Performance Impact of Reconfigurable L1 Cache on GPU Devices

Sasko Ristov, Marjan Gusev
Ss. Cyril and Methodius University
Rugjer Boshkovik 16, PO Box 393, 1000 Skopje, Macedonia
Email: {sashko.ristov, marjan.gushev}@finki.ukim.mk

Leonid Djinevski, Sime Arsenovski
FON University
Av. Vojvodina, 1000 Skopje, Macedonia,
Email: {leonid.djinevski, sime.aresnovski}@fon.edu.mk

***Abstract*—The newest GPU Kepler architecture offers a reconfigurable L1 cache per Streaming Multiprocessor with different cache size and cache associativity. Both these cache parameters affect the overall performance of cache intensive algorithms, i.e. the algorithms which intensively reuse the data. In this paper, we analyze the impact of different configurations of L1 cache on execution of matrix multiplication algorithm for different problem sizes. The basis of our research is the existing theoretical analysis of performance drawbacks which appear for matrix multiplication while executed on multicore CPU. We perform series of experiments to analyze the matrix multiplication execution behavior on GPU and its set associative L1 and L2 cache memory with three different configurations: cache size of 16KB, 32KB and 48KB with appropriate set associativity of 4 and 6, respectively. The results show that only L2 cache impacts the algorithm's overall performance, particularly the L2 capacity and set associativity. However, the configuration of the L1 cache with 48KB and 6-way set associativity slightly reduces these performance drawbacks, compared to other configurations of L1 with 32KB and 16KB using 4-way cache set associativity, due to greater set associativity.**

***Index Terms*—Cache Memory, Set Associativity, GPGPU.**

I. Introduction

CACHE memory is a very important part of memory hierarchy since it reduces the performance gap between the main memory and the CPU [1]. The algorithm performance with a certain problem size depends on several cache parameters: cache size, cache replacement policy, cache levels, cache-line size, cache inclusivity, cache associativity, etc.

Today's GPU (Graphics Processing Unit) devices are more appropriate for applications with regular data access patterns [2]. They have multilevel set associative cache memory expressed with L1 and L2 level. The former is private per SM (Streaming Processor), while the latter is shared among all SMs on a single GPU device. The NVIDIA's Fermi architecture introduced size configuration (and automatically the appropriate cache set associativity) of L1 cache memory, while the newest Kepler architecture allows the programmer even further configuration.

In this paper, we configure the GPU device with three different cache sizes and two different set associativity sizes in order to determine how this new feature impacts the most common cache intensive algorithm, i.e. dense matrix multiplication (DMM). Our intention is neither to speedup the algorithm execution using the power of many core GPU, nor to speedup the algorithm using some existing transformations, but to use the DMM algorithm as a benchmark and evaluate the impact of cache sizes and associativity on the overall performance. We use only one processing unit of only one SM and realize a micro-benchmark to avoid the impact of many cores and potential additionally generated cache misses.

Since the cache set associativity can provide huge performance drawbacks for cache intensive algorithms, such as DMM, we perform additional analysis on the performance of those matrix sizes where the drawbacks are expected due to L1 and L2 cache set associativity. A performance drawback is a phenomenon where the performance does not follow the existing trend and has smaller value than the performance obtained in the neighboring points. Usually this is reflected as a negative performance peak, i.e. the performance in analyzed point x is lower than the performance in the points left or right of x, which follow a trend in performance behavior.

The goal in this research is to determine which configuration of L1 cache memory provides the best cost - performance ratio.

The rest of the paper is organized as follows. In Section II, we give an overview of related work in the area of the research problem. Analysis of possible performance drawbacks and a description of methodology used in the experiments is presented in Section III. The results of the experiments are elaborated in Section IV. Finally, we conclude our work followed by our plans for future work in Section V.

II. Related work

The latest GPUs have two level cache hierarchy organized with set cache associativity. The impact of cache associativity on GPU performance was analyzed by several authors. Performance drawbacks are likely expected for DMM execution on GPU for particular matrix sizes, due to the usage of only small subset of the cache due to the matrix storage pattern, similar to the effect on multicore architectures reported by Gusev and Ristov [3]. An example of huge performance drawbacks of DGEMM (Double precision General Matrix Multiply) for matrix size that are multiples of 1024 are reported by Matsumoto et. al [4] without deeper explanation. This problem was also analyzed by Batson and Vijakumar [5]. They propose reactive mechanisms (selective displacement and feedback) as a solution. Calder et al. propose that way prediction [6] can improve set-associative cache access times.

TABLE I
CONDITIONS FOR PERFORMANCE DRAWBACKS

L1 (16KB)				L1 (32KB)				L1 (48KB)				L2 (512KB)			
N	d	N/d	n	N	d	N/d	n	N	d	N/d	n	N	d	N/d	n
64	16	4	4	64	32	2	4	64	32	2	6	64	128	0.5	16
128	**8**	16	4	**128**	**16**	8	4	**128**	**16**	8	6	128	64	2	16
256	**4**	64	4	**256**	**8**	32	4	**256**	**8**	32	6	256	32	8	16
512	**2**	256	4	**512**	**4**	128	4	**512**	**4**	128	6	**512**	**16**	32	16
1024	**1**	1024	4	**1024**	**2**	512	4	**1024**	**2**	512	6	**1024**	**8**	128	16
2048	/	/	4	**2048**	**1**	2048	4	**2048**	**1**	2048	6	**2048**	**4**	512	16
4096	/	/	4	4096	/	/	4	4096	/	/	6	**4096**	**2**	2048	16
8192	/	/	4	8192	/	/	4	8192	/	/	6	**8192**	**1**	8192	16

Greater set associativity will reduce the cache misses, but will still not improve the performance since this will increase the cache hit access time. Padding the first element of the second matrix will amortize the performance drawback due to cache associativity [7]. Hongil [8] dynamically selects an optimized replacement policy for each cache set via workload speculation mechanism to improve the cache performance. Ding et al. [9] designed a software runtime library to include intelligence in the cache allowing the programmers to manage and optimize last level cache usage by allocating proper cache space for different data sets of different threads.

Gusev and Ristov [3] proved both theoretically and experimentally that CPU cache memory storage pattern can significantly reduce the performance of DMM execution by increasing the generation of last level cache misses due to the usage of set associative cache. By using their theorems one can determine the matrix sizes where maximum cache performance drawback in the matrix multiplication algorithm will appear due to matrix storage pattern in a n-way associative memory. Our recent research proved those theoretical results for GPU's L2 set associativity cache [10]. In this paper, we set a research problem to check validity of theoretical results and experimentally test if they hold for different configurations of L1 set associative cache in GPU architectures.

Two problems are exposed with usage of the caches, cache capacity problem refers to the lack of the resources, while the cache associativity problem refers to inefficient usage of the cache. In this paper, we are focused on performance analysis of the cache associativity problem.

III. TESTING METHODOLOGY

We use the classical DMM algorithm, where the operations are performed column-wise in order to exploit the effect of cache reuse, assuming that the matrix elements are stored in row-major order, usually used in C programming language.

This research is focused on GPUs analyzing both the cache capacity and cache associativity problems defined for multicore architectures by Gusev and Ristov [3].

Table I presents the cache parameters of the GPU model GeForce GTX 680 for each configuration of L1 cache and for L2 cache, using the theoretical analysis described in [3]. According to this analysis, we expect the performance drawbacks for bold values of d ($n < N/d$), as presented in Table I. Further on we calculate that maximum N for which the performance drawback will appear is determined as:

- $N = 1024$ for L1 cache configured with 16KB cache and 4 way set associative;
- $N = 2048$ for L1 cache configured with 32KB (4 way set) or 48KB (6 way set); and
- $N = 8192$ for 512KB L2 cache 16 way set associative.

This paper aims to confirm these theoretical results for GPUs by experimental research.

The Ubuntu 12.04 LTS operating system runs on Intel i7-3770 CPU@3.40GHz, 32GB of Kingston RAM @ 1.60GHz and NVIDIA GeForce GTX 680 GPU. The implementations of all of the experiments are compiled with the Nvidia's nvcc compiler from the CUDA 5.0 toolkit.

We conducted experiments for three different configurations of 16/32/48KB of L1 cache memory. Since the cache-line size (cbs) does not influence the equations in our theoretical analysis, there isn't any particular reason to choose a value of cbs. In our case we have chosen 128B. We also assume that the cache memories are set-associative.

Six experiments of the sequential matrix multiplication algorithm were performed in the regions around the critical matrix sizes: $N = 64, 128, 256, 512, 1024$ and 2048. The sequential implementation of the DMM runs one thread per only one active SM [11], thus the whole L1 cache is dedicated to the thread. Each experiment consists of twenty test cases for problems in the area around the critical points.

Average execution time is measured from 10 iterations, excluding the first iteration.

IV. RESULTS OF THE EXPERIMENTS

The obtained results of the experiments on the GPU to analyze the impact of different L1 cache size and set associativity configuration are presented in this section. Our focus is to observe the areas around the problem size for the points where maximum drawbacks are expected from the theoretical analysis performed in Section III. All 6 experiments are performed for three different configurations of L1 cache size, i.e., 16KB, 32KB or 48KB.

The results on performance around the area of $N = 64$ (**Experiment 1**) are depicted in the left part of Figure 1. For this example, both matrices can be stored completely in the L1 cache.

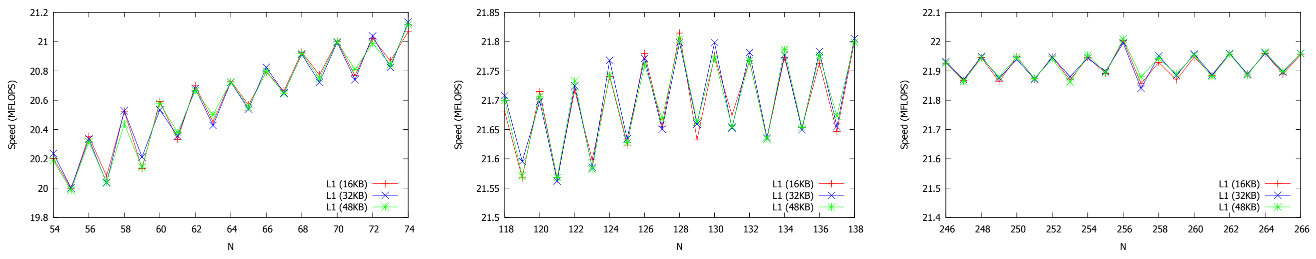

Fig. 1. Speed in the area around $N = 64$ (left), $N = 128$ (middle) and 256 (right)

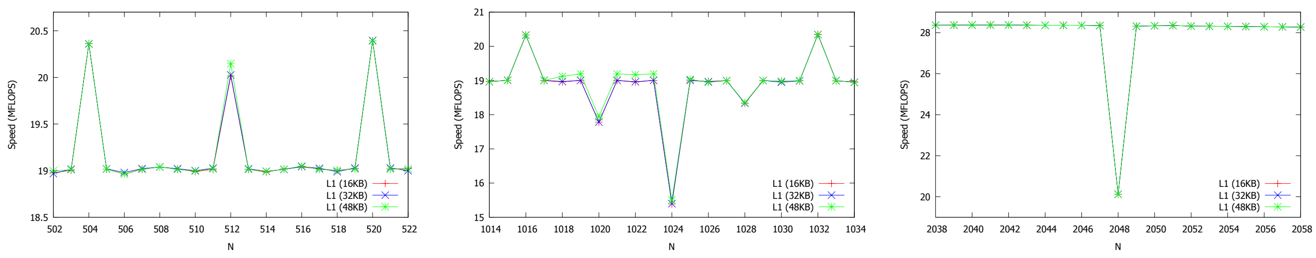

Fig. 2. Speed in the area around $N = 512$ (left), $N = 1024$ (middle) and 2048 (right)

This experiment proves our theoretical analysis since the performance drawbacks do not appear in this region. The elements of a matrix B column can be stored in a particular set and no cache misses will be generated. All three L1 cache size configuration comply with the theoretical analysis.

We observe a very strange speed curve, which increases for even matrix size N and decreases for odd matrix size N. This phenomenon appears due to the average load time for a matrix element, which is smaller for an even matrix size N. Therefore, the matrix elements fulfill the cache line more efficiently within a given cache line.

The speed in this region has a positive increasing trend due to increased amount of data reuse without repeated generation of cache misses.

Experiment 2 covers the area around matrix size $N = 128$. The middle part of Figure 1 depicts the results. The second matrix cannot be stored completely in the L1 cache for these matrix sizes and therefore drawbacks appear due to insufficient L1 cache. However, comparing it with the previous experiment, we can conclude that despite the increased number of generated L1 cache misses, the speed in the Experiment 2 is greater than the speed achieved in Experiment 1.

The results show that performance drawbacks due to L1 cache set associativity are seemingly small due to the unsatisfied condition for L2 cache set associativity, as presented in Table I. We observe a slight speed discrepancy for different L1 cache configurations using the same matrix size. Similar to the Experiment 1, performance discrepancy is observed for even and odd matrix sizes. The speed holds the positive trend as in the Experiment 1, but with smaller intensity.

The **Experiment 3** covers the area of $N = 256$. The second matrix cannot be stored completely in the L1 cache as in the Experiment 2. The speed in this region has even lighter positive trend than experiments 1 and 2, as depicted in the right part of Figure 1.

Similar to the previous case, performance drawbacks are not observed in this region, due to the smaller impact of L1 cache associativity in comparison to L2, where the set associativity problem does not appear in this region. Additionally, we observe that the performance for $N = 256$ is even greater than the values near N in that region, for each L1 cache size configuration. We explain this observation with the fact that despite the L1 cache associativity problem, the whole matrix row can be stored in the exact number of cache blocks and no L2 cache misses are generated neither due to L2 capacity nor L2 associativity problem. Therefore, the average access time is smaller for a matrix element stored in a particular cache block for $N = 256$. We observe a slightly higher speed while L1 cache is configured with 48KB.

Similar performance discrepancy is observed for even and odd matrix sizes, as in previous experiments.

The **Experiment 4** covers the area around matrix size $N = 512$ and the results of the experiment are depicted in Figure 2 (left). Matrix B cannot be stored completely neither in L1 nor L2 cache, and thus the drawback exists mainly due to their size and associativity.

Although one might think the results are strange in this region, there is an explanation. The speed for matrix size $N = 512$ is much greater than the other problem sizes in the region, except for $N = 504$ and $N = 520$. We have also tested the other close positioned points $N \in \{488, 496, 528, 536\}$ and achieved the same positive peaks. The conclusion is that the execution for $N = 512$ has performance drawback compared to these points. The observation consists of two parts: greater speed and performance drawbacks. The former appears for the same reason as explained for $N = 256$. The latter appears compared due to cache associativity and condition of Table I, compared to points $N = 504$ and $N = 520$ (and for the other points that we measured additionally).

We also observe a slightly better speed while L1 cache is configured with 48KB for $N = 512$.

Experiment 5 analyzes the area around $N = 1024$. Matrix B cannot be stored completely in L2 cache and drawbacks appear due to L2 cache size and associativity. The speed drawbacks are clearly detected and they are depicted in Figure 2 (middle).

Significant performance drawback appears as stated in Table I, but also smaller performance drawbacks appeared in points $N - 4$ and $N + 4$ (as well as for $N + 12$ and $N - 12$).

The same positive peaks are observed in the points $N + 8$ and $N - 8$ as in the region around $N = 512$.

We also observe a slightly better speed while L1 cache is configured with 48KB in the point $N = 1024$, similar to the result for $N = 512$.

Experiment 6 analyzes the area around the matrix size $N = 2048$ where matrix B also cannot be stored completely in the L2 cache. Performance drawback is clearly observed for $N = 2048$ as depicted in Figure 2 (right).

Neither additional positive nor negative peaks are observed in this area since the number of generated L2 cache misses is huge. The impact of L2 cache capacity problem is greater than the positive impact of cache memory to data locality in this region, i.e., loading the elements of the whole cache line while reading one element of that cache line.

V. Conclusion and Future Work

The performance of GPU general purpose application can be seriously degraded by the set associative L1/L2 caches. In this paper, we present the performance drawbacks for specific problem sizes of the DMM algorithm for different L1 cache configurations and fixed L2 cache size. We have performed series of experiments in the areas of critical problem sizes, which prove the analysis.

It is shown that the side effects of the associative cache on the CPU, as discussed in our earlier paper are also present in the GPU environment. However, this paper shows also some other interesting conclusions, due to a specific organization of caches in GPU, which is quite different from CPU (very small 1st level cache and no third level cache in comparison to CPU).

A total of six experiments were evaluated in points where theoretical results expect negative performance peaks with analysis of speed diagrams. The results show that the configuration of L1 cache size does not influence significantly on performance drawbacks, which appear for $N = 1024$ and 2048 due to L2 cache set associativity. Because L2 cache set size is enough to fit the cache storage requirements for problem sizes $N = 64, 128$ and 256, performance drawbacks are not observed, i.e., the algorithm performance depends mostly on L2 cache size, rather than L1's. The performance for $N = 256$ is even greater than the matrix size values near N in that region for all L1 cache size configuration.

An interesting phenomenon appears in the region around $N = 512$. The speed for $N = 512$ is greater than the other problem sizes in the region, except for $N = 504$ and $N = 520$. Although higher values are obtained than the neighboring points, still there is a performance drawback compared to analyzed points $N = 504$ and $N = 520$. More interestingly, we have found smaller negative peaks in the region around $N = 1024$ in points $N + 4$ and $N - 4$, as well as positive peaks in the points $N + 8$ and $N - 8$.

Another phenomenon was observed in the regions around $N = 64, 128$ and 256, i.e., the speed increases for even matrix size N and decreases for odd matrix size N. This happens due to the effect of loading the elements of the whole cache line while reading one element of the same cache line.

Probably the most important result is to report the platform impact of reconfigurable cache, i.e. what the user can choose for configuration of the L1 cache to achieve maximum processing speed and avoid associativity problems.

Future work will cover further research on these phenomenons, as well as analysis of correlation of power consumption with the L1/L2 capacity and associativity, since the results show that L1 cache size does not impact the algorithm performance, but different cache associativity configuration due to different cache size configuration can reduce the power consumption.

Also, we plan to measure the number of generated L1 and L2 cache misses to determine the performance drawbacks and performance discrepancies more precisely.

References

[1] J. L. Hennessy and D. A. Patterson, *Computer Architecture, Fifth Edition: A Quantitative Approach.* MA, USA: Elsevier, 2012.

[2] D. Tarjan, J. Meng, and K. Skadron, "Increasing memory miss tolerance for simd cores," in *Proc. of the Conf. on High Performance Computing Networking, Storage and Analysis*, ser. SC '09, 2009, pp. 22:1–22:11.

[3] M. Gusev and S. Ristov, "Performance gains and drawbacks using set associative cache," *Journal of Next Generation Information Technology (JNIT)*, vol. 3, no. 3, pp. 87–98, 31 Aug 2012.

[4] K. Matsumoto, N. Nakasato, and S. Sedukhin, "Implementing a code generator for fast matrix multiplication in opencl on the gpu," in *Embedded Multicore Socs (MCSoC), 2012 IEEE 6th International Symposium on*, sept. 2012, pp. 198–204.

[5] B. Batson and T. N. Vijaykumar, "Reactive-associative caches," in *Proceedings of the 2001 International Conference on Parallel Architectures and Compilation Techniques*, ser. PACT '01, 2001, pp. 49–60.

[6] B. Calder, D. Grunwald, and J. Emer, "Predictive sequential associative cache," in *Proceedings of the 2nd IEEE Symposium on High-Performance Computer Architecture*, ser. HPCA '96, 1996, pp. 244–253.

[7] S. Williams, L. Oliker, R. Vuduc, J. Shalf, K. Yelick, and J. Demmel, "Optimization of sparse matrix-vector multiplication on emerging multicore platforms," *Parallel Comput.*, vol. 35, no. 3, pp. 178–194, 2009.

[8] H. Yoon, T. Zhang, and M. H. Lipasti, "Sip: Speculative insertion policy for high performance caching," Computer Sciences Department University of Wisconsin-Madison, Tech. Rep. 1676, 2010.

[9] X. Ding, K. Wang, and X. Zhang, "Ulcc: a user-level facility for optimizing shared cache performance on multicores," in *Proceedings of the 16th ACM symposium on Principles and practice of parallel programming*, ser. PPoPP '11. ACM, 2011, pp. 103–112.

[10] L. Djinevski, S. Arsenovski, S. Ristov, and M. Gusev, "Performance drawbacks for matrix multiplication using set associative cache in gpu devices," in *MIPRO, 2013 Proceedings of the 36th International Convention, IEEE Conference Publications*, Croatia, 2013, pp. 213–218.

[11] L. Djinevski, S. Ristov, and M. Gusev, "Superlinear speedup for matrix multiplication in gpu devices," in *ICT Innovations 2012*, ser. AISC. Springer Berlin Heidelberg, 2013, vol. 207, pp. 285–294.