Optimal Configuration of GPU Cache Memory to Maximize the Performance

Leonid Djinevski[1], Sime Arsenovski[1], Sasko Ristov[2], and Marjan Gusev[2]

[1] FON University,
Av. Vojvodina, 1000 Skopje, Macedonia,
leonid.djinevski@fon.edu.mk, sime.aresnovski@fon.edu.mk
[2] Ss. Cyril and Methodious University,
Rugjer Boshkovikj 16, 1000 Skopje, Macedonia
sashko.ristov@finki.ukim.mk, marjan.gushev@finki.ukim.mk

Abstract. GPU devices offer great performance when dealing with algorithms that require intense computational resources. A developer can configure the L1 cache memory of the latest GPU Kepler architecture with different cache size and cache set associativity, per Streaming Multiprocessors (SM). The performance of the computation intensive algorithms can be affected by these cache parameters. In this paper, we evaluate the influence of the performance for all possible configurations of L1 cache size and associativity, for dense matrix-matrix multiplication algorithm for various problem sizes. The results show a small impact of various L1 cache memory configurations for the overall performance of the algorithm.

Keywords: Cache Memory, SIMD, GPGPU.

1 Introduction

Today's world of "big data" requires computing resources and appropriate algorithms, which are capable to crunch massive quantities of data in a reasonable time. The instructions upon the big data provide enormous I/O operations, creating a bottleneck in the memory controllers. The hardware designers introduced the cache memory concept in order to reduce the gap between the computing elements and the main memory [10]. The cache memory can provide significant speedup for specific algorithms. If an algorithm reuses the data (cache intensive algorithm), which can be stored in the cache memory, then the data access time will be reduced. Another example is the data locality feature of an algorithm, which also utilizes the cache memory efficiently and effectively.

Today's multi-chip and multi-core CPUs, and many-core GPUs have a multilevel cache memory with different cache architectures and organizations, i.e., different cache parameters: cache size, cache replacement policy, cache levels, cache-line size, cache inclusivity, cache associativity, etc. Cache memory can be either shared among several cores, or dedicated per core. Modern multiprocessors (CPUs) have three cache levels (L1, L2 and L3), while GPUs have two cache levels (L1 and L2). Both devices use set associative cache.

GPUs achieve much greater performance than the CPUs, even though they are in the same or in the neighboring price range. However, GPU devices achieve the best efficiency for applications with regular data access patterns [22]. The Nvidia's Fermi (also supported in the latest Kepler architecture) introduced a configurable L1 cache size (and automatically the set associativity), as well as the cache line size. Different configurations of cache memory can have a great impact on a cache intensive algorithm, both positive and negative. It also impacts on the electricity cost since the cache set associativity impacts on GPU's energy consumption [5].

In this paper, we analyze the performance of a dense matrix-matrix multiplication executed on GPU using different L1 cache configurations in order to determine the optimal configuration of GPU cache memory, which achieves the best performance. We have set a hypothesis H1 that greater cache memory will provide better performance, since greater matrices can be stored in L1 cache, thus generating smaller number of cache misses. The focus is given on the regions where the dense matrix-matrix multiplication provides performance drawbacks due to cache associative and capacity problems [9].

The rest of the paper is organized as follows. In Section 2 we give overview of related work in the area of the research problem. Section 3 briefly presents the GPU memory architecture. A description of the methodology used in the experiments is presented in Section 4. The results of the experiments are elaborated in Section 5. Finally, we conclude our work followed by our plans for future work in Section 6.

2 Related work

Many authors analyzed and proposed techniques to improve the data access and thus the overall performance of a cache intensive algorithm. Reactive mechanisms (selective displacement and feedback) [2] and way prediction [3] can improve set-associative cache access times. Ahamed and Magoules determined that the performance of the matrix multiplication algorithm strongly depends of the matrix elements storage pattern, when executed on GPU [4]. A mechanism to avoid cache race and cache split is proposed by Likun and Dingfang [12], which improves an algorithm's speed on GPU. Tang et al. [21] modeled the cache misses and optimized the cache locality on GPU. Volkov and Demmel [23] analyzed many operations on dense matrices and conducted detailed benchmarks of the GPU memory system, kernel start-up costs, and arithmetic throughput.

Several papers addressed the impact of various cache parameters on cache intensive algorithms. Anchev et al. [1] determined the optimal cache replacement policy for dense matrix-matrix multiplication algorithm. Matrix multiplication can suffer from cache set associativity for particular matrix size [9]. Williams et al. [24] introduced padding of the first element of the second matrix to amortize the performance negative peaks due to cache set associativity. Hongil [25] selected dynamically an optimized replacement policy for each cache set via workload speculation mechanism to improve the cache performance.

A comprehensive analysis about performance drawbacks of matrix-matrix multiplication algorithm is conducted by Gusev and Ristov [9]. Using their theorems for CPUs, Djinevski et al. [6] proved experimentally the existence of performance drawbacks for particular matrix size for set associative cache in GPU architectures. They have benchmarked the algorithm on a Kepler GPU configured with 32KB L1 cache size and 4 way set associativity. In this paper, we extend the research and compare the performance of matrix-matrix multiplication algorithm while executed on GPU, which is configured with 16KB and 48KB L1 memory cache size.

Several authors determined performance drops of matrix-matrix multiplication on GPUs. Matsumoto et. al [13] determined huge performance drops of DGEMM for matrix size that are in multiples of 1024 without deeper explanation.

3 Memory hierarchy of the GPU Architecture

GPU devices are quite popular when designing hardware infrastructure dedicated for intensive computational applications, which is based on the the high performance/cost ratio. However, one should be careful when designing GPU solutions, since many applications do not utilize advanced optimizations [11], such as auto-vectorization, memory alignment, optimization at program design, etc.

The GPU devices are characterized as SIMD parallel machines [8] with convenient memory hierarchy. There are two programming models for development of application that utilize GPU resources: CUDA programming model [18], which is proprietary for Nvidia Corporation and OpenCL [15], which is an open standard, without royalty, hardware agnostic, and platform independent.

The many-core architecture of the GPU devices consists an array of Streaming Multiprocessors (SMs), each containing up to 192 Scalar Processors (SPs) for the latest Kepler architecture [17]. Additionally, Single Instruction Multiple Thread (SIMT) programming model [16] is supported in CUDA, where a single instruction is executed by all threads only within one SM. However, threads from different SMs execute instructions independently from each other.

The first level of memory hierarchy is a 64KB configurable memory by varying the L1 cache memory and shared memory. There are maximum 3 possible configurations that can be obtained with the following sizes 16/32/48KB for L1 and 48/32/16KB respectively for the shared memory. The other levels of the memory hierarchy are the L2 cache memory and global memory. These are off-chip memories with fixed sizes. However, the cache-line (cache block) size is configurable. It can be either 64B or 128B, and as a parameter influences the performance of the other memories.

Technical details from the manufacturer are very obscure when it comes to cache associativity of the L1 and L2 cache memories on GPU devices. However, the research community developed few micro-benchmarks like [23, 19, 14] which help towards understanding the memory hierarchy of GPU devices. To our knowledge, there are not any published analysis regarding the influence of cache associativity on GPU performance.

4 Testing Methodology

In this section, we describe the used testing methodology. It is based on a series of experiments realized in order to determine the performance negative peaks due to cache set associativity problem.

4.1 Matrix Multiplication Algorithm

We use matrices with equal matrix size $N \cdot N$ for simplicity. The result product matrix $C = [c_{ij}]$ is defined in (1) by multiplying matrices $A = [a_{ij}]$ and $B = [b_{ij}]$ where $i, j = 0, 1, \dots N - 1$.

$$C_{N \cdot N} = A_{N \cdot N} \cdot B_{N \cdot N}, \quad c_{ij} = \sum_{k=0}^{N-1} a_{ik} \cdot b_{kj} \tag{1}$$

This research is focused both on the cache capacity and the cache associativity problems.

4.2 Testing Environment

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.

4.3 Experiments and Test Cases

Since we are testing the impact of the L1 cache size and associativity, the overall performance of the dense matrix-matrix multiplication algorithm is conducted per SM. Hence, we perform micro-benchmarking of the GPU architecture by running one thread per only one active SM [7], enabling an environment where the whole L1 cache is dedicated to the thread.

Additionally, we test the influence of the L1 cache size by conducting 3 experiments of the sequential dense matrix-matrix multiplication algorithm for different configurations 16/32/48KB of the L1 cache memory size. Each experiment consists of test cases for problem sizes starting from 8 elements, up to 2058 elements. In the range (8 - 1034) all tests were performed with step of (1) element, while for the range (1035 - 2054) the step was 4 elements. The second range were performed with higher step, in order to obtain the results in a reasonable time.

4.4 Test Data

To maintain more precise values, each test case is executed in 10 iterations (excluding the first iteration). Then the execution time is calculated as average of all 10 measured times and the speed as defined in (2), since the number of operations is $2 \cdot N^3$.

$$V = \frac{2 \cdot N^3}{T_{avg} \cdot 10^6} \ (MFlops) \tag{2}$$

Denote by V_{16} the speed for configuration of L1 cache with 16KB, and by V_{32} and V_{48} the speed for configurations of L1 cache size of 32KB and 48KB respectively. Our target is comparison of the performances and thus we will analyze *relative speedup* indicators for all pairs of configurations. Relation (3) presents the relative speedup achieved by 32KB L1 cache over 16KB configuration, while 4) and (5) present the relative speedup achieved by 48KB L1 cache over 32KB and 16KB L1 caches correspondingly.

$$S_{32R16} = \frac{V_{32}}{V_{16}} \tag{3}$$

$$S_{48R32} = \frac{V_{48}}{V_{32}} \tag{4}$$

$$S_{48R16} = \frac{V_{48}}{V_{16}} \tag{5}$$

5 Results of the Experiments

This section presents the results obtained from the experiments described in the testing methodology. Our goal is to observe the overall performance of the dense matrix-matrix multiplication algorithm. In Figure 1 we present the measured speeds for the three experiments.

We can conclude that all three experiments show comparable results, as presented in Figure 1. It means that the L1 cache size does not impact the performance a lot and the measured speed is different within a range of less than 1%.

Three regions are clearly observed depending on matrix size N, i.e., L_1, L_2 and L_3 [6]. The L_1 region is defined as a region where both matrices can be stored in the L1 cache and no (or a small number of) L1 cache misses are generated. The L_2 region is defined as a region where both matrices cannot be stored in L1 cache, but can be stored in L2 cache completely. In this region, a lot of L1 cache misses are generated, while no (or a small number of) L2 cache misses are generated. Finally, the L_3 region is a region where both matrices cannot be stored in L2 cache memory and a lot of L1 and L2 cache misses are generated.

The L_1 region is specified with $N \leq 50$ and the speed in this region increases enormous when N increases. The speed continues to increases in the L_2 region, which is defined in the range $50 < N \leq 350$. We observe a phenomenon about

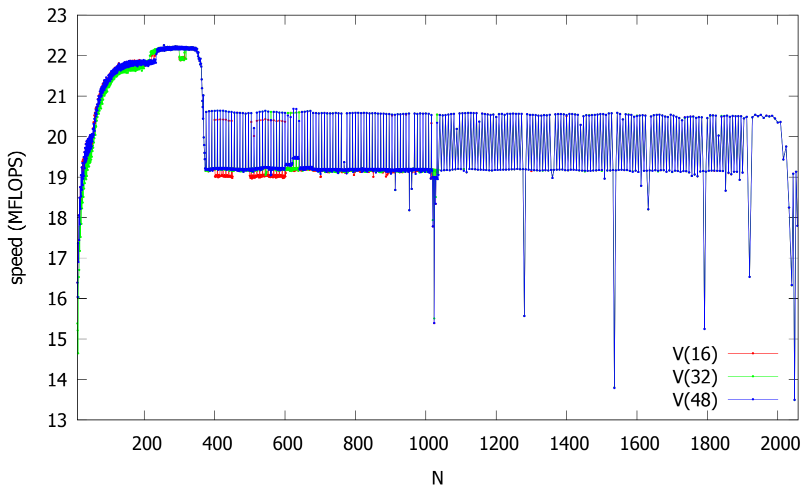

Fig. 1: Measured speed for all experiments.

the speed in L_1 and L_2 regions. That is, the speed in the L_2 region is greater than the speed in the L_1 region, despite the increased number of L_1 cache misses in the L_2 region. Let's explain this phenomenon. Each element of both matrices is accessed N times; once from the main memory and $N-1$ times from L2 cache in both cases. Thus, increasing N, the average access time tends to L2 hit time in L_2 region, while it is in the range of main memory access time in L_1 region. Finally, the speed saturates in L_3 region, which is defined in the range $N > 350$. The speed in this region is smaller than the speed in the L_2 region since the number of cache both L1 and L2 cache misses increases.

The analysis shows a lot of performance drops, and especially huge for $N =$ 1024, 1280, 1536, 1792 and 2048, that is, starting from $N = 1024$ on a step of 256. The appear due to associativity of both cache levels. We have provided a lot of details about associativity problems in caches and conflict cache misses in [20].

In addition, we provide more detailed analysis around $N = 64, 128, 256$, 512, 1024 and 2048 to find out the effect of associativity with different L1 cache sizes. Figures 2, 3 and 4 present relative speed in neighborhood around analyzed points, when 16KB L1 cache is considered as baseline, that is, it is presented with a line with value of 1. The other two configurations are presented as relative speed indicators S_{48R16} and S_{32R16}, correspondingly for confirmations of 48KB L1 and 32 KB L1 cache.

Detailed analysis of relative speeds around points where associativity impact is expected shows the following. The influence of conflict misses due to associativity is the smallest in the configuration with 48KB for all analyzed points, while for $N = 2048$ the effect is the same.

Additionally, we evaluate the relative speedup indicator for different configurations. Figure 5 presents the S_{32R16} indicator and calculates how much the

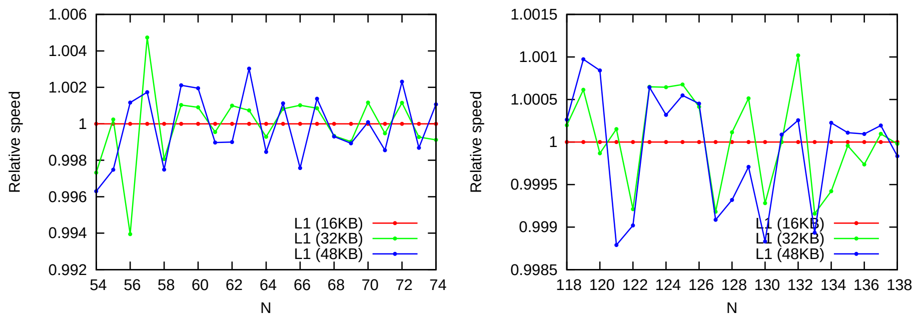

Fig. 2: Relative speed in the area around $N = 64$ (left), and $N = 128$ (right)

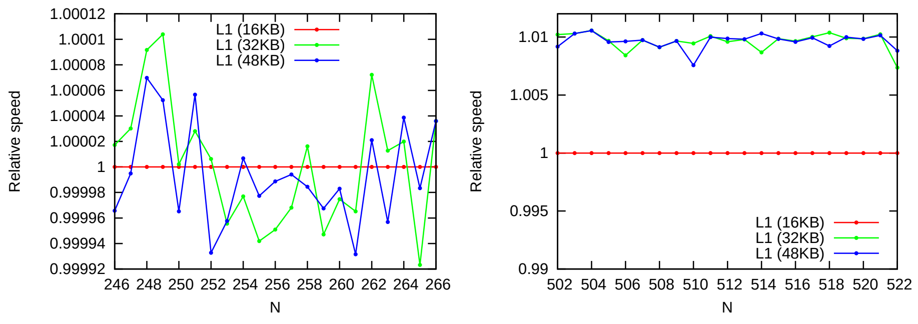

Fig. 3: Relative speed in the area around $N = 256$ (left), and $N = 512$ (right)

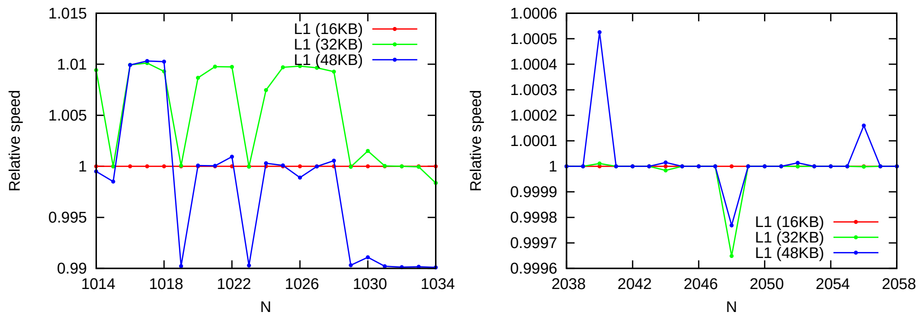

Fig. 4: Relative speed in the area around $N = 1024$ (left), and $N = 2048$ (right)

performance of the 32KB L1 cache configuration is better over the 16KB configuration.

Figures 6 and 7 present how much he 48KB L1 cache is better over 16KB L1 cache and 32KB cache correspondingly, that is, they present overall behavior of S_{48R16} and S_{48R32} indicators. We can observe that 48KB L1 configuration is even slower for values up to $N = 224$ in comparison to the 32KB L1 configuration. The best overall performance is obtained by the 32KB L1 configuration in comparison to the other for the region up to $N = 224$.

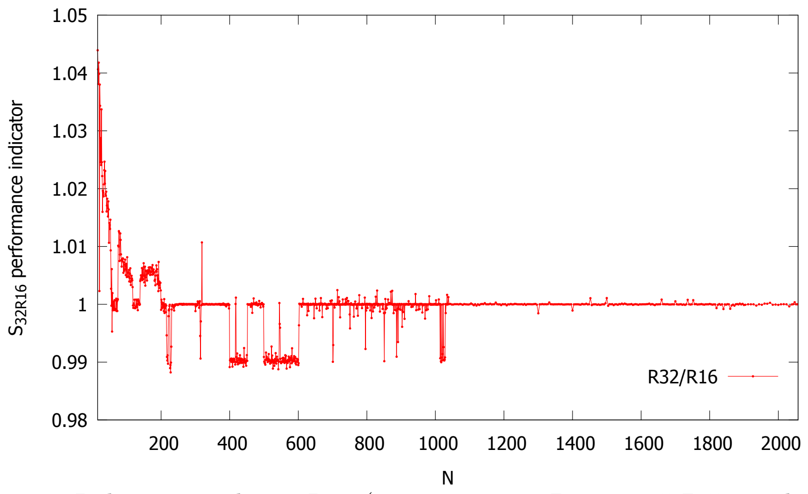

Fig. 5: Relative speedup 32R16 (comparing 32KB over 16KB L1 cache).

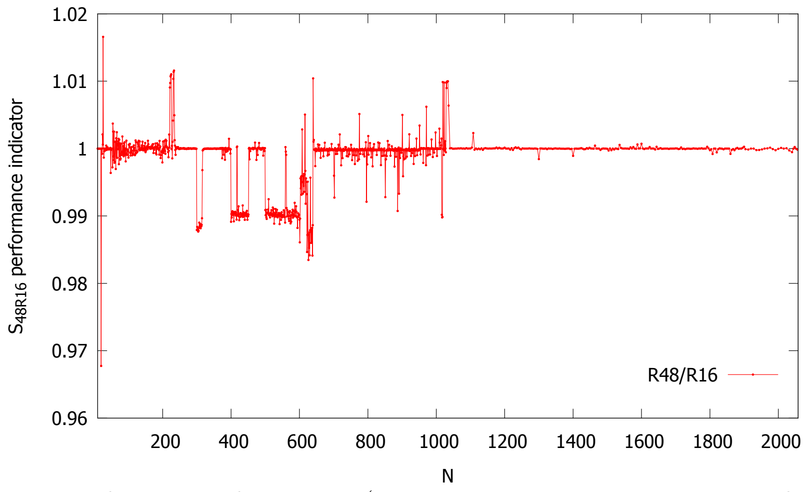

Fig. 6: Relative speedup 48R16 (comparing 48KB over 16KB L1 cache).

The obtained results are discrepant for all experiments. There is no significant influence on the performance for any configuration. However, as we increase the problem size N, the differences between the L1 configurations are decreasing, mainly due to huge number of cache capacity misses. This is especially distinguished in Figure 4, where the differences are almost non existent.

The maximum measured relative performance difference was 1.069045%, and the average relative performance difference among executions on different L1 configurations was 0.134699%.

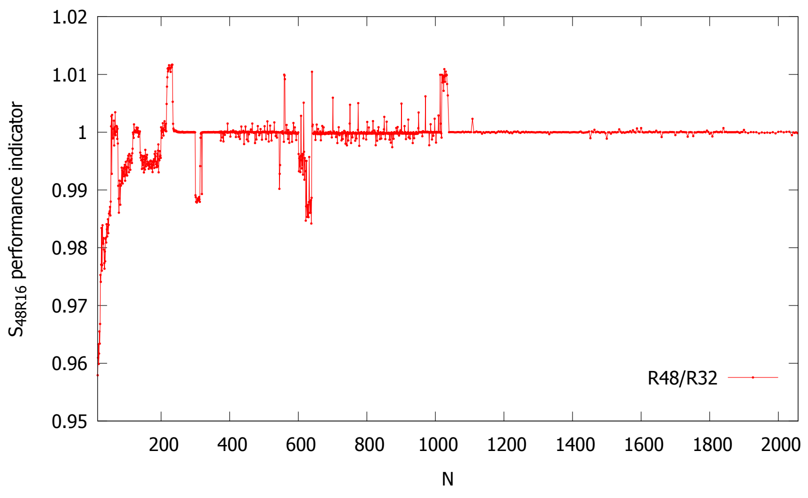

Fig. 7: Relative speedup 48R32 (comparing 48KB over 32KB L1 cache).

6 Conclusion and Future Work

We conducted several experiments and series of test cases for each experiment to determine the optimal L1 cache configuration (cache size and associativity) for cache intensive algorithm, i.e., dense matrix-matrix multiplication. Despite the hypothesis that greater L1 cache memory will achieve better performance, the results show that the relative speeds for 16KB and 48KB are very similar and very close to the speed achieved by 32KB L1 configuration. We conclude that the main factor for the algorithm performance is the L2 cache memory, and the small L1 cache sizes which generate huge cache capacity misses.

However, despite the small L1 cache compared to L2, the speed in L_2 region is greater than the speeds in L_1 region for all three L1 cache sizes. It is also greater than the speed in L_3 region. Another interesting result was speed differentiation in the L_3 region, i.e., in the several regions in the range of matrix size $350 \leq N \leq 650$.

Since the speed is similar for each L1 cache size (and associativity) in all three cache regions (L_1, L_2 and L_3), as well as in the critical matrix size where performance drawbacks appear due to cache set associativity, we propose L1 cache memory to be configured with the cache size (and associativity) that spends less power consumption.

We will continue with our research in order to determine the performance drawbacks due to associativity for different L1 cache configurations, i.e., not only the cache size, but cache line, as well. Another unexpected phenomenon in the performance discrepancy for the points with step 4 that were observed in the L_3 region will be also analyzed and modeled.

References

1. Anchev, N., Gusev, M., Ristov, S., Atanasovski, B.: Optimal cache replacement policy for matrix multiplication. In: Markovski, S., Gusev, M. (eds.) ICT Innovations 2012, Advances in Intelligent Systems and Computing, vol. 207, pp. 71–80. Springer Berlin Heidelberg (2013)
2. Batson, B., Vijaykumar, T.N.: Reactive-associative caches. In: Proceedings of the 2001 International Conference on Parallel Architectures and Compilation Techniques. pp. 49–60. PACT '01, IEEE Computer Society (2001)
3. Calder, B., Grunwald, D., Emer, J.: Predictive sequential associative cache. In: Proceedings of the 2nd IEEE Symposium on High-Performance Computer Architecture. pp. 244–. HPCA '96, IEEE Computer Society (1996), http://dl.acm.org/citation.cfm?id=525424.822662
4. Cheik Ahamed, A.K., Magoules, F.: Iterative methods for sparse linear systems on graphics processing unit. In: High Performance Computing and Communication 2012 IEEE 9th International Conference on Embedded Software and Systems (HPCC-ICESS), 2012 IEEE 14th International Conference on. pp. 836 –842 (june 2012)
5. Collange, S., Defour, D., Tisserand, A.: Power consumption of gpus from a software perspective. In: Computational Science–ICCS 2009, pp. 914–923. Springer (2009)
6. Djinevski, L., Arsenovski, S., Ristov, S., Gusev, M.: Performance drawbacks for matrix multiplication using set associative cache in GPU devices. In: MIPRO, 2013 Proceedings of the 36th International Convention, IEEE Conference Publications. pp. 213–218. Opatija, Croatia (2013)
7. Djinevski, L., Ristov, S., Gusev, M.: Superlinear speedup in gpu devices. In: to be published in ICT Innovations 2012. Springer Berlin / Heidelberg (2012)
8. Grama, A., Karypis, G., Kumar, V., Gupta, A.: Introduction to Parallel Computing (2nd Edition). Addison Wesley, 2nd edn. (Jan 2003)
9. Gusev, M., Ristov, S.: Performance gains and drawbacks using set associative cache. Journal of Next Generation Information Technology (JNIT) 3(3), 87–98 (31 Aug 2012)
10. Hennessy, J.L., Patterson, D.A.: Computer Architecture, Fifth Edition: A Quantitative Approach. Elsevier, MA, USA (2012)
11. Lee, V.W., Kim, C., Chhugani, J., Deisher, M., Kim, D., Nguyen, A.D., Satish, N., Smelyanskiy, M., Chennupaty, S., Hammarlund, P., et al.: Debunking the 100X GPU vs. CPU myth: an evaluation of throughput computing on CPU and GPU. ACM SIGARCH Computer Architecture News 38(3), 451–460 (2010)
12. Likun, Z., Dingfang, C.: Accelerate your graphic program with GPU/CPU cache. In: Proceedings of the 2008 International Conference on Cyberworlds. pp. 667–671. CW '08, IEEE Computer Society (2008)
13. Matsumoto, K., Nakasato, N., Sedukhin, S.: Implementing a code generator for fast matrix multiplication in opencl on the gpu. In: Embedded Multicore Socs (MCSoC), 2012 IEEE 6th International Symposium on. pp. 198 –204 (sept 2012)

14. Meltzer, R., Zeng, C.: Micro-benchmarking the c2070 (Jan 2013), `http://people.seas.harvard.edu/~zeng/microbenchmarking/`
15. Munshi, A., et al.: The opencl specification. Khronos OpenCL Working Group 1, l1–15 (2009)
16. Nickolls, J., Dally, W.: The GPU computing era. IEEE Micro 30(2), 56–69 (2010)
17. NVIDIA: Next generation CUDA compute architecture: Kepler GK110 (2012)
18. NVIDIA: CUDA programming guide (Jan 2013), `http://developer.download.nvidia.com/compute/DevZone/docs/html/C/doc/CUDA_C_Programming_Guide.pdf`
19. Papadopoulou, M., Sadooghi-Alvandi, M., Wong, H.: Micro-benchmarking the GT200 GPU. Computer Group, ECE, University of Toronto, Tech. Rep (2009)
20. Ristov, S., Gusev, M., Djinevski, L., Arsenovski, S.: Performance Impact of Reconfigurable L1 Cache on GPU Devices. In: 2013 Federated Conference on Computer Science and Information Systems FEDCIS'13. p. in press. Krakow, Poland (Sep 2013)
21. Tang, T., Yang, X., Lin, Y.: Cache miss analysis for gpu programs based on stack distance profile. In: Proceedings of the 2011 31st International Conference on Distributed Computing Systems. pp. 623–634. ICDCS '11, IEEE Computer Society (2011)
22. Tarjan, D., Meng, J., Skadron, K.: Increasing memory miss tolerance for simd cores. In: Proceedings of the Conference on High Performance Computing Networking, Storage and Analysis. pp. 22:1–22:11. SC '09, ACM (2009)
23. Volkov, V., Demmel, J.: Benchmarking gpus to tune dense linear algebra. In: Proceedings of the 2008 ACM/IEEE conference on Supercomputing. p. 31. IEEE Press (2008)
24. Williams, S., Oliker, L., Vuduc, R., Shalf, J., Yelick, K., Demmel, J.: Optimization of sparse matrix-vector multiplication on emerging multicore platforms. Parallel Comput. 35(3), 178–194 (Mar 2009)
25. Yoon, H., Zhang, T., Lipasti, M.H.: Sip: Speculative insertion policy for high performance caching. Tech. Rep. 1676, Computer Sciences Department University of Wisconsin-Madison (2010)