Interference Evaluation In CPU-GPU Heterogeneous Computing

Hao Wen
Electircal and Computer Engineering
Virginia Commonwealth University
Richmond, Virginia 23284
Email: wenh2@vcu.edu

Wei Zhang
Electircal and Computer Engineering
Virginia Commonwealth University
Richmond, Virginia 23284
Email: wzhang4@vcu.edu

***Abstract*—Current trend of CPU-GPU heterogeneous architecture is to integrate general purpose CPUs and highly thread-level parallelized GPUs in the same die, in which the main memory is shared between CPUs and GPUs. Shared resources contention between CPU and GPU, such as the last level cache (LLC), interconnection network and DRAM, may degrade both CPU and GPU performance. We partition the workload between CPU and GPU. Our experimental results show that the CPU performance is slowed down significantly when it runs with GPU concurrently. For the benchmarks that we have evaluated, we observe there is limited interference in the LLC, interconnection network and DRAM. The main performance bottleneck may come from the cache coherence protocol.**

I. Introduction

In the integrated CPU-GPU heterogeneous architecture, CPU and GPU applications compete for shared resources in LLC, interconnection network (or NoC, Network-on-Chip) and DRAM. In the shared LLC, conflict misses between CPU and GPU applications (also called inter-core conflict misses) may slow down both CPU and GPU applications. The cache coherent requests are increased significantly due to the high throughput GPU threads, making the cache coherent directory controller become a bottleneck. In the interconnection network, virtual channels in the router are shared between CPU and GPU, which may block each other when the CPU and GPU packets flow in the virtual channels between different ports of the router. Similarly, the memory channels in the DRAM controller are shared, making the CPU and GPU memory requests interleaved in the request buffer. The DRAM scheduling policy may unfavorably prioritize GPU requests over CPU requests, as indicated in the prior study [2]. These shared resources contentions across LLC, NoC and DRAM may degrade both CPU and GPU performance of the integrated CPU-GPU architecture. In order to see how much performance degradation is resulted from CPU-GPU interference, We divide the workloads between CPU and GPU and compare the CPU and GPU performance running concurrently with the base performance when they execute alone respectively. We observe that the GPU performance degradation is small, while the CPU suffers significant performance loss. This observation is consistent with the previous work [1]. We are trying to evaluate different part of the architecture to identify the bottleneck that leads to the CPU performance degradation.

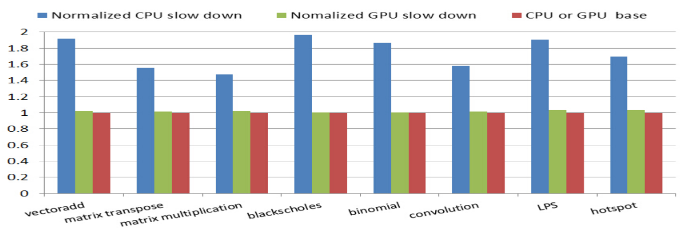

Fig. 1. Normalized CPU and GPU performance slow down

Benchmark	CPU	CPU-GPU	GPU	GPU-CPU
vectoradd	96.98%	97.11%	97%	96.87%
matrix transpose	30.95%	36.89%	68.3%	68.13%
matrix multiplication	2.9%	0.1%	5.38%	5.2%
blackscholes	91.95%	87.57%	84.62%	85.42%
binomial	56.57%	56.41%	71.21%	71.37%
convolution	49.75%	46.52 %	63.07%	63.6%
LPS	53.26%	54.07%	64.32%	63.96%
hotspot	93.14%	85.88%	89.61%	89.27%

TABLE I
LLC MISS RATE CHANGE WHEN CPU AND GPU APPLICATIONS RUN CONCURRENTLY.

II. Simulation Environment

We use gem5-gpu [3], a CPU-GPU heterogeneous simulator, to evaluate our work. The benchmarks are chosen from Rodinia [4]. We manually divide the workload between CPU and GPU. The CPU threads are created by mthreads. There are 8 CPU threads distributed to 8 CPU cores.

In the following section, the performance results are measured in terms of simulation cycles.

III. Experimental Results

A. *Last Level Cache(LLC)*

Table I shows the LLC miss rate results of different benchmarks that running on the CPU and GPU concurrently, which are compared to the base results when CPU and GPU portion execute alone. There are limited conflict misses between CPU and GPU.

B. *Interconnection Network*

Virtual Channel Partitioning (VCP) is used to reduce the traffic interference between CPU and GPU in the network.

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

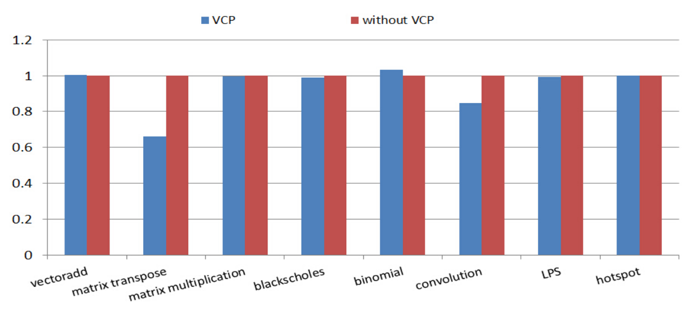

Fig. 2. Normalized CPU performance of VCP compared to non-VCP

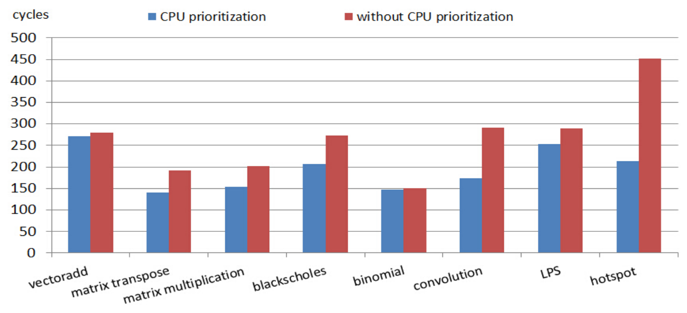

Fig. 4. reduced DRAM access latency of CPU requests after prioritization

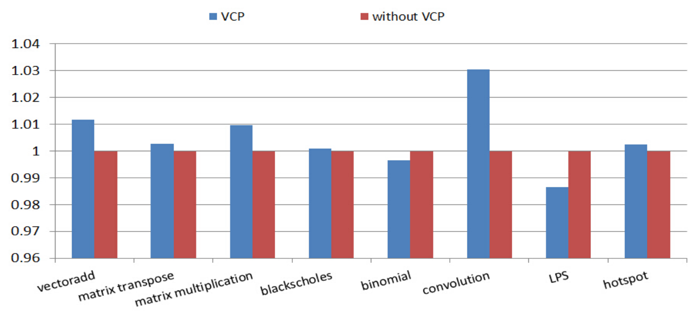

Fig. 3. Normalized GPU performance of VCP compared to non-VCP

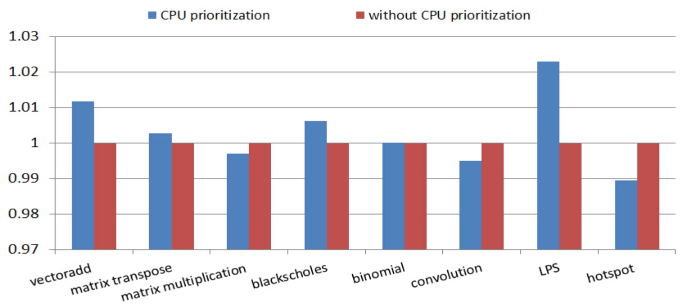

Fig. 5. Normalized CPU performance after CPU prioritization in the DRAM compared to the results without CPU prioritization

Figure 2 and Figure 3 shows the normalized CPU and GPU performance with VCP compared to the performance without VCP. Most of the CPU benchmarks do not benefit from VCP.

C. DRAM Controller

In order to see whether interferences in the DRAM is the main reason that causes CPU performance degradation. We prioritize the CPU requests in the DRAM to see how it impact the performance. Figure 4 compare the access latency of CPU requests in the DRAM before and after CPU requests prioritization, which is reduced because of higher priority compared to the original latencies. However, the CPU performance is not improved, as the Figure 5 shows, it is vary close to the original performance results without CPU prioritization. It indicates that the DRAM is not likely the bottleneck of CPU performance bottleneck.

Benchmark	CPU	GPU/CPU	CPU-GPU/CPU
vectoradd	6694	2.02	3.64
matrix transpose	3384	57.45	57.51
matrix multiplication	442723	2.41	2.37
blackscholes	4157	30.98	33.38
binomial	956	15.05	15.81
convolution	1931	51.25	55.96
LPS	16970	25.18	28.59
hotspot	3861	77.65	79.39

TABLE II
CACHE COHERENT REQUESTS INCREASES SIGNIFICANTLY WHEN CPU AND GPU RUN TOGETHER

D. Cache coherent protocol

Table II shows the cache coherent requests information in the directory. The CPU column is the number of cache coherent requests when the CPU portion workload executes alone. The other two columns represent how many times more the cache coherent requests when GPU portion executes alone and CPU/GPU run concurrently. The number of cache coherent requests increases significantly and it is very likely that the coherence directory becomes bottleneck, which results in the CPU performance degradation.

IV. ACKNOWLEDGMENT

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

REFERENCES

[1] Onur Kayiran, Nachiappan Chidambaram Nachiappan, Adwait Jog, Rachata Ausavarungnirun, Mahmut T. Kandemir, Gabriel H. Loh, Onur Mutlu, Chita R. Das, Managing GPU Concurrency in Heterogeneous Architectures, In Proc. of MICRO, 2014.
[2] Rachata Ausavarungnirun, Kevin Chang, Lavanya Subramanian, Gabriel Loh, and Onur Mutlu, Staged Memory Scheduling: Achieving High Performance and Scalability in Heterogeneous Systems, In Proc. of ISCA, 2012.
[3] Jason Power, Joel Hestness, Marc S. Orr, Mark D. Hill, David A. Wood, gem5-gpu: A Heterogeneous CPU-GPU Simulator, Computer Architecture Letters vol. 13, no. 1, Jan 2014.
[4] M. Boyer, J. Meng, D. Tarjan, J.W. Sheaffer, Sang-Ha Lee, K. Skadron,Rodinia: A benchmark suite for heterogeneous computing, In Proc. of IISWC, 2009.
[5] Vineeth Mekkat, Anup Holey, Pen-Chung Yew, Antonia Zhai, Managing Shared Last-Level Cache in a Heterogeneous Multicore Processor, In Proc. of PACT, 2013.