Proceedings of the 12th International Conference
on Computational and Mathematical Methods
in Science and Engineering, CMMSE 2012
26–30 June 2012.

Measuring the Impact of Configuration Parameters in CUDA Through Benchmarking

Yuri Torres[1], Arturo Gonzalez-Escribano[1] and Diego R. Llanos[1]

[1] *Department of Computing Science, University of Valladolid*

emails: yuri.torres@infor.uva.es, arturo@infor.uva.es, diego@infor.uva.es

Abstract

The threadblock size and shape choice is one of the most important user decisions when a parallel problem is coded to run in GPU architectures. In fact, threadblock configuration has a significant impact on the global performance of the program. Unfortunately, the programmer has not enough information about the subtle interactions between this choice of parameters and the underlying hardware.

This paper presents uBench, a suite of micro-benchmarks, in order to explore the impact on performance derived from the combination of (1) the threadblock size and shape choice criteria, and (2) the GPU hardware resources and configurations. Each micro-benchmark has been designed as simple as possible to focus on a single effect derived from the hardware or threadblock parameter choice.

As an example of the capabilities of this benchmark suite, this paper shows an experimental evaluation of the Fermi architecture, in terms of configuration parameters. This study confirms some previous experimental results and gives new insights on the influence of these parameters on the performance delivered by this GPU architecture.

Key words: Micro-benchmarks, CUDA, GPGPU, performance measurement.

1 Introduction

Many-core Graphics Processing Units (GPUs) have evolved from graphics-specific accelerators to general-purpose computing devices. The use of high-level parallel languages such as CUDA [1] and OpenCL [2] reduces the programmer's burden in writing parallel applications for GPUs.

The GPGPU [1, 3] community provides general and intuitive guidelines for optimizing application using CUDA model. However, the understanding of important hardware effects

 ISBN:978-84-615-5392-1

and their associated performance in the applications still remains homework for programmers. This is a significant problem in order to maximize the performance of any parallel problem implementation, due to the in-depth knowledge needed about GPU underlying architecture details.

In CUDA, it is always necessary to define the grid of threadblocks. The threadblock shape is a very important decision to develop a high-performance code implementation for NVIDIA GPUs. A correct choice of threadblock size and shape can significantly affect the use of GPU resources. Therefore, the global implementation performance is closely related to the threadblock configuration. Currently, many programmers select the threadblock size and shape (that is, the threadblock configuration) by trial-and-error, spending a significant amount of time.

In our previous work [4, 5] we have stated the importance of a correct threadblock configuration. In this paper we present a suite of micro-benchmarks (uBenchs), in order to explore the impact on performance derived from the combination of (1) the threadblock size and shape choice criteria, and (2) the GPU hardware resources and configurations. Each uBench has been designed as simple as possible to focus on a single effect derived from the hardware or threadblock parameter choice. While the uBench suite has originally designed for Fermi architecture, this suite could also be used with minimal adjustments for the study of any other NVDIA's CUDA architecture such as the recently Kepler [6] release.

uBench's design focuses on the following hardware resource features: The number of streaming-processor and cores, the number of GPU global-memory banks, L1 cache memory configuration and memory-transaction segment size. The benchmarks can be used to study performance effects, such as bottlenecks in access to GPU global-memory or L1/L2 caches, global-memory bank conflicts, or thrashing on L1 cache memory. All of these effects are related with the threadblock configuration.

Our experimental results using uBench with the Fermi architecture show that underlying hardware details can be used to isolate the performance behavior of different threadblock configurations, in terms of the global memory access pattern of the application.

The rest of this paper is organized as follows. Section 2 describes the NVIDIA Fermi architecture, in order to understand how uBench was designed. Section 3 shows some related work. Section 4 introduces a subset of the uBench benchmark suite. Section 5 uses uBench to show some interesting effects derived from the choice of threadblock different configurations and their relationship with the underlying hardware. Finally, Sect. 6 shows our conclusions.

2 NVIDIA Fermi architecture

Fermi [7, 8, 9] is one of the NVIDIA's latest generation of CUDA architecture, released in early 2010. The main features introduced by this architecture include double precision

 ISBN:978-84-615-5392-1

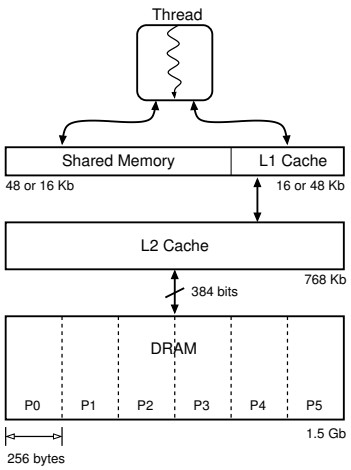

Figure 1: FERMI memory hierarchy (NVidia GTX-480).

performance, error correction code support, transparent L1/L2 cache hierarchy, configurable L1 and shared memory, faster Context Switching, and faster atomic operations. This section shows some hardware details of the Fermi architecture to better understand the hardware effects depending on the threadblock configuration.

Transparent L1/L2 cache memory Fermi introduces an L1/L2 transparent cache memory hierarchy (see Fig. 1). The programmer can choose between two configurations: 48 KB of shared memory and 16 KB of L1 cache (default option), or 16 KB of shared memory and 48 KB of L1 cache. Besides this, the L1 cache memory can be deactivated by the user at compilation time.

Threadblocks, Warps and SMs The number of registers per multiprocessor in Fermi is 32KB, and the number of SP (Streaming Processor) per SM (Streaming Multiprocessor) is 32. The maximum number of threads per threadblock is doubled, from 512 to 1 024, while the maximum number of threads per SM is 1 536. Note that these changes force the programmer to re-calculate block parameters used in implementations for previous architectures, in order to maximize the use of SM resources. The threadblocks are dispatched to the SMs in row-major order [10].

Warp schedulers Regarding the dual warp scheduler present in Fermi, each scheduler has its own instruction dispatch unit (IDU). The SM executes simultaneously two halves of different warps. It is important to notice that this may influence the span of data that is requested to the cache hierarchy at the same time. Although the SMs have only 16

 ISBN:978-84-615-5392-1

load/store units, each half warp issues its memory requests on a different flank, allowing them to issue up to 32 memory requests on the same cycle, 16 from each half-warp.

Shared memory conflicts and global memory access Fermi has 32 shared memory banks. Currently, the global memory is also distributed on five or six banks in Fermi, with a bank width (called "word size" in CUDA terminology) of 256 bytes. The memory addresses are scattered across the banks. A frequent problem in pre-Fermi architectures is the *partition camping* problem [10]. This problem arises when concurrent threads request at the same time memory locations belonging to different transaction segments in the same data bank. In Fermi, the problem is alleviated (although not completely solved [**?**]) by the existence L2 cache.

3 Related work

The use of benchmarks to evaluate hardware configurations has a long tradition. In this paper we focus on benchmarking of GPU devices.

In [11], a set of micro-benchmarks is presented in order to obtain architectural features and basic program characteristics. These features and characteristics include vectorization, burst write latency, texture fetch latency, global read and write latency, ALU/Fetch operation. However the studied focuses on AMD GPU architecture.

The authors in [12] introduce a suite of micro-benchmarks to measure the performance of the GPU as well as the performance change when a specific optimization strategy is used. The authors measure the execution time and obtain the different latencies for the same threadblock configuration. The study focuses on pre-Fermi architectures.

In [13], a suite of micro-benchmarks are presented in order to find, for a given NVIDIA pre-Fermi architecture, the lower and upper bounds of the partition camping problem [10]. They present the global memory read and write operation with and without partition camping. As a result, the authors provide a spreadsheet that calculates an estimation of the partition camping problem for a given kernel. It takes into account the number of active warps and how many data are read by a single thread. They say that this problem does not exists in Fermi architecture and do not consider the full range of threadblock size and shape choices.

In [14], the authors introduce a performance model for pre-Fermi GPUs. This model is based on the results of a set of uBenchs in order to measure the time of each kind of instruction, and the global/shared memory accesses. The authors always use the same threadblock shapes (square) and show the different memory data transfer bandwidth extrapolated from the uBenchs.

In [15], performance analytic model for pre-Fermi architectures is introduced. The authors identify two main parameters: The time that a warp is waiting for data, and the

 ISBN:978-84-615-5392-1

number of operations that could be done during these delays. The authors estimate the values by a set of micro-benchmarks. Different threadblock shapes have not been considered. The paper studies the ocurrence of memory bottlenecks under two main situations: (1) when the computational cost is greater than the cost of memory requests, and (2) the opposite situation.

In summary, these works do not systematically explore all the threadblock configuration space. More over, several of these tests have been conducted using pre-Fermi GPU architectures and do not relate the threadblock configuration with the underlying hardware effects.

4 The uBench suite

This section shows the design principles of the different uBenchs implemented for this study. The benchmarks implemented are not constrained to a specific threadblock configuration. Thus, we can explore the different effects of the uBenchs across the whole threadblock configuration space.

4.1 uBench design principles

Each uBench performs a matrix operation. Some benchmarks uses the same matrix as input/output parameter, while others receives a matrix as input and returns a different matrix as output.

Data sizes and storage order The sizes of the matrices have been chosen taking into account the number and word-size of global-memory banks in Fermi, to produce perfect data alignments in memory banks, and allow the benchmarks to better control the alignment choices in read operations. The matrix size is chosen to obtain at least one block per SM, and to comply with the several thread number requirements in Fermi, for any possible threadblock shape configuration. Matrices are stored in row-major order. The number of the threads in the whole computation is equal to the number of matrix elements. All benchmarks work with integer elements for not considering multiply-add (FMA) double precision instruction and intra-SP dependences. Therefore, each thread writes in a different output matrix element. Finally, we chose sizes with different ratios of total number of threads in the kernel grid vs. the maximum number of concurrent threads that can be scheduled across the whole GPU at the same time: 1536 by the number of SMs in a Fermi chipset (between 14 and 16). The exact matrix sizes chosen are $N = M = 96, 192, 6\,144$ and $18\,432$. Thus, in a Fermi board with 15 SMs per chipset, the ratios are $r = 0.4, 1.6, 1638.4, 14745.6$ (below 0.5, more than 1.5, near the maximum number of threads per SM, and near the maximum size of input matrix that fits in the global-memory of a Fermi board). The matrix sizes

 ISBN:978-84-615-5392-1

should be adjusted to the global-memory bank properties, number of SMs, etc. in other GPU hardware releases.

Use of L1 cache Different uBenchs are designed to use differently the L1 cache. Thus, we can test the performance behaviors related to the configuration of the L1 cache (enabled, disabled and increased).

Data alignment Out-of-bound accesses should not be allowed. To avoid this, we can (a) add data padding, (b) add divergent branches and (c) adjust the threadblock shape to the sizes chosen for the matrices being processed. We have chosen the last option, because the former two may show performance irregularities depending on the particular blocks. Since each thread writes in a different output matrix position, the number of threads per block chosen to launch a uBench should be related to the constraints forced in the matrix sizes. Due to this restrictions and due to the maximum number of thread per threadblock supported by Fermi architecture (1024 threads) the threadblock shapes should fulfill with the following criteria: (1) (#rows and #columns) ≥ 1 and ≤ 1024; (2) (#rows $\times$ #columns) ≤ 1024; and (3) (#rows and #columns) should be multiple of two and three, because these are the only combinations that are exact divisors of the matrices sizes chosen. Recall that these sizes were also chosen to be aligned with the memory banks.

Access patterns Each uBench uses its own memory access pattern for the input matrix. Regarding writes to the output array, uBenchs uses one out of three different correlations types between the thread identification and the items of the output array that the thread should write.

- In the first one, that we call *A mode*, the global coordinates (one for each dimension) of each thread of the Grid are used directly to determine the item to be accessed by the thread. The indexes of $Matrix[\underline{row}][\underline{column}]$ are calculated as follows:

$$\begin{aligned} \underline{row} &\leftarrow blockIdx.y \times blockDim.y + threadIdx.y \\ \underline{column} &\leftarrow blockIdx.x \times blockDim.x + threadIdx.x \end{aligned}$$

- The second one, that we call *B mode*, each thread is assigned to a single uni-dimensional coordinate. Thus, the threadblock shape does not influence the accessed positions (all threadblocks with the same size access to the same data positions. The indexes of $Vector[\underline{resultPos}]$ are calculated as follows:

$$\begin{aligned} \underline{blockGlobalId} &\leftarrow blockIdx.y \times gridDim.x + blockIdx.x \\ \underline{resultPos} &\leftarrow (blockGlobalId \times blockSize) \\ &\quad +(threadIdx.y \times blockDim.x + threadIdx.x) \end{aligned}$$

 ISBN:978-84-615-5392-1

- A third model is used (*C mode*), where all threads that verifies the condition $threadIdx.x = threadIdx.y = 0$ access the output matrix as in A mode. The remaining threads do not perform any access.
- Finally, a fourth model (*D mode*) considers the case in which the GPU global memory is not accessed by any thread as input or output.

4.2 uBench classification criteria

The different uBenchs can be classified according to the following criteria:

1. The type of global memory access pattern (A, B, C or D)
2. High/low ratio of arithmetic instruction per thread compared to the number of global memory access (high, low or none).
3. High/low ratio of L1 cache memory lines evictions compared to the size of this memory (thrashing effects).
4. High/low ratio of global memory data reutilization compared to the number of arithmetic instruction per thread (L1 cache memory).

4.3 uBench suite description

The combination of all the criteria described above leads to 48 different benchmark classes. We have used all of them to explore the configuration space of a Fermi architecture. Due to space constraints, in this paper we will show the effects in terms of performance related to a specific subset of the uBench suite. The benchmarks used are the following:

uBench-1 *Input*: A flattened matrix. *Output*: The same flattened matrix. *Description*: Each thread copies the same constant value in its position. The accesses are perfectly coalesced. No data reutilization.

uBench-3 *Input*: A vector. *Output*: The same vector. *Description*: Each thread copies the same constant value in its position. The accesses are perfectly coalesced. No data reutilization.

uBench-5 *Input*: A vector. *Output*: The same vector. *Description*: Each thread copies the same constant value in its position. Each thread adds the values of its block on the input matrix. The accesses are perfectly coalesced with data reutilization.

 ISBN:978-84-615-5392-1

Benchmark	(1) Global memory access pattern	(2) Load ratio	(3) L1 eviction ratio	(4) Data reutilization ratio
uBench-1	A mode	Low	Low	Low
uBench-3	B mode	Low	Low	Low
uBench-5	B mode	Low	High	High
uBench-6	D mode	None	Low	Low
uBench-7	A mode	Low	Low	High
uBench-9	C mode	Low	Low	Low
uBench-10	A mode	High	Low	Low
uBench-11	B mode	High	Low	Low

Table 1: uBench classification according to the criteria proposed.

uBench-6 *Input*: A flattened matrix. *Output*: The same flattened matrix. *Description*: Without any kernel instruction. Without features.

uBench-7 *Input*: A flattened matrix. *Output*: The same a flattened matrix. *Description*: Each thread copies the sum of its threadblock row values in its position. The accesses are coalesced with data reutilization.

uBench-9 *Input*: A flattened matrix. *Output*: The same a flattened matrix. *Description*: Only the thread with $theadId.x = 0$ and $theadId.y = 0$ stores in its global matrix position the same constant value. There is not coalescence. No data reutilization.

uBench-10 *Input*: A flattened matrix. *Output*: The same flattened matrix. *Description*: Each thread copies the same constant value in its position. We have introduced a overloaded loop to add some instructions before performing the write access. In this benchmark, the loop is executed one thousand times. The accesses are perfectly coalesced. No data reutilization.

uBench-11 *Input*: A vector. *Output*: The same vector. *Description*: Each thread copies the same constant value in its position. We have introduced a overloaded loop to add some instructions before performing the write access. In this benchmark, the loop is executed one thousand times. The accesses are perfectly coalesced. No data reutilization.

Finally, Table 1 shows how each benchmark fulfills the criteria described in Sect. 4.2.

5 Experimental evaluation

We have explored all the combinations of sizes for each threadblock dimension (1, 2, 3, 4, 6, 8, 12, 16, 24, 32, 48, 64, 96, 128, 192, 256, 384, 512, 768, 1 024), L1 configurations (enabled,

 ISBN:978-84-615-5392-1

disabled, increased), and input data sizes (96, 192, 6 144, 18 432) for all the benchmarks described, in terms of execution time.

The experiments have been run on an Nvidia GeForce GTX 480 device. The host machine is an Intel(R) Core(TM) i7 CPU 960 3.20GHz, 64 bits compatible, with a global memory of 6 GB DDR3. It runs an UBUNTU desktop 10.10 (64 bits) operative system. The uBenchs have been developed using CUDA 4.0 toolkit and the 270.41.19 64-bit driver.

The resulting 108 tables can not be reproduced here in any format. Instead, we will briefly discuss some interesting effects observed.

Near-to-optimum threadblock size In [**?**] we showed that, for some specific applications (with perfectly coalesced access patterns, no data reutilization, and low ratio of operation per thread), using a threadblock size of 192 delivered near-to-optimum performance. uBench benchmarks 1, 3, and 7 confirm this effect for any application with these characteristics, leading to a performance loss lower than 5 percent in all cases. It is worthwile to note that uBench 3 has been specifically designed to isolate this specific effect.

In B-mode there is no correlation between threadblock shape and performance The use of B-mode makes that the access pattern for a given threadblock total size does not change when using any possible shape configuration. uBenchs 3, 5, and 6 show that, given a threadblock size, the best performance is obtained with any possible shape.

Each block should access to a single memory bank, and contiguous blocks to different banks The authors of [10] show that, for coalesced access patterns, performance increases when all global-memory accesses from the same threadblock are directed to the same global memory bank, and contiguous blocks direct their accesses to different banks. It is known that the scheduler tends to schedule contiguous blocks in different SMs. Thus, during the life of the kernel, all banks are used in a balanced way from different SMs, thus improving the data bandwidth. The uBenchs 1 and 2 confirm this effect.

It is better to disconnect L1 when input data is scarce Deactivating the L1 cache memory (forcing the hardware to reduce the transactional segment size to 32 Bytes) improves the execution time in certain kind of kernels. When the global memory accesses are not coalesced (uBench 9) reducing the size of the data transaction segment, also reduces the total memory requested, alleviating the memory bottlenecks. On the other hand, our uBenchs show that performance increases, even for coalesced access patterns (all uBenchs with mode A and mode B), if there is a low ratio between the number of threads in the grid and the maximum concurrent threads supported by the target GPU device ($r = 0.4$ and $r = 1.5$).

 ISBN:978-84-615-5392-1

Shapes, sizes and instruction overload We said in [**?**] that, in kernels with a high ratio of arithmetic operations per thread, for a given threadblock size, any block shape leads to practically the same performance. The latencies of the global memory accesses can be hidden inserting enough arithmetic operations between memory accesses. Thus, the execution time is not dominated by memory access latencies, and access patterns derived from the threadblock shape (uBench 1).

Data reutilization and L1 cache size When there is data reutilization by different threads in the same threadblock, better performance is obtained increasing the L1 cache memory. Increasing the cache memory more cache lines (global memory transaction segments) can be stored in the cache reducing thrashing effects. This effect is confirmed by uBenchs 5 and 7. Nevertheless, uBenchs 1 and 6 show that for kernels without any data reutilization, increasing the L1 cache size degrades the performance.

6 Conclusions

We have designed and implemented a suite of micro-benchmarks, called uBench, in order to identify the GPU hardware effects on performance for GPU architecture, depending on threadblock size and shape. With minimal adjustments, the uBench suite can be used to evaluate newer NVIDIA architectures, such as Kepler. After examining the results, we can conclude that the knowledge of the underlying hardware details can sometimes be used to predict the performance behavior, depending on threadblock configuration. In the future, we will use uBench to study the influence of hardware effects and threadblock configuration parameters in other NVIDIA architectures. The uBench benchmark suite is available under request.

Acknowledgements

This research is partly supported by the Ministerio de Industria, Spain (CENIT OCEAN-LIDER), MINECO (Spain) and the European Union FEDER (TIN2011-25639, CAPAP-H3 network TIN2010-12011-E), and the HPC-EUROPA2 project (project number: 228398) with the support of the European Commission - Capacities Area - Research Infrastructures Initiative.

References

[1] D. B. Kirk and W. W. Hwu, *Programming Massively Parallel Processors: A Hands-on Approach.* Morgan Kaufmann, Feb. 2010.

 ISBN:978-84-615-5392-1

[2] KHRONOS, "OpenCL home page," Last visit: March 2, 2011, http://www.khronos.org/opencl.

[3] NVIDIA, "Tuning CUDA Applications for Fermi," 2010, http://developer.download.nvidia.com/compute/cuda/3_1/toolkit/docs/NVIDIA_FermiTuningGuide.pdf, Last visit: Jan 2012.

[4] Y. Torres, A. Gonzalez-Escribano, and D. R. Llanos, "Understanding the impact of CUDA tuning techniques for Fermi," in *Intl. Conf. on High Performance Computing and Simulation, HPCS 2011*, 2011, pp. 631–639.

[5] ——, "Using Fermi architecture knowledge to speed up CUDA and OpenCL programs," in *Proc. ISPA '12*, Leganes, Madrid, Spain, 2012.

[6] NVIDIA, " NVIDIA GeForce GTX 680," 2012, http://www.nvidia.es/content/PDF/productspecifications/GeForce_GTX_680_Whitepaper_FINAL.pdf, Last visit: May 2012.

[7] ——, "Fermi Architecture Home Page," Last visit: August 2, 2010, http://www.nvidia.com/object/fermi_architecture.html.

[8] ——, "Nvidia cuda programming guide 3.0 fermi," 2010.

[9] ——, "Whitepaper: NVIDIA's next generation CUDA compute architecture: Fermi," 2010, http://www.nvidia.com/object/fermi_architecture.html,Last visit: Nov, 2010.

[10] P. M. Greg Ruetsch, "NVIDIA optimizing matrix transpose in CUDA," http://developer.download.nvidia.com/compute/cuda/3_0/sdk/website/CUDA/ website/C/src/transposeNew/doc/MatrixTranspose.pdf, Jun. 2010, Last visit: Dec 2, 2010.

[11] R. Taylor and X. Li, "A micro-benchmark suite for amd gpus," in *Parallel Processing Workshops (ICPPW), 2010 39th International Conference on*, sept. 2010, pp. 387 –396.

[12] H. Wong, M.-M. Papadopoulou, M. Sadooghi-Alvandi, and A. Moshovos, "Demystifying GPU microarchitecture through microbenchmarking," in *Performance Analysis of Systems Software (ISPASS), 2010 IEEE International Symposium on*, march 2010, pp. 235 –246.

[13] A. M. Aji, M. Daga, and W.-c. Feng, "Bounding the effect of partition camping in GPU kernels," in *Proceedings of the 8th ACM International Conference on Computing Frontiers*, ser. CF '11. New York, NY, USA: ACM, 2011, pp. 27:1–27:10. [Online]. Available: http://doi.acm.org/10.1145/2016604.2016637

 ISBN:978-84-615-5392-1

[14] Y. Zhang and J. Owens, "A quantitative performance analysis model for gpu architectures," in *High Performance Computer Architecture (HPCA), 2011 IEEE 17th International Symposium on*, feb. 2011, pp. 382 –393.

[15] S. Hong and H. Kim, "An analytical model for a gpu architecture with memory-level and thread-level parallelism awareness," in *Proceedings of the 36th International Symposium on Computer Architecture, ISCA '09*. New York, NY, USA: ACM, 2009, pp. 152–163. [Online]. Available: http://doi.acm.org/10.1145/1555754.1555775

 ISBN:978-84-615-5392-1