RACER data stream based array processor and algorithm implementation methods as well as their applications for parallel, heterogeneous computing architectures

Theses of the Ph.D. Dissertation

Ádám Rák

Scientific adviser:
György Cserey, Ph.D.

Pázmány Péter Catholic University
Faculty of Information Technology and Bionics

Budapest, 2014

1 Introduction and aim

In recent years a new direction has started in the world of computing, which is based on increasing the number of cores and execution units rather than the clock frequency of the processors. This trend is manifested in all of the network devices, desktop computers and even in cell phones. The main reason for this can be traced back to physical laws, as the miniaturization of microchips and the increase of the clock frequency led to a much too long communication time between the remote parts of the processor. This delay is caused mostly by wiring and metal connections of the chip. The further increase of the clock frequency is therefore not only impeded by a limit determined by the silicon's switching speed, but it also increases the experienced value of the delay. Too much delay implicates more fragmentation of the architecture into several execution units and cores.

According to Moore's law, the manufacturing cost of digital integrated electronics per transistor is becoming cheaper. This will help the above mentioned direction further, as in a well-designed multiprocessor system, the increase of the number of cores is a simple task. This way not only more and more transistors, but more and more cores (or, raw computing power, increasing at the same rate as defined in Moore's law) are gained for the same price.

This trend is lead by graphics processing unit (GPU), which is achieved and even exceeded the number of 5760 cores per microchip in 2014. These multi-core or many-core systems are DSP, FPGA, CELL and GPU, but this trend encompasses the

embedded, multimedia processors too.

Besides the rapid development of the hardware, the question arises how these architectures can be programmed efficiently. Many-core processor systems show not only more variety than traditional predecessors, but require fundamentally new programming approach. In order to integrate as many cores as possible in a processor unit, the computational units were simplified as much as possible. Practically most of the results of the last twenty years had been thrown away from single processor core optimization. Which focused on a single processor core optimization. Thus the difference between a simple computational unit, e.g. Floating-Point Unit, and a core with full functionality is not clear. The functional differences between many-core and traditional processors are illustrated by that if a strict serial program has been executed on a many-core processor, then the running time is often 100 times slower than running it on a non-parallel CPU. The standard OpenCL programming language has been created to program such a new parallel systems. OpenCL is a low level C language, which pushes off the problem of parallelization of the algorithms to the programmers.

The efficient implementation of an algorithm requires the deep knowledge of the target architecture. Based on experiences, this knowledge is necessary even while using OpenCL language because the smallest optimization solutions can be speed up the program by a few orders of magnitude. The problem is even more complicated, because the manufacturer (NVIDIA, AMD-ATI) changes its architecture in every year and fundamentally redesigns it in every two years. Moreover it is common that the manufacturer has difficulties to understand its own product

and exploit its advantages.

There is significant demand to have solutions that can automate the parallel implementation of algorithms with mathematically backed methods. This includes those methods too, where implementing algorithms efficiently on a new architecture is assisted by machine learning.

Considering these problems, my aim was to analyze the parallelization of general algorithm classes and demonstrate my results and methods on a few difficult algorithms.

The trend is obvious, the number of cores per processor will increase exponentially in the next five-ten years. However, the difference between each, following architectures is not only the number of processors, but the changes of architectures. This evolution leads not only to higher number of processing units, but to the more efficient and optimized operation and also to the increased computational power per area. If we examine the parallel architectures, we find that the objective is to maximize the general purpose computing power per unit area by employing trade-offs. These trade-offs and disadvantages at the most common architectures are the following:

The difficulties of memory reading and writing of CPUs are hidden by using traditional cache hierarchy. This solution, especially if we have more processor units, increases untenable the ratio between chip area of cache memory and chip area of pure computing. It is a good balance for the less computationally intensive tasks, but quite wasteful in case of scientific or graphical computations.

DSP: digital signal processor. These devices are very simi-

lar to CPUs, the difference is mainly between their parameters. DSPs are designed for running signal processing algorithms efficiently (FFT, matrix-vector operations) with low power consumption and competitive price. The chip area (ie. the cost of manufacturing) is much smaller than CPUs', because of the above reasons, DSPs has less cache memory. Therefore the system memory access patterns of DSPs is more restricted if we want to exploit the available bandwidth.

The vector based SIMD (single instruction multiply data) architecture of GPUs (graphical processing unit) introduces a very strong constraint on the implementation of threads. In a workgroup every thread has to do the same operation on different data, reading the data from adjacent memory. Therefore both the memory bandwidth and computing resource utilization of the silicon area are very high. But working with this architecture the programmer has to solve the efficient use of memory, contrary to the CPU, this system does not hide the architecture details and does not solve the related problems.

Cell BE (cell broadband engine): this is a hybrid architecture, which includes a classic PowerPC CPU processor connected to SPUs (synergistic processing units). The SPUs are very simplified vector processing units, which have relatively large local memory on chip. The programmers are responsible to solve even every tiny technical problems, from the appropriate feeding of the pipeline to organize the internal logic of the memory operations. This device has only indirect memory access via the local memory.

FPGA (field programmable gate array): On this architecture, arbitrary logic circuit can be implemented within certain

broad limits. Usually the implemented circuit is relatively efficient, since the desired circuit is realized physically on the FPGA by connecting on-chip switches. Consequently the logic circuits of the FPGA can be adapted directly to the given task, therefore this architecture can exploit most efficiently the available processing units. However the cost of this enormous flexibility is the low density of the processing units on the chip surface, since the switching circuits and universal wiring need large chip area.

Systolic Array: this classical topological array processor architecture contains effectively only execution (computing) units, adder and multiplier circuits, which are usually solve some linear algebra operations in parallel. Its applicability is very limited, because its topology is specific for the executed algorithm. This architecture does not contain neither memory architecture, nor program control structure. These units should be provided by another system. The flexibility is sacrificed for efficiency, since the computing units utilized almost 100 percentage during operation and the surface of the silicon chip contains effectively only computing units.

CNN (cellular nonlinear/neural networks): this architecture is efficient at using local image processing operations (low resolution image processing algorithms on grayscale images) with extremely high speed and low power consumption. Every pixel is associated to a processing unit, the process is analog and there is only a very little analog memory. Accessing the global memory compared to the internal speed is very slow and also needs the digitalization of the pixels. This architecture is optimized for 2D topological computations with low memory.

Considering these problems, my aim was to design a computational architecture (RACER architecture), which is not limited by the disadvantages of the previous parallel architectures, Turing complete and fully general algorithms can be implemented efficiently on it, moreover its performance per area is maximized as much as possible.

2 Methods used in the experiments

In the course of my work, instruments of numerous disciplines were applied. One of the most important of these is the theory of automatic parallelization. In case of automatic parallelization loop structures, which can be reformulated in a parallel way, are identified in the implemented algorithm. The loop structures are usually described by polyhedral representation, where the static loop structure is represented in a multi-dimensional discrete space and converted to the desired shape by affine geometric transformations. I supplemented this theory to work the dynamic control structures too. For the representation I used data-flow and control-flow description of the program, which enable the efficient automatic management and optimization of the code. I have learned the internal operation of the two currently most popular open source compilers (GCC, LLVM) and the optimization algorithms they use.

It was necessary to study in detail the following most widely used general-purpose programmable GPU architectures:

- NVIDIA GeForce8

- NVIDIA Fermi
- NVIDIA Kepler
- AMD(ATI) R800(Evergreen)
- AMD(ATI) R900(NI Cayman)
- AMD(ATI) R1000(Southern Islands GCN)

For the AMD architectures, the manufacturer provided to me the documentation of the machine code, the detailed structure of the architectures and also the general-purpose hardware-level programming. In the case of NVIDIA, the architecture dependent information was collected by disassembly and careful measurements.

While designing the RACER architecture, I used general engineering design methods of digital processors, such as pipeline design, digital synthesis, H-fractal clock routing method, resonant network and asynchronous network. During the construction, I tried to use as much existing IP-core modules as possible. I combined my experience of implementing complex algorithms on GPU (e.g. video compression, sparse-matrix algebra) with the local data processing methods of array processors and systolic arrays. I got acquainted with the operation of the processor memory communication and the limitation of memory circuits. The elimination of these limitations plays an important role in the RACER architecture. I have learned efficient simulation methods of digital circuits and their high-level design in VHDL and Verilog languages.

My results are in use in a GPU optimized quantum chemistry software computing the two electron integrals. I have implemented a specialized compiler software for achieve the massive parallelism and GPU optimized program code. This compiler is able to expand the integrals symbolically, in this reduction the following algorithms were taken as a basis:

- Boys
- Pople-Hehre
- Obara-Saika-Schlegel
- Head-Gordon-Pople
- McMurchie-Davidson
- MD-PRISM, HGP-PRISM

While understanding these algorithms, I became acquainted with the details of mathematical methods of numerical quantum chemistry, in particular with the computation of two electron integrals of Gaussian basis and the mathematical method of the general bra-ket. I have learned the methods which use the integrals for calculating the electrostatic potentials:

- Self Consistent Field : SCF-HF
- Density Function Theory : DFT-KS
- Moller-Plesset Perturbation Theory : MPPT

- Configuration Integration : CI
- Coupled Cluster computation : CC

3 New scientific results

1. Thesis: *I showed that in case of programming many-core architectures, besides classical static polyhedral representation, dynamic polyhedral loops and dynamic control structures can be represented by polyhedrons. In the case of dynamical polyhedrons, I showed and gave a formalism how to manage memory access patterns. I defined those algorithm classes, which can be managed efficiently with my proposed methods. [7, 8, 9]*

I proposed a new mathematical formalism for the high-level manipulation of the dynamical control structures of the programs. I reduced the dynamical control structures to infinite static structures with specific dynamical dependences. The infinite limits are necessary, because the dynamic limits are parametric in compilation time therefore they can be overestimated by infinity. Dependencies can be dynamic, which makes necessary to execute the parallelization in runtime. Thus, in this theory, the scheduling is the part of program execution, and this process determines that which computations where and when are executed. The theory is demonstrated by the parts of my H.264 video encoder implementation.

2. Thesis: *I designed a new data stream based parallel computing architecture (RACER), in which the tasks are distributed between the memory and processing units more efficiently than in previous architectures. For this achievement, the parallelism is extended to the memory as well. [14]*

I designed the modules of RACER data stream driven com-

putational architecture. Both the program and data streams pass through the array processor. The program stream forms the appropriate structure which processes the following data stream. The control of the data stream processing can be dynamic too including branches, loops, merges and forks. The connected memory system is also a very important part of the architecture, which contrary to the conventional memory, contains computing elements too, in particular the comparison arithmetic units. Thus, appropriate algorithm dependent ordering of the data can be achieved, which provides continuous data feed of the array processor.

2.1. I showed that due to the simplicity and locality of the control electronics and wiring, in case of the realization of VLSI, the 57-72 percent of the chip area is used by arithmetic processing units. Implementing RACER architecture, one of the highest arithmetic density could be reached compared to available GPU architectures.

I have chosen GRFPU-1 IP core from Aeroflex Gaisler for processing elements. In my estimations I have used 100K gates in one core for 65nm and 90nm technologies. I have estimated the number of gates of the routing elements also, which covers about 20 percent of the surface. From these values I estimated the final chip sizes and the power consumptions with Cadence InCyte Chip Estimator. I have compared GPU peak performances to the estimated RACER peak performances in order to highlight the possible performance gains coming from the higher number of computing cores. By estimation the RACER

architecture's surface covered by computing cores is between 57 and 72 percent of the chip area.

2.2. I showed that Mandelbrot and Conway's Game of Life algorithms can be implemented on the RACER architecture, while RACER remains a general architecture. With the implemented applications I demonstrated the functionality of the architecture and proved that the architecture is Turing complete.

I designed an possible hardware implementation and I proved the viability of the RACER architecture in simulations. I applied low level simulations where the input is the graph representation of the control and data flows. Thus, the proper functioning of the algorithms can be verified on the proposed RACER architecture.

3. Thesis: *I utilized my techniques for accelerating the computation of two-electron integrals in quantum chemistry algorithms in particular to the compatibility of the single-instruction-multiply-data (SIMD) architecture. I designed a meta algorithm (BRUSH) for GPUs, which assigns the optimal computational path for each Gaussian two electron integral. I showed that in the case of special contractions, the constant substitution and propagation is efficient on these architectures. [12, 1]*

I designed and implemented a specialized compiler for computing two-electron integrals of quantum chemistry methods. This compiler allows the efficient exploitation of parallel SIMD architectures. In quantum chemistry, the most important nu-

merical problem is the calculation of two-electron integrals. The input of my compiler is the actual integral problem, which is unfolded in compiling time contrary to the previous methods. All the dynamic control operations are executed during compilation. The optimal computational paths are calculated and chosen beforehand. The hardware specific machine code is generated from the received computational graph which contains a huge number of arithmetic operations. While I designed this transformation I paid special attention to the exploitation of the properties of the architecture. For example, the usage of multi-level memory structure to store temporary values, or the optimization of parallel processing of the SIMD cores.

4 Application of the results

My work and its theoretical results were motivated by practical utilization. The presented algorithms provide solutions for problems in real application domains.

The results of the first thesis group assist compilation of algorithms on many-core architectures (GPU, FPGA).

My second thesis group presents an architecture which has excellent computational performance in many different fields. These applications including but not limited to: 3D graphics rendering, raytracing, computation on unstructured grid, computer games, dealing with large databases and all problems which can be solved efficiently on GPU.

In the third thesis group, an algorithm was presented which can be used for general purpose applications. The GPU acceler-

ation of quantum chemical calculations can assist the synthetic molecule design by significantly reducing the running time.

5 Acknowledgements

I thank Interdisciplinary Technical Sciences Doctoral School of Pázmány Péter Catholic University, Faculty of Information Technology and Bionics, and its senior masters, Professor Dr. Tamás Roska, and Professor Dr. Péter Szolgay for the encouragement and advisement during my studies.

I would like to thank György Cserey, for his support, encouragement, advices and inspiration.

I thank my colleagues for their advices and with whom I could discuss all my ideas: Gergely Balázs Soós, Gergely Feldhoffer,

Ákos Tar, József Veres, Balázs Jákli, Norbert Sárkány, Gábor Tornai, Miklós Koller, István Reguly, Csaba Józsa, András Horváth, Attila Stubendek, Domonkos Gergely, Mihály Radványi, Tamás Fülöp, Tamás Zsedrovits, Csaba Nemes and Gaurav Gandhi.

I thank Vida Tivadarné her patience and devoted work to make the administrative issues much easier, the help of PPCU's dean's office and the help of academic and financial department.

The support of grants Nos. TÁMOP-4.2.1/B-11/2-KMR-2011-0002 and TÁMOP-4.2.2/B-10/1-2010-0014 are gratefully acknowledged.

I am grateful to Anna for providing her friendship and our discussions even her always busy schedule.

I am very grateful to my mother and father and to my whole family who always believed in me and supported me in all possible ways.

6 Publications

6.1 The author's journal publications

[1] **A. Rák** and G. Cserey, "The BRUSH algorithm for two-electron integrals on GPU," *MATCH Communications in Mathematical and in Computer Chemistry*, 2014. submitted.

[2] **A. Rák** and G. Cserey, "Macromodeling of the memristor in SPICE," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 29, no. 4, pp. 632–636, 2010.

[3] **A. Rák**, G. Gandhi, and G. Cserey, "Chua's circuit topology evolution using genetic algorithm," *International Journal of Bifurcation and Chaos*, vol. 20, no. 3, pp. 687–696, 2010.

[4] G. B. Soós, **A. Rák**, J. Veres, and G. Cserey, "GPU boosted CNN simulator library for graphical flow based programmability," *EURASIP Journal on Advances in Signal Processing*, 2009. Article ID 930619, 11 pages doi:10.1155/2009/930619.

[5] **A. Rák**, G. B. Soós, and G. Cserey, "Stochastic bitstream based CNN and its implementation on FPGA," *Interna-*

tional Journal of Circuit Theory and Applications, vol. 37, no. 4, pp. 587–612, 2009.

6.2 The author's international conference publications

[6] G. Cserey, **A. Rák**, B. Jákli, and T. Prodromakis, "Cellular neural networks with memristive cell devices," in *Proceedings of 17th IEEE International Conference on Electronics, Circuits, and Systems, ICECS 2010*, (Athens, Greece), pp. 938–941, Dec. 2010.

[7] **A. Rák**, G. Feldhoffer, G. B. Soós, and G. Cserey, "Standard C++ Compiling to GPU with Lambda Functions," in *Proceedings of 2010 International Symposium on Nonlinear Theory and its Applications (NOLTA 2010)*, (Krakow, Poland), 2010.

[8] **A. Rák**, G. Feldhoffer, G. B. Soós, and G. Cserey, "Standard c++ compiling to GPU," in *3rd HUNGARIAN-SINGAPOREAN WORKSHOP on SYSTEMS BIOLOGY and COMMUNICATION SYSTEMS*, (Budapest, Hungary), 2010.

[9] **A. Rák**, G. Feldhoffer, G. B. Soós, and G. Cserey, "CPU-GPU hybrid compiling for general purpose: Case studies,"

in *Proceedings of 12th International Workshop on Cellular Neural Networks and their Applications, CNNA 2010*, (Berkeley, USA), Feb. 2010.

[10] G. J. Tornai, G. Cserey, and **A. Rák**, "Spatial-temporal level set algorithms on CNN-UM," in *Proceedings of 2008 International Symposium on Nonlinear Theory and its Applications, NOLTA 2008*, (Budapest, Hungary), pp. 696–699, 2008.

[11] G. B. Soós, **A. Rák**, J. Veres, and G. Cserey, "GPU powered CNN simulator (SIMCNN) with graphical flow based programmability," in *Proceedings of 11th International Workshop on Cellular Neural Networks and their Applications, CNNA 2008*, (Santiago de Compostela, Spain), pp. 163–168, 2008. Cited: **2**.

6.3 The author's other publications

[12] **A. Rák**, and Feldhoffer, G., and Soós, G.B. and Höltzl, T., and Oroszi, B. and Cserey, György, "Eljárás és rendszer integrál kiszámításának párhuzamos architektúra szálára való leképezésére." Hungarian and PCT patent, 2012. 2013.

[13] G. Cserey and **A. Rák**, "High accuracy time-to-digital converter on FPGA." Hungarian patent, 2009.

[14] **A. Rák** and G. Cserey, "Számítógépes architektúra és feldolgozási eljárás." Hungarian patent, 2012.

[15] **A. Rák**, G. Cserey, and B. Jákli, "Eszköz és eljárás mért jel időbeliségének meghatározására." PCT patent, 2013.