JCSE Journal of Computing Science and Engineering

Regular Paper
Journal of Computing Science and Engineering,
Vol. 12, No. 4, December 2018, pp. 139-148

Estimating the Worst-Case Execution Time of the Shared Data Cache in Integrated CPU-GPU Architectures

Yijie Huangfu and Wei Zhang*
Department of Electrical and Computer Engineering, Virginia Commonwealth University, Richmond, VA, USA
{huangfuy2, wzhang4}@vcu.edu

Abstract
Integrated CPU-GPU architectures have the potential to increase performance and energy efficiency for a variety of applications, due to their tight coupling of the CPU and GPU cores. However, in order to serve hard real-time and safety-critical applications, the integrated CPU-GPU architecture must be time-predictable and worst-case execution time (WCET) analyzable. In this work, we study the shared data last-level cache (LLC) in the integrated CPU-GPU architecture and propose an *access interval*-based method in order to estimate the worst-case cache misses of the LLC. The results indicate 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 the GPU kernels running on the integrated CPU-GPU architecture.

Category: Real-Time Systems

Keywords: Graphics Processing Units (GPUs); Worst-Case Execution Time (WCET); Cache memories; Integrated CPU-GPU

I. INTRODUCTION

Graphics processing units (GPUs) have been widely used to accelerate general-purpose applications ranging from deep learning to data-parallel real-time applications. One current trend involves integrating the CPU and GPU tightly on the same chip and enable data sharing between them in the shared DRAM or caches. Such integrated CPU-GPU architectures exploit the unique strengths of both types of processing units (PUs) as well as their shared resources in order to further improve the performance, compared to that of 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 discrete CPU-GPU architecture, CPUs and GPUs can have separate memory spaces and be connected through a Peripheral Component Interconnect Express (PCIe) bus, which is referred to as a *discrete* system. GPUs and CPUs transfer data back and forth through the PCIe bus in such a system, requiring programmers to manage the data needed by both CPUs and GPUs, and this can introduce performance overheads. As a result, the *integrated* CPU-GPU architecture is proposed and implemented in order to allow for CPUs and GPUs to share the same memory space and avoid such data transfer, e.g., AMD's accelerated processing units (APUs) [2].

Heterogeneous architectures have also become increasingly popular in embedded applications, which typically have stringent constraints on power dissipation, performance, or cost. For instance, the *big.LITTLE* technology [3] combines high-performance cores and energy-efficient

Open Access http://dx.doi.org/10.5626/JCSE.2018.12.4.139 http://jcse.kiise.org

Received 10 August 2018; **Accepted** 11 October 2018
*Corresponding Author

cores in order to achieve power-optimization while delivering peak-performance capability. In addition, by the integration of the GPU and CPU architectures, the Tegra [4] processors bring general-purpose GPU computing power to embedded systems.

Real-time systems can also potentially benefit from the improved performance and energy efficiency of the integrated CPU-GPU architecture. However, the issues of time-predictability in such systems need to be addressed first, as knowing the WCET is required for hard real-time systems. One of the challenges is estimating the behavior of the shared 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 first exploring the WCET analysis of the shared data LLC in the integrated architecture, then using these analysis results to estimate 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 conflicts between different cores in the LLC [5-7] 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 (TDMA) shared bus access scheme is proposed in [9] to enable the static shared bus scheduling and shared cache conflict analysis.

Research efforts have gone toward 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 [10-13]. Some studies have focused on the compiler-level methods to automatically generate the programs for heterogeneous systems [14, 15], while others have proposed programming frameworks to utilize the resources [16, 17]. Comparisons between the discrete and integrated CPU-GPU architectures show that the integrated architecture can help reduce the performance and/or energy overheads [18-20]. However, few studies have focused on the time-predictability issues of the integrated CPU-GPU architectures.

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 timing model for GPU WCET analysis is proposed in [23] for a GPU system without L1 or L2 caches. By contrast, this work studies the timing analysis of the shared data LLC in the integrated architecture, which can lead to a more realistic and tighter timing model. There have also been research efforts [24-27] to improve the time predictability of discrete GPUs. By contrast, this work studies the method to estimate the WCET for integrated CPU-GPU.

Table 1. Example of Reuse Distance

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

III. REUSE DISTANCE

We propose applying the *Reuse Distance* theory to estimate the timing behavior of the LLC. *Reuse Distance* [28] can be used to analyze cache behaviors in CPU or GPU programs [29, 30]. For set-associative cache memories, the reuse distance of a cache access A can be defined as the number of unique cache accesses 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 a reuse distance less than N will be a hit; otherwise, it will be a miss.

For instance, Table 1 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 value of each access is shown in Table 1. Accesses 0 to 3 with addresses A, B, C, and D all have the reuse distance of infinity, since they are all the very first accesses to those addresses. Similarly, access 6 is the very first access to address E. Access 4 with address A has a reuse distance of 3, since there are accesses to three unique addresses (B, C, D) between access 0 and access 4. Similarly, accesses 5 and 7 have reuse distance values of 2 and 4, respectively.

IV. SHARED DATA LLC ANALYSIS

A. The CPU-GPU Architecture under Analysis

We use the gem5-gpu [31] simulator to study and evaluate the target architecture under analysis. In the default architecture of the gem5-gpu simulator, the CPU and GPU both have their own LLC, then connect to the off-chip memory. In order 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.

The default architecture of the gem5-gpu simulator is shown in Fig. 1, where the CPU and GPU both have their own LLC, and then connect to the off-chip memory. In order to support the shared LLC between GPU and CPU, the memory system in the simulator is modified as shown in Fig. 2, where there are 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

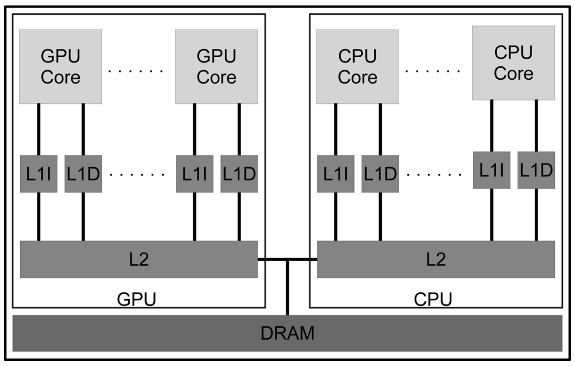


Fig. 1. The default gem5-gpu simulator architecture.

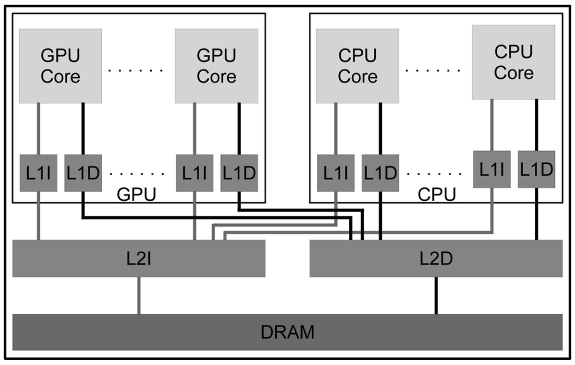


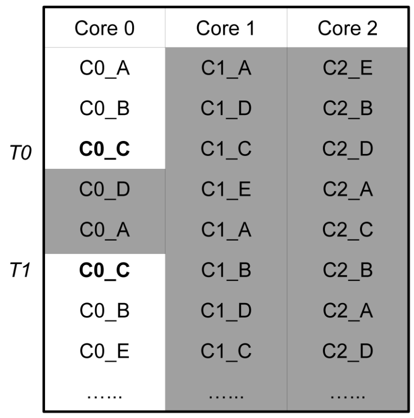
Fig. 2. Modified gem5-gpu simulator architecture with shared LLC.

and data. However, since the focus of this work is on analyzing the shared last level data cache, the LLCs in the target architecture are separated into instruction and data, as shown in Fig. 2.

B. Simple Shared Data LLC Analysis Method

It is important to know the order of the memory accesses to the cache in using the reuse distance to predict cache hit and miss. In the example of the sequence of memory access addresses in Table 1, if the access order between accesses 3 and 4 is not certain, i.e., if access 4 with address *A* can be either before or after access 3 with address *D*, the reuse distance of access 4 with address *A* can then be either 2 or 3, respectively. If there are many accesses whose access orders to the cache cannot 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 in order to predict whether the access is a hit or not. For instance, in Table 1, if the access order of accesses 4 to 7 is not known, access 4 can become the last access in the sequence, in which case the reuse distance of this access will be 4 rather than 3, as it



	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. 3. Example of accesses from different cores.

has in 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 change from hit to miss. This shows how uncertainty in the access order can lead to overestimation of 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.

Fig. 3 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 Fig. 3. The reuse distance, for instance, of the access *C*0_*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 two other cores which access the shared LLC simultaneously and independently. In this case, in order to find the safe upper bound of the cache miss rate, all of the accesses in the gray area under the three columns need to be considered as the possible accesses to the shared LLC between time point *T0* and *T1*. It is clear that this analysis method simply takes all of the accesses from other cores as well as the accesses in between the current core in order to estimate the worst-case reuse distance. Therefore, it is referred to as the *Simple* method.

Table 2 shows the miss rate estimation results using this *Simple* method. These results are from 8 GPU kernel

Table 2. Shared data LLC (512 kB) 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 accesses	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	62.9	67.0	100	100

benchmarks running on the gem5-gpu simulator with a shared data LLC of 512 kB. The cache line size is 128 B and the associativity is 32. The simulator is configured to have 15 GPU SMs (streaming multiprocessors) and one 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 of the accesses from other cores are considered to be possible conflicting accesses in estimating the reuse distance. We also find that the first two benchmarks have less overestimation because they have substantially reduced total numbers of accesses than the others.

C. Access Interval Based Shared Data LLC Analysis Method

Although the Simple method introduced in Section IV-B is straightforward and easy to implement, the resulting overestimation can be very high. The major reason for this is that too many accesses from other cores are considered to be possible conflicts. Based on the comparison between the results of the 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 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. In order to address this problem, we propose using the *Access Interval*-based analysis method to enable tighter WCET analysis of the data LLC in the integrated CPU-GPU without significantly affecting the average-case performance. Some architectural extensions are needed in this *Access Interval*-based method, as shown in Fig. 4. 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 of the active cores have reached the quota, Fig. 5 shows a simple example to illustrate the *Access Interval*-based method. In this example, the quota of each access interval is set to two accesses. Then, for the estimation of the access *C*0_*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 of the cores, except (1) the accesses from the core that has the latest previous access and the accesses 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

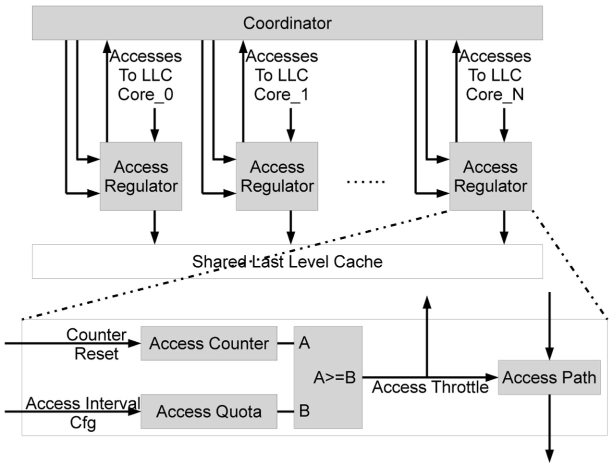


Fig. 4. Architectural extensions for access interval regulation.

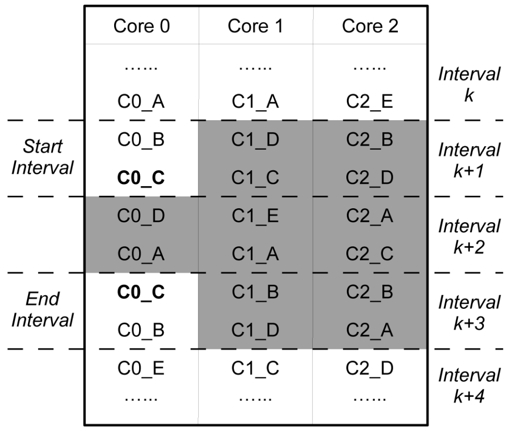


Fig. 5. An example of access interval based method.

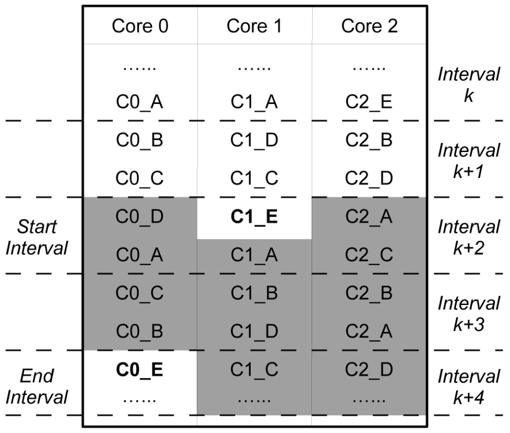


Fig. 6. An example of access interval based method.

and the accesses that are also later than the access under analysis in the *End Interval*. In this example, these are the accesses in the gray area.

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. 6. 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 Fig. 6.

The comparison between Figs. 3, 5, and 6 shows that the number of possible conflicting accesses is largely reduced using the *Access Interval*-based method. Therefore, this *Access Interval*-based method is likely to lead to substantially tighter WCET estimation for the data LLC of the integrated CPU-GPU. In addition, since different SMs execute the same GPU kernel code, and thus have generally 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 is not expected to significantly impact the performance of the system, as indicated 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 [23]. 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 cannot overlap with each other, while the execution latencies can overlap with the issuing and execution latencies of other warps, as shown in Fig. 7. Based on this timing model, the WCET of a GPU kernel can be derived using Eqs (1)–(3). Eqs. (4)–(7) elaborate on how to compute $LI_{i,j}$ and $LE_{i,j}$, which are used in Eqs (1)–(3). Eqs. (4)–(6) compute the *LI* and *LE* based on the

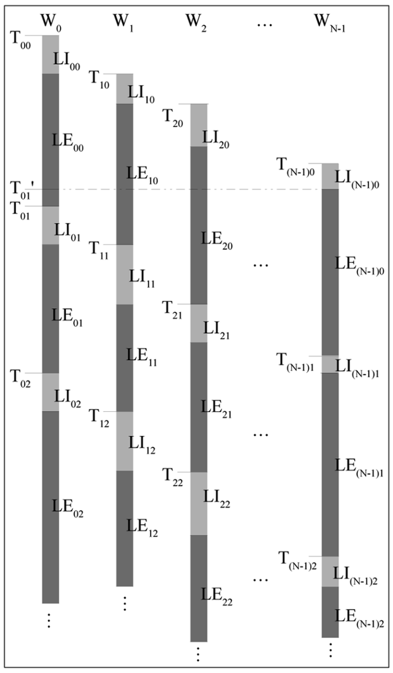

W_0 W_1 W_2 … W_{N-1}
T_{00} LI_{00} LE_{00}
T_{01}' T_{01} LI_{01} LE_{01}
T_{02} LI_{02} LE_{02}
T_{10} LI_{10} LE_{10}
T_{11} LI_{11} LE_{11}
T_{12} LI_{12} LE_{12}
T_{20} LI_{20} LE_{20}
T_{21} LI_{21} LE_{21}
T_{22} LI_{22} LE_{22}
$T_{(N-1)0}$ $LI_{(N-1)0}$ $LE_{(N-1)0}$
$T_{(N-1)1}$ $LI_{(N-1)1}$ $LE_{(N-1)1}$
$T_{(N-1)2}$ $LI_{(N-1)2}$ $LE_{(N-1)2}$

Fig. 7. Timing model for WCET analysis. Adapted from Huangfu and Zhang, "Static WCET analysis of GPUs with predictable warp scheduling," 2017 with the permission of IEEE [23].

type of instruction, i.e., arithmetic and memory, which can stall the pipeline and delay the execution in different ways. More details of the timing model can be seen in [23]. However, the timing model in [23] assumes no L1 or L2 caches (i.e., all memory accesses go to the off-chip memory directly), while this work extends the timing model to consider the cache impact on GPU WCET estimation.

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.

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

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

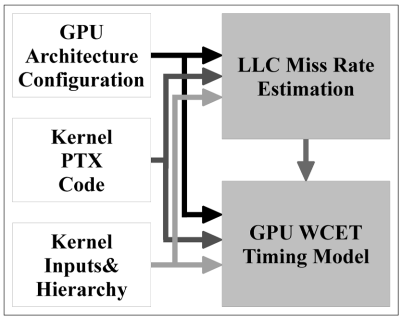


Fig. 8. The framework of the proposed WCET analysis tools.

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

However, based on the worst-case timing analysis of the data LLC, the latency of each memory instruction can be set to 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, aside from 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 detail in Section VI-A. It should also be noted that other parts of this timing model are not affected by the integration of the cache hit/miss estimations.

Fig. 8 shows the framework of the WCET analysis tools proposed in this work. The parameters of the GPU architecture configuration, the PTX GPU kernel assembly code and values of the inputs, and GPU kernel hierarchy configurations are first sent to the LLC miss rate analyzer. Then, together with the LLC miss rate estimation results, these values are sent to the WCET analyzer in order to obtain the WCET estimation results

$$LI_{inst_{Arithmetic}} = (N <= C_{pipeline})\ ?\ 0 : LI_{Stall_{Arithmetic}}$$
$$LI_{inst_{Memory}} = (N <= C_{pipeline})\ ?\ 0 : LI_{Stall_{Memory}} \tag{4}$$

$$LI_{Stall_{Arithmetic}} = L_{initiation}$$
$$LI_{Stall_{Memory}} = N_{coal} + N_{coal} \times N_{CompetingSM} \tag{5}$$

$$LE_{inst_{Arithmetic}} = Length_{pipeline} + L_{initiation} + L_{execution}$$
$$LE_{inst_{Memory}} = L_{base} + Length_{pipeline} \times (N_{coal} + N_{coal} \times N_{CompetingSM}) \tag{6}$$

$$LI_{ij} = 1 + LI_{inst_{ij}}$$
$$LE_{ij} = LE_{inst_{ij}} \tag{7}$$

Table 3. 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	64 kB
L1 cache line size	128 B
L1 cache associativity	4
L2 data cache size	256 kB / 512 kB / 1024 kB
L2 cache line size	128 B
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

VI. EVALUATION RESULTS

A. Experimental Methodology

1) Simulator

As mentioned in Section IV-A, the gem5-gpu [31] simulator is used to implement and evaluate the proposed methods. The gem5-gpu simulator integrates the simulators of GPGPU-Sim [32], which simulates the GPU cores and executes the GPU kernels, and the gem5 [33], which simulates the CPU cores, executes the CPU code, and launches the GPU kernels to the GPGPU-Sim simulator.

Table 3 shows some of the basic configuration values of the gem5-gpu simulator. Since the focus of this work is on analyzing the shared data LLC and its impact on GPU kernel analysis, the CPU part of the system is relatively simple with one 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, cache line size, and associativity as shown in Table 3. There are separate instruction caches, which are modified and configured as perfect caches (as this work focuses on analyzing the data LLC). All of the caches use the LRU replacement policy. In order 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 follow the configuration for the Fermi architecture [34] in the GPGPU-Sim simulator.

2) Benchmarks

The GPU kernels used in the evaluations are obtained

Table 4. 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

from the Rodinia [35] benchmark suite. Table 4 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 of 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.

Using the access interval method, extra delays can be introduced in accessing the LLC, which can lead to performance overhead. In order to measure this potential overhead, each benchmark is first executed without the access interval regulation so as to obtain baseline performance results. Then, with the access interval enabled, each benchmark is executed again to obtain 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. 9 shows the actual and estimated miss rate results of a 512 kB 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.

Fig. 10 shows the actual and estimated miss rate results of three different LLC sizes, including 256 kB, 512 kB, and 1024 kB. As shown in Fig. 10, for most kernels, the overestimation is reduced as the LLC size increases. For example, the overestimation in *k3* reduces from over

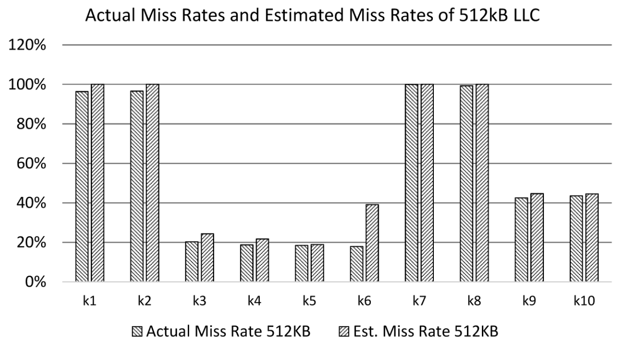


Fig. 9. Miss rate estimation results of a 512 kB LLC.

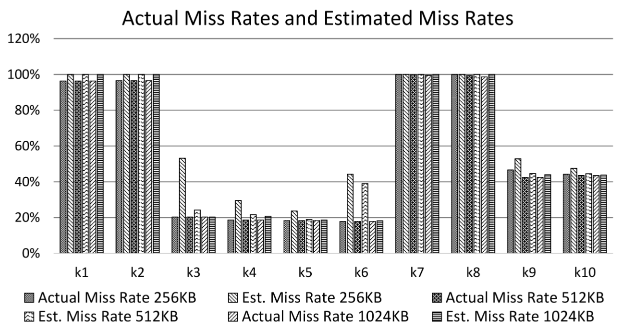


Fig. 10. Miss rate estimation results of different LLC sizes.

100% with 256 kB LLC to less than 1% with 1024 kB LLC. This is because a larger LLC has more cache sets, and hence the number of possible conflicting accesses mapped to the same set is reduced, which leads to a tighter estimation of reuse distance values.

2) WCET Estimation Results of GPU Kernels

Fig. 11 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 256 kB LLC for each benchmark. As shown in Fig. 11, some benchmarks benefit from larger cache sizes, while others do not. Part of the reason for this 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. 12 shows the normalized performance and WCET estimation results with different shared LLC sizes. The performance and estimation results in Fig. 12 are normalized to the actual performance results with a 256 kB shared data LLC for each benchmark. The results indicate that if the LLC miss rate is overestimated, it can result in a highly overestimated WCET result, such as *k6* with more than 140% in the overestimation of LLC miss rate and more than 35% overestimation in WCET with a 256 kB LLC.

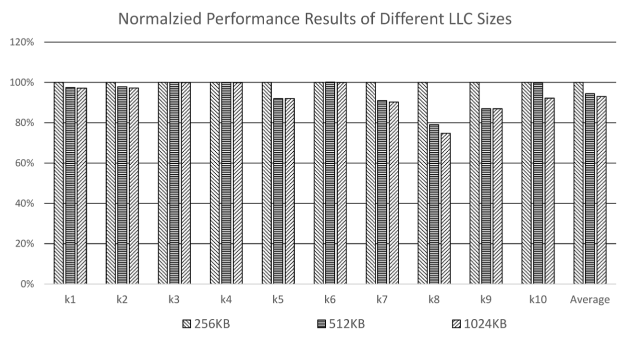


Fig. 11. Normalized performance results of different LLC sizes.

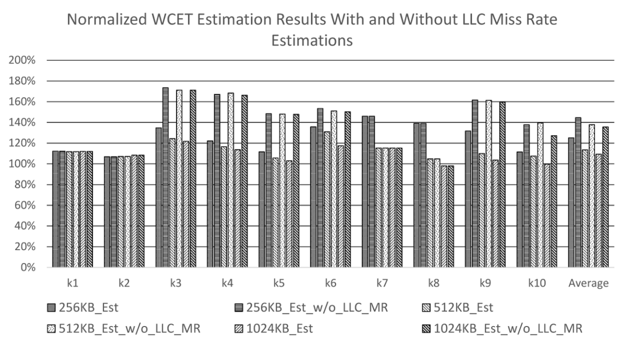


Fig. 13. Normalized WCET estimation results with and without LLC miss rate estimation.

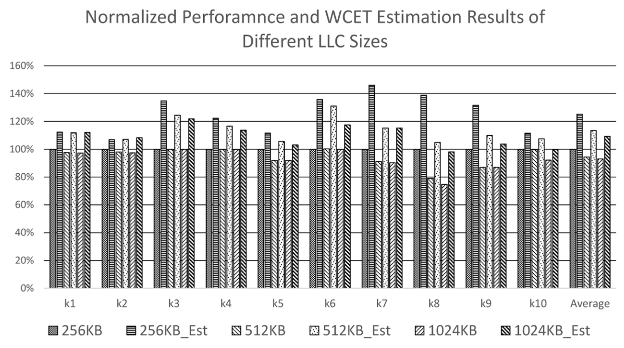


Fig. 12. Normalized WCET estimation results of different LLC sizes.

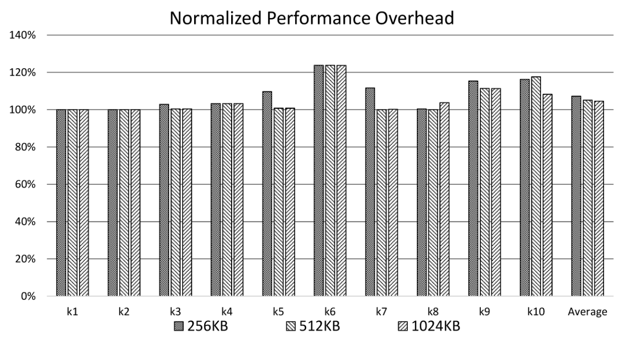


Fig. 14. Normalized access interval method performance results of different LLC sizes.

It should be noted that the overestimation is also related to the ratio between the maximum and average latencies of accessing different levels of the memory system. For example, although the overestimation of the LLC miss rate is very low for benchmarks *k7* and *k8* as shown in Fig. 10, 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.

Fig. 13 shows a comparison between the normalized WCET estimation results with and without the LLC miss rate estimations. Without the LLC miss rate estimation, the WCET analyzer assumes that all accesses need to go to off-chip memory. Therefore, as shown in Fig. 13, the overestimation is much larger. However, for *k1*, *k2*, *k7*, and *k8*, the overestimation is the same, since the LLC miss rate is very high and the estimated miss rates are 100% for these kernels.

3) Performance Overhead of the Access Interval Based Execution Model

Fig. 14 shows the normalized performance results of the benchmarks with three 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 over-head in *k6* is higher than that in 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 Fig. 14, the average performance overhead is less than 8%, which is not prohibitive considering the benefit of the much tighter timing analysis.

VII. CONCLUSION

The integrated CPU-GPU architecture has great potential to offer better performance and energy efficiency for a variety of applications. In such an architecture, the shared LLC is a crucial architectural component for performance improvement as well as a key source of time-unpredictability. Since different cores simultaneously access the shared LLC, the run-time behavior of the shared LLC is hard to predict statically, if not impossible. In order to use integrated CPU-GPU processors for real-time systems, it is necessary to estimate the WCET of the shared LLC.

In this work, we propose a technique of regulating the accesses to the shared LLC by enforcing access intervals so as 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 affecting the average-case performance. It is also shown that by integrating the miss rate estimations into the WCET timing model, the WCET estimations can be further improved.

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. We would also like to explore the possibility of building different GPU warp scheduling policies with good time-predictability and performance.

ACKNOWLEDGMENT

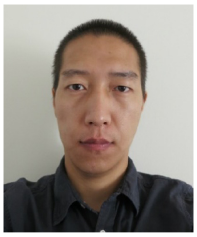
This work was funded in part by the NSF grant (No. CNS-1421577).

REFERENCES

1. Green500 List for June 2016, https://www.top500.org/green500/lists/2016/06/.
2. AMD Athlon, https://www.amd.com/en/processors/athlon-and-a-series#ATHLON.
3. Arm big.LITTLE technology, https://developer.arm.com/technologies/big-little.
4. NVIDIA Tegra Mobile Processor, https://www.nvidia.com/object/tegra.html
5. L. Sha, M. Caccamo, R. Mancuso, J. E. Kim, M. K. Yoon, R. Pellizzoni, et al., "Real-time computing on multicore processors," *Computer*, vol. 49, no. 9, pp. 69-77, 2016.
6. R. Mancuso, R. Dudko, E. Betti, M. Cesati, M. Caccamo, and R. Pellizzoni, "Real-time cache management framework for multi-core architectures," in *Proceedings of 2013 IEEE 19th Real-Time and Embedded Technology and Applications Symposium (RTAS),* Philadelphia, PA, 2013, pp. 45-54.
7. B. C. Ward, J. L. Herman, C. J. Kenna, and J. H. Anderson, "Outstanding paper award: Making shared caches more predictable on multicore platforms," in *Proceedings of 2013 25th Euromicro Conference on Real-Time Systems (ECRTS)*, Los Alamitos, CA, 2013, pp. 157-167.
8. M. Paolieri, E. Quinones, F. J. Cazorla, G. Bernat, and M. Valero, "Hardware support for WCET analysis of hard real-time multicore systems," in *Proceedings of the 36th Annual International Symposium on Computer Architecture*, Austin, TX, 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*, St. Goar, Germany, 2010.
10. G. Bernabe, J. Cuenca, and D. Gimenez, "Optimization techniques for 3D-FWT on systems with manycore GPUs and multicore CPUs," *Procedia Computer Science*, vol. 18, pp. 319-328, 2013.
11. M. E. Belviranli, L. N. Bhuyan, and R. Gupta, "A dynamic self-scheduling scheme for heterogeneous multiprocessor architectures," *ACM Transactions on Architecture and Code Optimization (TACO)*, vol. 9, no. 4, article no. 57, 2013.
12. C. Yang, W. Xue, H. Fu, L. Gan, L. Li, Y. Xu, et al., "A peta-scalable CPU-GPU algorithm for global atmospheric simulations," *ACM SIGPLAN Notices*, vol. 48, no. 8, pp. 1-12, 2013.
13. T. P. Stefanski, "Implementation of FDTD-compatible Green's function on heterogeneous CPU-GPU parallel processing system," Progress In Electromagnetics Research, vol. 135, pp. 297-316, 2013.
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*, Salt Lake City, UT, 2012.
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*, Eugene, OR, 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 *Proceedings of 2012 IEEE 26th International Parallel & Distributed Processing Symposium (IPDPS)*, Shanghai, China, 2012, pp. 644-655.
17. T. Odajima, T. Boku, T. Hanawa, J. Lee, and M. Sato, "GPU/CPU work sharing with parallel language XcalableMP-dev for parallelized accelerated computing," in *Proceedings of 2012 41st International Conference on Parallel Processing Workshops (ICPPW)*, Pittsburgh, PA, 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*, Cagliari, Italy, 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 *Proceedings of 2011 Symposium on Application Accelerators in High-Performance Computing (SAAHPC)*, Knoxville, TN, 2011, pp. 141-149.
20. Y. Ukidave, A. K. Ziabari, P. Mistry, G. Schirner, and D. Kaeli, "Quantifying the energy efficiency of FFT on heterogeneous platforms," in *Proceedings of 2013 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*, Austin, TX, 2013, pp. 235-244.
21. A. Betts and A. Donaldson, "Estimating the WCET of GPU-accelerated applications using hybrid analysis," in *Proceedings of 2013 25th Euromicro Conference on Real-Time Systems (ECRTS),* Paris, France, 2013, pp. 193-202.
22. K. Berezovskyi, L. Santinelli, K. Bletsas, and E. Tovar, "WCET measurement-based and extreme value theory characterisation of CUDA kernels," in Proceedings of the 22nd International Conference on Real-Time Networks and Systems, Versaille, France, 2014, p. 279.
23. Y. Huangfu and W. Zhang, "Static WCET analysis of GPUs with predictable warp scheduling," in *Proceedings of 2017 IEEE 20th International Symposium on Real-Time Distributed*

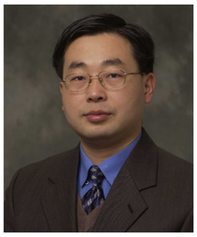
Computing (ISORC), Toronto, Canada, 2017, pp. 101-108.

24. X. Wang and W. Zhang, "Cache locking vs. partitioning for real-time computing on integrated CPU-GPU processors," in *Proceedings of 2016 IEEE 35th International Performance Computing and Communications Conference (IPCCC),* Las Vegas, NV, 2016, pp. 1-8.
25. Y. Huangfu and W. Zhang, "Warp-based load/store reordering to improve GPU data cache time predictability and performance," in *Proceedings of 2016 IEEE 19th International Symposium on Real-Time Distributed Computing (ISORC)*, York, UK, 2016, pp. 166-173.
26. Y. Huangfu and W. Zhang, "Hardware-based and hybrid L1 data cache bypassing to improve GPU performance," in *Proceedings of 2015 IEEE 12th International Conferen on Embedded Software and Systems (ICESS)*, New York, NY, 2015, pp. 972-976.
27. J. Picchi and W. Zhang, "Impact of L2 cache locking on GPU performance," in *Proceedings of the SoutheastCon 2015*, Ft. Lauderdale, FL, 2015, pp. 1-4.
28. K. Beyls and E. D'Hollander, "Reuse distance as a metric for cache behavior," in *Proceedings of the IASTED Conference on Parallel and Distributed Computing and Systems*, Anaheim, CA, 2001, pp. 617-622.
29. C. Ding and Y. Zhong, "Predicting whole-program locality through reuse distance analysis," *ACM SIGPLAN Notices*, vol. 38, no. 5, pp. 245-257, 2003.
30. C. Nugteren, G. J. Van den Braak, H. Corporaal, and H. Bal, "A detailed GPU cache model based on reuse distance theory," in *Proceedings of 2014 IEEE 20th International Symposium on High Performance Computer Architecture (HPCA),* Orlando, FL, 2014, pp. 37-48.
31. J. Power, J. Hestness, M. S. Orr, M. D. Hill, and D. A. Wood, "gem5-gpu: a heterogeneous CPU-GPU simulator," *IEEE Computer Architecture Letters*, vol. 14, no. 1, pp. 34-36, 2015.
32. A. Bakhoda, G. L. Yuan, W. W. Fung, H. Wong, and T. M. Aamodt, "Analyzing CUDA workloads using a detailed GPU simulator," in *Proceedings of IEEE International Symposium on Performance Analysis of Systems and Software*, Boston, MA, 2009, pp. 163-174.
33. N. Binkert, B. Beckmann, G. Black, S. K. Reinhardt, A. Saidi, A. Basu, et al., "The gem5 simulator," *ACM SIGARCH Computer Architecture News*, vol. 39, no. 2, pp. 1-7, 2011.
34. NVIDIA, "NVIDIA's next generation: CUDA Computer Architecture Fermi," 2009; https://www.nvidia.com/content/pdf/fermi_white_papers/nvidia_fermi_compute_architecture_whitepaper.pdf.
35. S. Che, M. Boyer, J. Meng, D. Tarjan, J. W. Sheaffer, S. H. Lee, and K. Skadron, "Rodinia: a benchmark suite for heterogeneous computing," in *Proceedings of IEEE International Symposium on Workload Characterization*, Austin, TX, 2009, pp. 44-54.



Yijie Huangfu

Yijie Huangfu received his B.S. degree in the major of Automation, and M.S. degree of Communication and Information System from the Dalian University of Technology in 2006 and 2009, respectively. He received his Ph.D. degree in Electrical and Computer Engineering from the Virginia Commonwealth University in 2017. He has become an Architecture Engineer at NVIDIA since June 2017.



Wei Zhang

Wei Zhang is a professor in the Department of Electrical and Computer Engineering at Virginia Commonwealth University. Dr. Wei Zhang received his Ph.D. from the Pennsylvania State University in 2003. From August 2003 to July 2010, Dr. Zhang worked as an assistant professor and then as an associate professor (tenured) at Southern Illinois University Carbondale. His research interests are in embedded and real-time computing systems, computer architecture, compiler, and low-power systems. Dr. Zhang has received numerous awards such as the 2016 Engineer of the Year award from RJEC (Richmond Joint Engineering Council), the 2009 SIUC Excellence through Commitment Outstanding Scholar Award for the College of Engineering, and 2007 IBM Real-time Innovation Award etc. Dr. Zhang has received 7 research grants from the National Science Foundation as the PI. In addition, his research and educational efforts have been supported by industry including leading IT companies such as IBM, Intel, Motorola, and Altera. Dr. Zhang has published more than 150 papers in refereed journals and conference proceedings. He is a senior member of the IEEE, and an associate editor of two international journals. He has served as a member of the organizing or program committees for several IEEE/ACM international conferences and workshops.