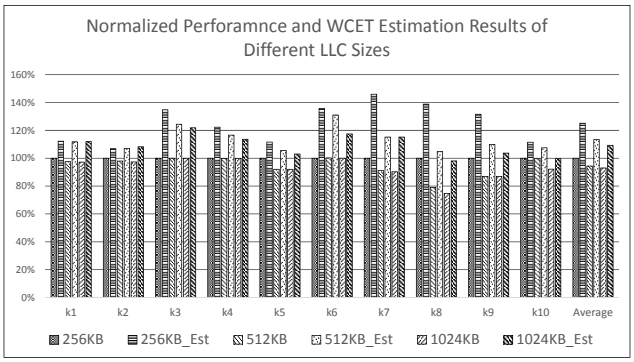

Fig. 9: Normalized WCET Estimation Results of Different LLC Sizes

It should be noted that the overestimation is also related to the ratio between the maximum and the average latencies of accessing different levels of the memory system. For example, although the overestimation of the LLC miss rate is very low for benchmark *k7* and *k8* as shown in Fig. 7, the overestimation in WCET is high (35% to 40%). This is because the ratio between the maximum and average latencies in accessing the off-chip memory is around 2.5 for these two benchmarks while this ratio is below 1.5 for other benchmarks, and the WCET analyzer has to use the maximum latency for every access in the estimation.

3) Performance Overhead of the Access Interval Based Execution Model: Fig. 10 shows the normalized performance results of the benchmarks with 3 different LLC sizes. The results are the execution cycles of the GPU kernel benchmarks with the access interval normalized to the execution cycles without the access interval regulations. The performance overhead in *k6* is higher than the others, because synchronizations are used in this kernel, together with which the access interval regulations lead to longer delays for warps to reach the synchronization barriers. As shown in the figure, the average performance overhead is less than 8%, which is not prohibitive considering the benefit of much tighter timing analysis.

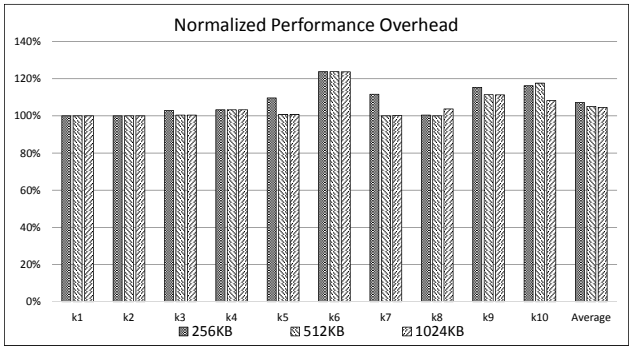

Fig. 10: Normalized Access Interval Method Performance Results of Different LLC Sizes

VII. Conclusion

The integrated CPU-GPU architectures take the advantages of tightly-coupled CPUs and GPUs to further boost performance. In such architecture, the shared LLC is an important architectural component for performance improvement and a key source of time-unpredictability as well. Since different cores access the shared LLC simultaneously, the run-time behavior of the shared LLC is hard to predict statically, if not impossible. In this work, a technique of regulating the accesses to the shared LLC by enforcing access intervals is proposed to improve the time-predictability of the LLC. The results show that the proposed technique can significantly reduce the overestimation in the miss rates of the shared data LLC, without significantly impacting the average-case performance.

In the future, we plan to develop WCET analysis methods for the Network-on-Chip interconnections and its integration to the WCET analysis of the whole system. Also, we would like to explore the possibilities of building different GPU warp scheduling policies with good time-predictability and performance.

Acknowledgment

This work was funded in part by the NSF grant CNS-1421577.

978-1-5386-3472-1/17/$31.00 ©2017 IEEE

REFERENCES

[1] Green500. *The Green500 List*. www.top500.org/green500/lists/2016/06/

[2] Advanced Micro Devices, Inc. *AMD Athlon APUs* www.amd.com/en-us/products/processors/desktop/athlon

[3] ARM Ltd. *big.LITTLE Technology* www.arm.com/products/processors/technologies/biglittleprocessing.php

[4] Nvidia. *Nvidia Tegra Mobile Processors*. www.nvidia.com/object/tegra.html.

[5] L. Sha and M. Caccamo and R. Mancuso and J. E. Kim and M. K. Yoon and R. Pellizzoni and H. Yun and R. B. Kegley and D. R. Perlman and G. Arundale and R. Bradford. *Real-Time Computing on Multicore Processors*. In: Computer 49.9 (2016), pp. 69-77.

[6] R. Mancuso and R. Dudko and E. Betti and M. Cesati and M. Caccamo and R. Pellizzoni. *Real-time cache management framework for multi-core architectures*. In: Real-Time and Embedded Technology and Applications Symposium (RTAS), 2013 IEEE 19th. Apr. 2013, pp. 45-54.

[7] B. C. Ward and J. L. Herman and C. J. Kenna and J. H. Anderson. *Making Shared Caches More Predictable on Multicore Platforms*. In: 2013 25th Euromicro Conference on Real-Time Systems. July 2013, pp. 157-167.

[8] M. Paolieri, E. Quiñones, F. J. Cazorla and G. Bernatand and M. Valero. *Hardware Support for WCET Analysis of Hard Realtime Multicore Systems*. In: Proceedings of the 36th Annual International Symposium on Computer Architecture. ISCA '09. Austin, TX, USA: ACM, 2009, pp. 57-68.

[9] S. Chattopadhyay, A. Roychoudhury and T. Mitra. *Modeling Shared Cache and Bus in Multi-cores for Timing Analysis*. In: Proceedings of the 13th International Workshop on Software & Compilers for Embedded Systems. SCOPES '10. St. Goar, Germany: ACM, 2010, 6:1-6:10.

[10] G. Bernabe, J. Cuenca, and D. Gimenez. *Optimization Techniques for 3D-FWT on Systems with Manycore GPUs and Multicore CPUs*. In: Procedia Computer Science 18 (2013), pp. 319-328.

[11] M. E. Belviranli, L. N. Bhuyan, and R. Gupta. *A Dynamic Self-scheduling Scheme for Heterogeneous Multiprocessor Architectures*. In: ACM Trans. Archit. Code Optim. 9.4 (Jan. 2013), 57:1-57:20.

[12] C. Yang, W. Xue, H. Fu, L. Gan, L. Li, Y. Xu, Y. Lu, J. Sun, G. Yang and W. Zheng. *A Peta-scalable CPU-GPU Algorithm for Global Atmospheric Simulations*. In: Proceedings of the 18th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming. PPoPP '13. Shenzhen, China: ACM, 2013, pp. 1-12.

[13] T. P. Stefanski. *Implementation of FDTD-compatible green's function on heterogeneous cpu-GPU parallel processing system*. In: Progress In Electromagnetics Research 135 (2013), pp. 297-316.

[14] J. A. Pienaar, S. Chakradhar, and A. Raghunathan. *Automatic Generation of Software Pipelines for Heterogeneous Parallel Systems*. In: Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis. SC '12. Salt Lake City, Utah: IEEE Computer Society Press, 2012, 24:1-24:12.

[15] K. Kofler, I. Grasso, B. Cosenza and T. Fahringer. *An Automatic Input-sensitive Approach for Heterogeneous Task Partitioning*. In: Proceedings of the 27th International ACM Conference on International Conference on Supercomputing. ICS '13. Eugene, Oregon, USA: ACM, 2013, pp. 149-160.

[16] W. Jiang and G. Agrawal. *MATE-CG: A Map Reduce-Like Framework for Accelerating Data-Intensive Computations on Heterogeneous Clusters*. In: Parallel Distributed Processing Symposium (IPDPS), 2012 IEEE 26th International. May 2012, pp. 644-655.

[17] T. Odajima, T. Boku, T. Hanawa, J. Lee and M. Sato. *GPU/CPUWork Sharing with Parallel Language XcalableMPdev for Parallelized Accelerated Computing*. In: 2012 41st International Conference on Parallel Processing Workshops. Sept. 2012, pp. 97-106.

[18] K. L. Spafford, J. S. Meredith, S. Lee, D. Li, P. C. Roth and J. S. Vetter. *The Tradeoffs of Fused Memory Hierarchies in Heterogeneous Computing Architectures*. In: Proceedings of the 9th Conference on Computing Frontiers. CF '12. Cagliari, Italy: ACM, 2012, pp. 103-112.

[19] M. Daga, A. M. Aji, and W. C. Feng. *On the Efficacy of a Fused CPU+GPU Processor (or APU) for Parallel Computing*. In: 2011 Symposium on Application Accelerators in High-Performance Computing. July 2011, pp. 141-149.

[20] Y. Ukidave and A. K. Ziabari and P. Mistry and G. Schirner and D. Kaeli. *Quantifying the energy efficiency of FFT on heterogeneous platforms*. In: Performance Analysis of Systems and Software (ISPASS), 2013 IEEE International Symposium on. Apr. 2013, pp. 235-244.

[21] A. Betts and A. Donaldson. *Estimating the WCET of GPU-Accelerated Applications Using Hybrid Analysis*. In Proc. of 25th Euromicro Conference on Real-Time Systems, 2013.

[22] K. Berezovskyi, et al. *WCET Measurement-based and Extreme Value Theory Characterisation of CUDA Kernels*. In Proc. of RTNS, 2014.

[23] K. Beyls and E. H. DŠHollander. *Reuse distance as a metric for cache behavior*. In: IN PROCEEDINGS OF THE IASTED CONFERENCE ON PARALLEL AND DISTRIBUTED COMPUTING AND SYSTEMS. 2001, pp. 617-662.

[24] C. Ding and Y. Zhong. *Predicting Whole-program Locality Through Reuse Distance Analysis*. In: SIGPLAN Not. 38.5 (May 2003), pp. 245-257.

[25] C. Nugteren and G. J. van den Braak and H. Corporaal and H. Bal. *A detailed GPU cache model based on reuse distance theory*. In: 2014 IEEE 20th International Symposium on High Performance Computer Architecture (HPCA). Feb. 2014, pp. 37-48.

[26] J. Power and J. Hestness and M. S. Orr and M. D. Hill and D. A. Wood. *gem5-gpu: A Heterogeneous CPU-GPU Simulator*. In: Computer Architecture Letters 13.1 (Jan. 2014).

[27] Y. Huangfu and W. Zhang. *Worst-Case Execution Time Analysis of GPU Kernels*. To appear In Proc. of the 23rd IEEE International Conference on Embedded and Real-Time Computing Systems and Applications (RTCSA), August 2017.

[28] Y. Huangfu and W. Zhang. *Static WCET Analysis of GPUs with Predictable Warp Scheduling*. In: IEEE International Symposium on Real-Time Computing (ISORC). May. 2017

[29] A. Bakhoda and G. L. Yuan and W. W. L. Fung and H. Wong and T. M. Aamodt. *Analyzing CUDA workloads using a detailed GPU simulator*. In: (2009), pp. 163-174.

[30] N. Binkert, B. Beckmann, G. Black, S. K. Reinhardt, A. Saidi, A. Basu, J. Hestness, D. R. Hower, T. Krishna, S. Sardashti, R. Sen, K. Sewell, M. Shoaib, N. Vaish, M. D. Hill and D. A. Wood. *The gem5 Simulator*. In: SIGARCH Comput. Archit. News 39.2 (Aug. 2011), pp. 1-7.

[31] NVIDIA. *NVIDIAs Next Generation CUDA Compute Architecture Fermi*. $www.nvidia.com/content/pdf/fermi\_white\_papers/nvidia\_fermi\_compute\_architecture\_whitepaper.pdf$

[32] S. Che, M. Boyer, J. Meng, D. Tarjan, J. W. Sheaffer, S. Lee, K. Skadron. *Rodinia: A benchmark suite for heterogeneous computing*. In: 2009 IEEE International Symposium on Workload Characterization (IISWC). Oct. 2009, pp. 44-54.

978-1-5386-3472-1/17/$31.00 ©2017 IEEE