Proceedings of the First PhD Symposium on Sustainable Ultrascale Computing Systems (NESUS PhD 2016)
Timisoara, Romania

Jesus Carretero, Javier Garcia Blas
Dana Petcu
(Editors)

February 8-11, 2016

Techniques for Autotuning Algorithms on Heterogenous Platforms

Adrián P. Diéguez, Margarita Amor, Ramón Doallo

University of A Coruña, Spain
{adrian.perez.dieguez,margarita.amor,ramon.doallo}@udc.es

Abstract

*Current GPUs (*Graphic Processing Units*) can obtain high computational performance in scientific applications. Nevertheless, programmers have to use suitable parallel algorithms for these architectures and have to consider optimization techniques in the implementation in order to achieve that performance. This thesis is focused on designing and implementing parallel prefix algorithms into GPU architectures with little effort. For that, we have developed a very optimized library called BPLG (Tuning Butterfly Processing Library for GPUs) and based on a set of building blocks that enable to easily design well-known algorithms such as FFT, tridiagonal systems solvers, scan operator, sorting or signal processing. This library is designed under a tuning methodology based on two-stages indentified as GPU resource analysis and operator string manipulation. Specifically, this strategy is focused on a set of parallel prefix algorithms that can be represented according to a set of common permutations of the digits of each of its element indices [4], denoted as Index-Digit (ID) algorithms. So far, the proposed methodology has obtained very good results with respect to state-of-art libraries, as CUFFT, CUSPARSE, CUDPP or ModernGPU.*

Keywords CUDA, parallel prefix algorithms, GPU, ID-algorithms, tuning

I. Motivation

In recent years, GPUs (Graphics Processing Units) have experienced a noticeable increase in its relevance and usage in high performance computing. Nevertheless, programmers have to use suitable parallel algorithms for these architectures that also require special languages such as NVIDIA CUDA or OpenCL; and finally, have to consider optimization techniques in the implementation in order to achieve high performance.

The algorithms examined in this thesis are described using a parallel prefix approach [17], one of the most popular parallel paradigms. Some parallel prefix algorithms may be also represented according to a set of common permutations of the digits of each element index [4], denoted as Index-Digit (ID) algorithms. In this thesis, we have focused on the following ID-algorithms: FFT, Tridiagonal Systems Solvers, Scan Operator and Sorting algorithms.

The *FFT* is a highly important operation for many applications, such as image and digital signal processing, filtering, compression or partial differential equation resolution. Tridiagonal linear systems arise in many scientific and engineering problems such as fluid dynamics, heat conduction, numerical analysis, ocean models or cubic spline approximations. The scan operator is widely used in areas such as the construction of summed area tables, stream compaction, image filtering, or cryptography, among many others. Sorting is a computational building block of high importance, being one of the most studied algorithms due to its impact. Many algorithms rely on the efficiency of sorting routines. For example, computer graphics, and geographic information systems or MapReduce patterns.

Thus, it is relevant the importance of efficently solving these algorithms. For that, GPUs provides an excellent hardware desing where executing these parallel algorithms. For achieving this goal, there are several proposals in order to facilitate the programmability of these architectures: automatic parallelization, directive-based compiler approaches and auto-tuning frameworks or libraries.

Automatic parallelization and performance optimization of affine loop nests on GPU is developed using a polyhedral compiler model of data dependence abstraction and program transformation. In [2], a compiler algorithm revises data placement across different types of GPU resources using input optimized programs. Shared memory multiplexing [22] allows a higher number of thread blocks to be executed concurrently. GPU caches suffer contention due to massive multithreading, an adaptive cache bypass is presented in [20] in order to reduce contention and preserve space for reused cache lines.

Frameworks using directive-based compiler approaches [19, 18] have been developed to automatically optimize GPU programs. Most of this kind of libraries require to have GPU expertise, specifying the number of threads to be used, which loops are parallelised or when to synchronize. Furthermore, the code is not easily readable, complicating the tuning process, and there are some limitations as programmer cannot use CUDA intrinsic functions within the accelerator region.

Autotuning is a very interesting option for applications whose execution time, memory usage or energy consumption can vary depending on a set of parameters and their execution environment. These parameters can take a small number of values and the autotuner determines the best combination to maximise an user-defined metric. On GPUs, there are various tunable parameters, such as the number of warps per block or the *workload* per thread. Nevertheless, this technique requires writing code in a parametrized way to accommodate various performance tuning parameters. Taking into account previous proposals disadvantages, we have decided to focus my thesis on this approach.

II. Related Work

There are several implementations on GPU for each cited algorithm. Furthermore, there are also some GPU methodologies based on an autotuning approach. All of them are studied in this section.

There are some auto-tuning proposals for FFTs on *GPUs*, achieving high performance, such as [21]. Specifically, approaches focused on large 1D FFT on a single coprocessor is [21]. However, the most used and well-known *GPU* implementation is *NVIDIA's CUFFT* [12]. There are some *GPU* tridiagonal solvers implementations based on different algorithms, such as [23, 10]. There are also *GPU* proposals based on auto-tuning design for tridiagonal solvers in [1]. Most scan implementation on GPU are based on either the Kogge-Stone or the Brent-Kung parallel prefix patterns, being important [8] and [9]. Finally, there are several parallel sorting algorithms which have been developed for GPUs. Radix sort for GPUs was efficiently implemented in [11] and Quicksort algorithm in GPU was implemented in [3].

Most of previous approaches provide a solution focused in just one algorithm; however, there is a growing trend of using acceletared libraries that solve this and other parallel algorithm being devoted to a set of algorithms. Our proposal gives a solution based on the development of a small number of efficient parametrizable skeleton building blocks carefully designed to achieve high level of efficiency in CUDA architecture and thought to be used by a set of parallel prefix algorithms instead of focusing on just one. Other examples are CUSPARSE [16] and CUDPP [14], accelerated libraries developed by *NVIDIA*; Merrill's CUB [15] and ModernGPU [13].

III. Thesis idea

The thesis is focused on developping a 2-stage methodology for implementing efficient parallel prefix algorithms on GPU architectures. In the first stage,

performance parameters are obtained from a GPU performance analysis in order to achieve a set of premises such as the maximum parallelism to keep all elements of the GPU occupied. In the second stage, CUDA kernels are obtained from a combination of two techniques called *index-digit permutations* and *tuned mapping vector*, which are used to adjust the data distribution in the GPU according to the resource analysis made at the first stage and the digits of the element's index. Furthermore, our code is designed as building blocks. That means, the functions used are very abstract and they can be reused for the different algorithms. These functions, or building blocks, are parameterized (data types and performing variables are unspecified) and then, the corresponding tuned parameters for each architecture are selected at compile-time and sent them to these functions. So, in the end, thanks to this parametrization of the code, we are designing GPU algorithms with little effort and obtaining very competitive performance with respect to other approaches.

Depending on the size of the problem, we have divided the development of our methodology in three phases:

- The problem data fits in shared memory. Each problem is assigned to a single CUDA block, using the shared memory to perform communications.
- The problem size is bigger than shared memory but can be allocated in the GPU memory of a single GPU. The work is distributed among several blocks, using several kernels for coordinating them.
- The problem size is bigger than the GPU memory of a single GPU, using streams and MPI for dealing with that in a *MultiGPU* approach.

So far, we have implemented FFT, Hartley transform, Discrete cosine transform, different tridiagonal systems solvers, different scan operator algorithms and an algorithmic variant of Bitonic Sort for sorting; obtaining very good results [5, 6, 7] with respect to other state-of-art libraries such as *CUDPP*, *CUSPARSE*, *CUFFT* and *ModernGPU*.

IV. Conclusions

This thesis presents a two-stages methodology for efficiently implementing parallel prefix algorithms into GPU architectures with little effort. Specifically, the strategy is focused on a set of algorithms known as ID-algorithms. In the first stage a *GPU resource analysis* is performed, where performance parameters are obtained from a GPU performance analysis. In the second stage, *operators string manipulation*, kernels are obtained after adjusting the data distribution in the GPU according to the first stage. These kernels are developed with a set of *building blocks* that enable to easily design flexible code, and are integrated in our BPLG library (*Tuning Butterfly Processing Library for GPUs*).

Depending on the problem size, three different strategies have been considered. So far, we have tested this methodology for small and medium problem sizes, outperforming well-known libraries as *CUFFT*, *CUSPARSE*, *CUDPP* and *ModernGPU*.

Acknowledgment

This work is supported by EU under the COST Program Action IC1305: Network for Sustainable Ultrascale Computing (NESUS).

References

[1] A. Davison and J. D. Owens. Register Packing for Cyclic Reduction: A Case Study. In *Proc. of the Fourth Workshop on General Purpose Processing on Graphics Processing Units (GPGPU-4)*, pages 4:1–4:6, 2011.

[2] C. Li, Y. Yang, Z. Lin and H. Zhou. Automatic Data Placement into GPU On-Chip Memory Resources, booktitle = Proceedings of the 13th Annual IEEE/ACM International Symposium on Code Generation and Optimization, CGO'15, year = 2015, pages = 23–33.

[3] Daniel Cederman and Philippas Tsigas. GPU-Quicksort: A Practical Quicksort Algorithm for

Graphics Processors. *J. Exp. Algorithmics*, 14:4:1.4–4:1.24, January 2010.

[4] D. Fraser. Array Permutation by Index-Digit Permutation. *Journal of ACM*, 23(2):298–309, 1976.

[5] Adrián P. Diéguez, Margarita Amor, and Ramón Doallo. Efficient Scan Operator Methods on a GPU. In *Proceedings of the 2014 IEEE 26th International Symposium on Computer Architecture and High Performance Computing*, SBAC-PAD '14, pages 190–197, 2014.

[6] Adrián P. Diéguez, Margarita Amor, and Ramón Doallo. BPLG-BMCS: GPU-sorting algorithm using a tuning skeleton library. *The Journal of Supercomputing*, pages 1–13, 2015.

[7] Adrian P. Diéguez, Margarita Amor, and Ramon Doallo. New Tridiagonal Systems Solvers on GPU architectures. In *Proceedings of IEEE International Conference on High Performance Computing (2015), HiPC'15 (accepted)*, 2015.

[8] D.Merrill and A. Grimshaw. Parallel scan for stream architectures. In *Technical report*. Dept. of Computer Science, Univ. of Virginia, December 2009.

[9] Yuri Dotsenko, Naga K. Govindaraju, Peter-Pike Sloan, Charles Boyd, and John Manferdelli. Fast scan algorithms on graphics processors. In *Proceedings of the 22Nd Annual International Conference on Supercomputing (2008)*, pages 205–213, 2008.

[10] H.-S. Kim, S. Wu, L.-W. Chang, W.W. Hwu. A Scalable Tridiagonal Solver for GPU. In *Int. Conf. on Parallel Processing*, pages 444–453, 2011.

[11] Mark Harris, Shubhabrata Sengupta, and John D Owens. Parallel prefix sum (scan) with CUDA. *GPU Gems*, 3(39):851–876, 2007.

[12] NVIDIA. *CUDA CUFFT Library*, 2012. v5.0.

[13] Nvidia Comp. Modern gpu library, 2013.

[14] Nvidia Comp. CUDPP: CUDA Data Parallel Primitives Library, 2014.

[15] Nvidia Comp. Cub library, 2015.

[16] NVIDIA-Corporation. CUDA CUSPARSE Library. 2012.

[17] R. E. Ladner and M. J. Fischer. Parallel Prefix Computation. *Journal of the ACM*, 27(4):831–838, 1980.

[18] S. Wienke, P. Springer, C. Terboven, D. an Mey. OpenACC: First Experiences with Real-world Applications. In *Proceedings of the 18th International Conference on Parallel Processing, EuroPar12*, pages 859–870, 2012.

[19] T. Han and T. Abdelrahman. hiCUDA: A High-level Directive-based Language for GPU Programming. In *GPGPU-2: Proceedings of 2nd Workshop on General Purpose Processing on Graphics Processing Units*, pages 52–61, 2009.

[20] X. Chen, S. Wu, L.-W. Chang, W.-S. Huang, C. Pearson, Z. Wang and W.-M. W. Hwu. Adaptive Cache Bypass and Insertion for Many-core Accelerators. In *Proceedings of International Workshop on Manycore Embedded Systems, MES'14*, pages 1:1–1:8, 2014.

[21] Y. Dotsenko, S.S. Baghsorkhi, B. Lloyd and N.K. Govindaraju. Auto-Tuning of Fast Fourier Transform on Graphics Processors. In *Proceedings of Principles and Practice of Parallel Programming (PPoPP '11)*, pages 257–266, 2011.

[22] Y. Yang, P. Xiang, M. Mantor, N. Rubin and H. Zhou. Shared memory multiplexing: A novel way to improve gpgpu throughput. In *Proceedings of the 21st International Conference on Parallel Architectures and Compilation Techniques*, PACT '12, pages 283–292. ACM, 2012.

[23] Y. Zhang, J. Cohen, J.D. Owens. Fast Tridiagonal Solvers on the GPU. In *Proceedings of the 15th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming (PPoPP 2010)*, pages 127–136, 2010.