Jiří Khun; Ivan Šimeček

Parallelization of artificial immune systems using a massive parallel approach via modern GPUs

In: Jan Chleboun and Petr Přikryl and Karel Segeth and Jakub Šístek and Tomáš Vejchodský (eds.): Programs and Algorithms of Numerical Mathematics, Proceedings of Seminar. Dolní Maxov, June 8-13, 2014. Institute of Mathematics AS CR, Prague, 2015. pp. 106--111.

Persistent URL: http://dml.cz/dmlcz/702670

Programs and Algorithms of Numerical Matematics 17
J. Chleboun, P. Přikryl, K. Segeth, J. Šístek, T. Vejchodský (Eds.)
Institute of Mathematics AS CR, Prague 2015

PARALLELIZATION OF ARTIFICIAL IMMUNE SYSTEMS USING A MASSIVE PARALLEL APPROACH VIA MODERN GPUS

Jiří Khun, Ivan Šimeček

Department of Computer Systems,
Faculty of Information Technology,
Czech Technical University in Prague,
Thákurova 9, 160 00 Prague 6, Czech Republic
jiri.khun@fit.cvut.cz, ivan.simecek@fit.cvut.cz

Abstract

Parallelization is one of possible approaches for obtaining better results in terms of algorithm performance and overcome the limits of the sequential computation. In this paper, we present a study of parallelization of the opt-aiNet algorithm which comes from Artificial Immune Systems, one part of large family of population based algorithms inspired by nature. The opt-aiNet algorithm is based on an immune network theory which incorporates knowledge about mammalian immune systems in order to create a state-of-the-art algorithm suitable for the multimodal function optimization. The algorithm is known for a combination of local and global search with an emphasis on maintaining a stable set of distinct local extrema solutions. Moreover, its modifications can be used for many other purposes like data clustering or combinatorial optimization. The parallel version of the algorithm is designed especially for modern graphics processing units. The preliminary performance results show very significant speedup over the computation with traditional central processor units.

1. Introduction

Research behind this paper represents an intersection of several scientific disciplines but two of them are playing a key role: artificial immune systems (AIS) and design of parallel algorithms with an emphasis on general purpose computing via graphic processing units (GPU).

1.1. Artificial immune systems

Artificial immune systems are a part of a large field of computational intelligence approaches inspired by nature. Their basic principles are based on the knowledge obtained by studying real biological immune systems, especially mammalian.

In general, we can imagine any biological immune system as a mechanism which responses on varied incoming threats represented by pathogens and toxic substances in order to protect the host organism. Pathogens can be represented by a wide

range of different types of micro-organisms like parasites, bacteria, viruses, prions and others. Every immune system's main task is to detect such threats and try to eliminate them.

After ages, the mammalian immune system has developed itself into a very complex part of a host body and the development is still in progress based on everyday life experiences. This is the main reason why the immune systems became an inspiration for the computational intelligence area.

Theories behind the biological immune systems have become an inspiration for many algorithms like Clonal Selection, Negative Selection, Danger Theory, Theory of Immune Network and others [1]. Our research is focused on the Immune Network algorithms which enhanced an original theory of the clonal selection.

1.2. General computing using graphics processing unit

Almost every modern GPU is able to provide a general purpose computational performance at least an order of magnitude bigger than a present-day central processing unit (CPU). In the area of high performance computing (HPC) and supercomputers, the use of GPUs is a standard way to achieve a significantly higher performance than delivered by previous generation computers while keeping the same power consumption.

Great performance hidden in GPUs is achieved by large amount of simple processing elements working in parallel. Every individual element deals only with a small subset from the running task. Therefore, it is necessary to carefully design an algorithm for this parallel approach. But there are also many tasks that cannot be solved in parallel at all due to their sequential nature. Parallel approach has to deal with several aspects like data hazards or synchronization and, consequently, is increasing algorithm complexity.

2. Opt-aiNet

The opt-aiNet algorithm [2] is based on the theory of Immune Network which came with an idea that immune cells do not react only to foreign pathogens but can also react to other immune cells, see Algorithm 1. Thanks to this, the whole immune environment becomes a dynamic self-regulated network where individual cells constantly excite and inhibit each other. This behavior also leads to a richly diverse population of immune cells capable to react to a broad spectrum of possible threats.

An original version of the aiNet algorithm was intended for data clustering and was later extended to deal with optimization tasks. In our research, we are focusing on the version for continuous multi-modal optimization, but results of the research will be applicable across all modifications of the algorithm.

Figure 1 shows an example of the algorithm's results: identified local maxima of Schaffer's function of two real-valued variables.

Algorithm 1 Pseudocode for the opt-aiNet aglorithm [1]

```
1: procedure OPT-AINET
Input: PopulationSize, ProblemSize, Nclones, Nrandom, AffinityThreshold
Output: BestCell
2:     Population ← InsertInitialPopulation(PopulationSize, ProblemSize);
3:     while not(TerminationCondition) do
4:         EvaluatePopulation(Population);
5:         BestCell ← GetBestSolution(Population);
6:         Progeny ← (nothing)
7:         AvgPopFitness ← 0;
8:         while CalculateAvgPopFit(Population) > AvgPopFitness do
9:             AvgPopFitness ← CalculateAvgPopFit(Population);
10:            for Cell(i) in Population do
11:                Clones ← CreateClones(Celli, Nclones);
12:                for Clone(i) in Clones do
13:                    Clone(i) ← MutateAccordingFitnessParent(Clone(i), Cell(i));
14:                EvaluatePopulation(Clones);
15:                Progeny ← GetBestSolution(Clones);
16:    SupressLowAffinityCells(Progeny, AffinityThreshold);
17:    Progeny ← CreateRandomCells(Nrandom);
18:    Population ← Progeny;
19:    return BestCell;
```

3. Analysis of parallelization

Design of parallel algorithms for general purpose GPU computations (GPGPU) is not always a straightforward and simple approach. Especially for tasks that aren't completely data parallel. There are several rules that must be met otherwise the computation performance can be even much lower that on a CPU.

The most important rule is full utilization of GPU resources. It is necessary to run thousands or more independent computational threads. Only such amount of threads can hide some problematic architectural areas like relatively big latency between GPU's cores and their memory.

Another limitation during the design of the parallel algorithm for GPU is the fact that individual threads are run in groups containing tens of threads (usually 64). The threads within a group are using the same instruction buffer and must perform the same program's code. Therefore if any of the threads is branching, the computation must be serialized with a significant negative impact on the performance.

3.1. Parallelization of opt-AiNet

The opt-aiNet algorithm is relatively complex and contains a lot of data dependencies. Therefore it is not reasonable to run it in parallel like a one piece of code.

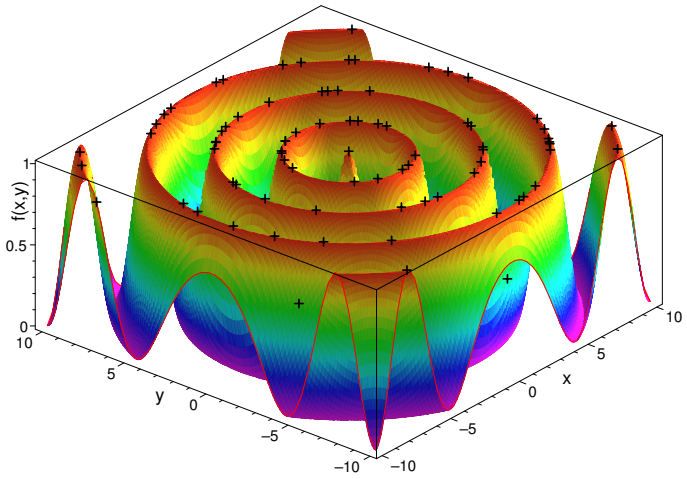

Figure 1: An example of results of multimodal search performed by opt-AiNet on a testing function of two real-valued variables (Schaffer's function). Individual marks represent the found maxima of the function. Redrawn from [2].

During our analysis we discovered 6 individual parallel regions that can be effectively parallelized in a large-scale satisfying the above-mentioned requirements for the efficient GPU computation. The parallel regions will be discussed in the following subsection in detail.

Not all parallel regions are purely data-parallel. Some of them contain internal data dependencies that require intra-thread communication (e.g., via special so-called *shared* memory) and synchronization. Some regions also represent a parallel reduction pattern that require a lot of intra-kernel synchronization.

The thread synchronization and communication represent the biggest challenge during the parallelization because GPU threads are sensitive to synchronization methods and a wrong approach can devastate the overall performance.

3.2. Parallel regions within the opt-AiNet algorithm

As mentioned above, our analysis showed that the algorithm can be divided into several parts with a potential for the large parallelization targeting GPU. These parallel regions are covering almost all necessary steps that must be processed during the algorithm's execution.

3.2.1. Insertion of an initial or an additional population

The insertion of an initial population, consisting of individual immune cells, is the first step of the algorithm. Another cells are also inserted during later stages of the algorithm.

The insertion can be fully done in parallel without any significant obstacle. Every computational thread will insert one or more immune cells and there is not any relation between inserted cells.

It is important to highlight that every computational thread need to have an own independent random number generator because, for proper results of the algorithm, every cell has to be generated with completely random initial configuration values.

3.2.2. Calculation of function values

In this step every cells' internal configuration is used for the calculation of the functional value of the optimized function. The calculation of the function values can be done fully in parallel and in any order because there are not any direct bindings between individual cells during this step.

3.2.3. Calculation of fitness values

The fitness value represents how close the particular immune cell is to the currently best found solution (the maxima of the optimized function). It is an important value that influences another life span of the particular cell (solution).

The calculation of the fitness value is not a data parallel task because at the beginning it is necessary to found the largest function value across all cells in the population. The parallel reduction pattern can be used for this approach. A complication within this approach is the out of order execution of the groups of threads (as mentioned above) on GPUs that requires intergroup synchronization.

3.2.4. Cloning and mutation of the population

In this step, the individual immune cells are cloned and mutated with certain probability. The probability is based on the fitness value due to the fact that the cells with better fitness values have higher chance to clone themselves.

This leads to a varying size of the population that represents another problem for a real implementation of the algorithm on a GPU because current program model needs to specify memory regions in advance before computation.

On the other hand, the mutation as a subsequent step after the cloning does not represent a difficult task and can be done fully in parallel assuming independent random number generator for every computational thread.

3.2.5. Selection of the best cells

During every iteration of the opt-aiNet algorithm, a proportional part of cells is selected for transfer to the next generation. In general, only the best solutions are chosen therefore the algorithm needs to be able to search the whole population in parallel and select them. Logical approach is to sort all cells by the fitness value and then select the part of the cells with required quality.

This can be done in parallel with the help of a parallel sort pattern. There are several types of the parallel sort algorithm suitable for GPU implementation. For example the merge sort.

3.2.6. Suppression of similar cells

This step represents the part of the algorithm where low affinity cells with fitness lower than the required level are suppressed in order to avoid situations where too

many cells within a population are covering the same state space. It is the most challenging part in the whole parallelization processes of the opt-aiNet algorithm because every cell in the population must be compared against the rest.

Our current approach is to sort all cells in parallel by their affinity with the help of the parallel merge pattern. Then we do the suppression on a local level within individual groups of threads (every computational thread represents one cell). The last step is to apply the suppression to the individual groups of threads on a global level.

4. Conclusion

Opt-aiNet represents an important algorithm intended for the multi-modal search. It comes from the family of AiNet algorithms which are influencing wide area of data processing tasks like data clustering, data compression or combinatorial optimization. As many other nature-inspired heuristics, opt-aiNet is capable to perform very well in terms of solutions' quality but it needs a corresponding amount of the compute power. Therefore any possible improvement in this area is welcome.

Within this paper, we discussed possibilities of parallelization of the opt-aiNet algorithm as a potential source of a large improvement from the perspective of computational performance. We have focused especially on massive parallel approach represented by modern GPUs allowing universal non-graphical computations because these devices start to be a common part of almost every present-day supercomputer, work-station or a notebook.

Our analysis is showing a large potential of possible parallelization even for the massive parallel approach represented by GPUs and their thousands of computational threads. On the other hand, there are also obstacles which make the parallelization relatively challenging and non-trivial.

Our preliminary testing implementation is showing promising results and we are expecting a speed-up factor of 5 at least if we compare the original sequential approach running on a present-day average CPU (Intel Core i5 4200M) and the GPU implementation running on a low-end GPU (AMD Radeon HD 8750M).

Acknowledgements

This research has been supported by SGS grant No. SGS14/106/OHK3/1T/18.

References

[1] Brownlee, J.: *Clever Algorithms: Nature-inspired programming recipes*. LuLu, 1st edition, 2011.

[2] de Castro, L. N. and Timmis, J.: An Artificial immune network for multimodal function optimization. In: *Proceedings of the 2002 Congress on Evolutionary Computation (CEC'02)*, vol. 1, pp. 699–704, May 2002.