GPU-BASED ACCELERATION OF THE GENETIC ALGORITHM

Petr Pospíchal
Doctoral Degree Programme (1), FIT BUT
E-mail: ipospichal@fit.vutbr.cz

Supervised by: Josef Schwarz
E-mail: schwarz@fit.vutbr.cz

ABSTRACT

Genetic algorithm, a robust, stochastic optimization technique, is effective in solving many practical problems in science, engineering, and business domains. Unfortunatelly, execution usually takes long time. In this paper, we study a possibility of utilization consumer-level graphics cards for acceleration of GAs. We have designed a mapping of the parallel island genetic algorithm to the CUDA software model and tested our implementation on GeForce 8800GTX and GTX285 GPUs using a Rosenbrock's, Griewank's and Michalewicz's benchmark functions. Results indicates that our optimization leads to speedups up to seven thousand times compared to single CPU thread while maintaing reasonable results quality.

1 INTRODUCTION

Genetic Algorithms (GA) [2] are powerful, domain-independent search techniques inspired by Darwinian theory. In general, GAs employ selection, mutation, and crossover to generate new search points in a state space. A genetic algorithm starts with a set of individuals that forms a population of the algorithm. On every iteration of the algorithm, each individual is evaluated using the fitness function and the termination function is invoked to determine whether the termination criteria have been satisfied. The algorithm ends if an acceptable solutions have been found or the computational resources have been spent.

Although GAs are very effective in solving many practical problems, their execution time can become a limiting factor for some huge problems, because a lot of candidate solutions must be evaluated.

There is variety of possibilities how to accelerate GAs. One of the most promising variant is an island model parallelization. The island models can fully explore the computing power of course grain parallel computers. The population is divided into a few subpopulations, and each of them evolves separately on different processor. Island populations are free to converge toward different sub-optima. The migration operator is supposed to mix good features that emerge locally in the different subpopulations.

Driven by ever increasing requirements from the video game industry, GPUs have evolved into a very powerful and flexible processors, while their price remained in the range of consumer

market. They now offer floating-point calculation much faster than today's CPU and, beyond graphics applications; they are very well suited to address general problems that can be expressed as data-parallel computations (i.e. the same code is executed on many different data elements).

In this paper, we would like to show that consumer-level GPUs have a great potential for acceleration of the genetic algorithms.

2 GPU-BASED GENETIC ALGORITHM

We have chosen the CUDA (Compute Unified Device Architecture) [1] framework to implement our GA on GPU as it promises best achieved results so far.

The GPU is optimised to SIMD-type processing and contains hardware scheduler which swiftly swaps existing threads to hide main memory latency. Because of this, a proposed model should utilize thousands of parallel threads with minimum code branching. NVidia GPU consists of multiprocessors capable to perform tasks in parallel. Threads running in these units are very lightweight and can be synchronized using the barriers so that data consistency is maintained.

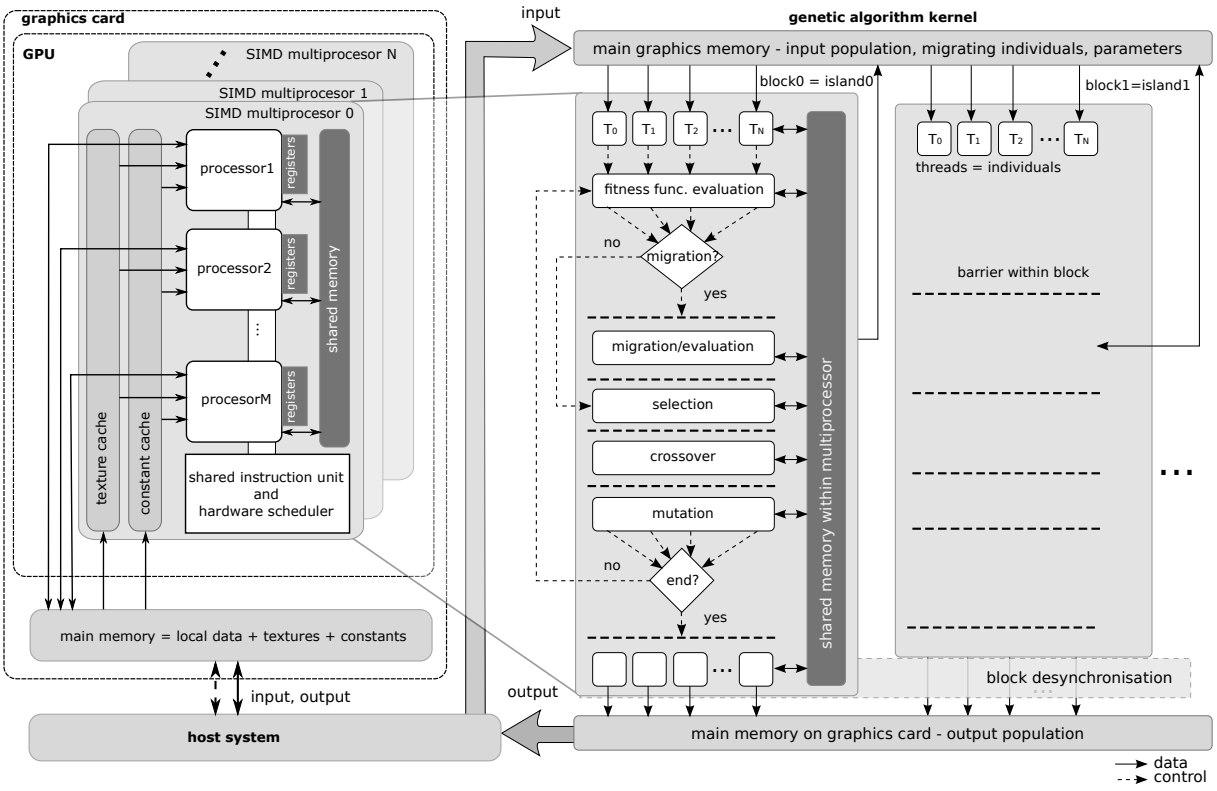


Figure 1: Mapping of the genetic algorithm to CUDA hardware and software model.

The memory attached to graphics cards is divided into two levels — main memory and on-chip memory. The main memory has a big capacity (hundreds of MB) and holds a complete set of data as well as user programs. It also acts as an entry/output point during communication with CPU. Unfortunately, big capacity is outweighed with high latency. On the other hand, the on-chip memory is very fast, but has very limited size. Apart from per-thread local registers, the on-chip memory contains particularly useful per-multiprocessor shared segments. This 16KB

array acts as a user managed L1 cache and, under some conditions, allows concurrent access for whole group of threads. The size of on-chip memory is a strongly limiting factor for designing efficient GA, but existing CUDA applications greatly benefit there.

In order to summarize earlier paragraphs, our primary concern during designing GA accelerated by GPU is to create its efficient mapping to CUDA software model with a special focus on the massive parallelism and usage of the shared memory within multiprocessors.

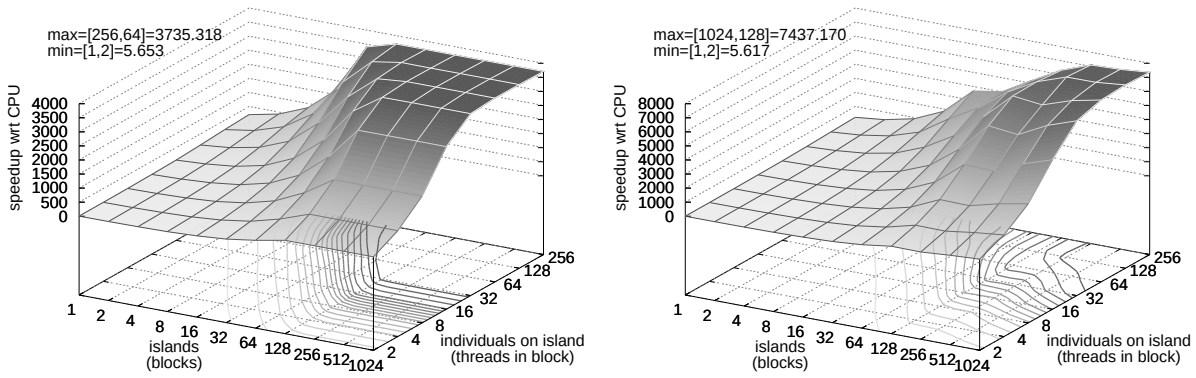
Fig. 1 shows the GA mapping to CUDA model. We assume an island based GA with asynchronous migration along an unidirectional ring. Every individual is controlled by a single CUDA thread. The local populations are stored in shared on-chip memory on particular GPU multiprocessors (CUDA blocks). The local island populations as well as whole islands are thereby evaluated entirely on GPU in parallel. This ensures both computationally intensive execution and massive parallelism needed for the GPU to reach its full potential. As communication between CPU and GPU happens only during results exchange, this model also avoids PCI express bandwidth bottleneck which drastically chokes performance of some existing applications [5, 8, 6].

3 RESULTS

Achievable speedups and solution quality of the proposed GA were examined using Griewank's, Michalewicz's and Rosenbrock's artificial benchmark functions that are often used for GA analysis. Reference CPU version of the GA is a single-thread program running on Core i7 920 implemented using well known GAlib library [7].

3.1 ACHIEVED PERFORMANCE

The speedup of our implementation was investigated using intel Core i7 920 processor and two nVidia consumer-level graphics cards: 8800 GTX (16 multiprocessors / 128 cores) and GTX 285 (30 multiprocessors / 240 cores).



(a) 8800 GTX (b) GTX 285

Figure 2: Speedup on Griewank's function depending on GA parameters and GPU.

The charts shown in Fig. 2 illustrate achieved speedups on two different GPUs against CPU, based on population sizes and the number of simulated island. The 8800 GTX graphic card saturates its computational resources from 256 islands and 32 individuals individuals per an

Table 1: Comparison of the solutions quality.

genes	mean best fitness								
	Rosenbrock			Michalewicz			Griewank		
	CPU	GPU1	GPU2	CPU	GPU1	GPU2	CPU	GPU1	GPU2
2	0.086	3.468	$7.57 \cdot 10^{-7}$	-1.022	-1.768	-1.801	0.0005	0.0020	$3.99 \cdot 10^{-12}$
3	1.897	4.996	0.447	-1.220	-2.336	-2.760	0.0051	0.0048	$1.06 \cdot 10^{-8}$
4	8.900	4.997	0.494	-1.459	-2.748	-3.696	0.0156	0.0188	$1.22 \cdot 10^{-7}$
5	22.112	17.332	2.042	-1.684	-3.184	-4.628	0.0246	0.0414	0.0001
6	48.450	56.045	4.313	-1.817	-3.654	-5.440	0.0408	0.0570	0.0005
7	83.455	42.509	6.903	-2.035	-3.646	-6.163	0.0479	0.0620	0.0012
8	128.710	155.233	9.257	-2.120	-3.805	-6.659	0.0650	0.1360	0.0027
9	167.329	131.737	12.045	-2.176	-4.830	-7.136	0.0749	0.1444	0.0042
10	233.364	184.370	15.379	-2.391	-5.009	-7.649	0.0805	0.1758	0.0058

island. The maximal speedup of 3735 against CPU is reached with 256 islands and 64 individuals per island. The GTX285 provides about twice better peak speedup, but it is necessary to provide much more computational work to it. The computational resources of this GPU are not saturated even for 1024 islands and 128 individuals per island, where this GPU has attacked the speedup of 7437[1]. Situation is similar for Rosenbock's resp. Michalewicz's benchmark functions as maximal speedup reaches 8674 resp. 7760 times in case of GTX285 and 4255 resp. 3957 times in case of 8800 GTX card. CUDA Profiler indicates approx. 99.8% instruction throughput for maximal GPU performance.

3.2 SOLUTIONS QUALITY

The proposed implementation of the tournament selection slightly differs from the original GAlib's one. In order to ensure the same testing condition for the both CPU and GPU versions, the GAlib's selection were reimplemented. Arithmetic crossover and mutation were kept untouched as they have been defined in the same way.

Tests CPU and GPU1 were performed on artificial benchmark functions mentioned earlier on a single island (obviously with no migrations) with 32 individuals, 70% crossover probability, 5% mutation probability and elitism turned off. Each run was terminated after 100 generations of evolution and the best (lowest) fitness value was taken into consideration.

Test GPU2 was performed with the same GA parameters and benchmarks but with maximum GPU exploitation resulting from simulating 1024 populations (islands) in parallel. Additionally, migrations were performed every 10 generations with 3 individuals.

Table 1 shows the mean value over 100 measured runs. Lower values means better solutions for all tested functions. Higher number of genes simulates rising problem complexity. Test GPU2 shows that the fully utilized GPU can achieve far better results in the same number of iterations. Overall, GPU1 results are better than CPU by approx. 20%. This shows that proposed GPU implementation of GA is able to optimise numerical functions.

[1]As it was mentioned earlier, a single threaded CPU implementation was tested. Benchmarked CPU Core i7 allows parallelisation to 4 physical cores + 4 virtual Hyper-Threading ones. Hence, ideally paralleled CPU version with 50% speed benefit from HT technology would change the maximum speedup from 7437 to approx. 1239 times. GAlib also computes variety of additional statistics.

4 CONCLUSIONS

Speedups up to seven thousand times higher clearly show that GPUs have proven their abilities for acceleration of genetic algorithms during optimization of simple numerical functions. The results also show that the proposed GPU implementation of GA can provide better results in the shorter time or produce better results in equal time. Peak GPU performance has more than 400-times better power-to-watt ratio, thus saving electrical energy during process. Furthermore, used graphics card is hundreds times cheaper than any CPU grid at same speed. Solution can be run at any of nVidia GPU supporting ShaderModel 4.0 and on both Linux and Windows platform. Practically each problem, whose solution is codable to string of bits and where the quality of two candidate solution can be compared, is solvable using genetic algorithms. The area of potential applications of the proposed model is thereby very large.

ACKNOWLEDGEMENT

This research has been carried out under the financial support of the research grants "Natural Computing on Unconventional Platforms", GP103/10/1517 (2010-2013) of Grant Agency of Czech Republic, "Security-Oriented Research in Infeormation Technology", MSM 0021630-528 (2007-13), the BUT FIT grant FIT-S-10-1 and the research plan MSM0021630528.

REFERENCES

[1] NVIDIA, C.: Compute Unified Device Architecture Programming Guide. NVIDIA: Santa Clara, CA, 2007.

[2] Holland, J. H.: Adaptation in Natural and Artificial Systems. University of Michigan Press, 1975.

[3] Pharr, M. and Fernando, R.: GPU Gems 2: Programming Techniques for High-Performance Graphics and General-Purpose Computation. Addison-Wesley Professional, 2005.

[4] Yu Q., Chen, C., and Pan Z.: Parallel genetic algorithms on programmable graphics hardware. In Advances in Natural Computation ICNC 2005, Proceedings, Part III, volume 3612 of LNCS, pp 1051–1059, Changsha, August 27-29 2005.

[5] Li, J.-M., Wang, X.-J, He, R.-S. and Chi, Z.-X.: An efficient fine-grained parallel genetic algorithm based on gpu-accelerated. In Network and Parallel Computing Workshops, 2007. NPC Workshops. IFIP International Conference on, pp 855–862, 2007.

[6] Maitre, Q., Baumes, L.A., Lachiche, N., Corma, A. and Collet, P.: Coarse grain parallelization of evolutionary algorithms on GPGPU cards with EASEA, In: Proceedings of the 11th Annual conference on Genetic and evolutionary computation table of contents, Montreal, Québec, Canada, 2009, pp. 1403-1410, ISBN 978-1-60558-325-9.

[7] Matthew, W.: GAlib: A C++ Library of Genetic Algorithm Components. Massachusetts Institute of Technology, 1996.

[8] Wong, M.L and Wong, T. T.: Implementation of Parallel Genetic Algorithms on Graphics Processing Units, In: Intelligent and Evolutionary Systems, Vol. 187/2009, Springer Berlin / Heidelberg, pp. 197-216, ISBN 978-3-540-95977-9.