A GPU-Based Implementation of Differential Evolution for Solving the Gene Regulatory Network Model Inference Problem

Luis E. Ramírez-Chavez
CINVESTAV-IPN (Evolutionary Computation Group)
Department of Computer Science
Av. IPN No. 2508, Col San Pedro Zacatenco
Mexico City, Mexico
eramirez@computacion.cs.cinvestav.mx

Carlos A. Coello Coello
CINVESTAV-IPN (Evolutionary Computation Group)
Department of Computer Science
Av. IPN No. 2508, Col San Pedro Zacatenco
Mexico City, Mexico
ccoello@cs.cinvestav.mx

Eduardo Rodríguez-Tello
CINVESTAV-Tamaulipas, Information Technology Laboratory
Parque Científico y Tecnológico TECNOTAM, Km 5.5 carretera Cd. Victoria
Cd. Victoria, Tamaulipas, Mexico
ertello@tamps.cinvestav.mx

ABSTRACT

In this paper, we present what we believe to be the first GPU-based implementation (using CUDA) for solving the gene regulatory network model inference problem. Our implementation uses differential evolution as its search engine, and adopts a power law system of differential equations (an S-System) for modelling the dynamics of the gene regulatory networks of our interest. Our preliminary results indicate that the use of GPUs produces an important reduction in the computational times required to solve this costly optimization problem. This could bring important benefits in Bioinformatics because of the many practical applications that the solution of this problem has.

Categories and Subject Descriptors

J.3 [**Life and Medical Sciences**]: Biology and genetics; I.2.8 [**Problem Solving, Control Methods, and Search**]: Heuristic Methods; G.1.0 [**Numerical Analysis, General**]: Parallel Algorithms

General Terms

Heuristics, Bioinformatics, Parallel Computing

Keywords

Gene Regulatory Networks, S-Systems, Evolutionary Algorithms, GPU, CUDA

1. INTRODUCTION

Gene regulatory networks [4] are networks of interaction between different genes. That is, the level of influence that a set of genes has on another specific gene to either promote or suppress the expression of such gene. Knowing and understanding these relationships is one of the most important objectives in the study of the functioning of biological systems, since this knowledge could represent the development of treatments for diseases for which no treatment currently exists. This could be done by predicting the behavior of an organism when using a certain medication. Indeed, the study of gene regulatory networks may also lead to the intelligent design of synthetic organisms, because it could reveal one of the basic mechanisms of life [3].

Searching for these models only through biological experiments (e.g., DNA microarrays) is an impractical and costly task. A better approach is to use a small set of biological data which had been obtained by direct experimentation, and then analyze it with a computer, with the aim of doing some sort of reverse engineering to nature [14]. Clearly, the aim is to obtain a computational model that can describe these complex biological mechanisms [6].

A variety of approaches have been undertaken to model gene regulatory networks [22]. However, when tackling this problem as an optimization task (particularly, when using evolutionary algorithms to solve it), the most popular modelling approach has been to adopt ordinary differential equations [7]. Particularly, S-systems [21] have been the most popular choice within the evolutionary computation community [18] in spite of the large number of parameters that need to be inferred when using them. S-systems are nonlinear differential equations that adopt a canonical power law representation to approximate the behavior of a dynamic system [24]. However, the use of S-systems has been found to be a relatively robust model from which parameters can be inferred using a metaheuristic (evolutionary algorithms have been the most popular choice in this regard).

Evolutionary algorithms are very suitable to be parallelized, and this feature turns out to be very useful when trying to solve the gene regulatory network model inference problem, because of its high computational cost. The use of Graphics Processing Units (GPUs), which have become increasingly cheaper, is an ideal choice to process massive amounts

of operations in parallel, achieving higher speed ups than multi-core processors if properly programmed. The goal of this paper is precisely to show a GPU-based implementation of differential evolution, which is tailored to solve the gene regulatory network model inference problem.

The remainder of this paper is organized as follows. Section 1.1 provides a short introduction to evolutionary algorithms in general, and differential evolution in particular, since the latter is the metaheuristic adopted in this paper. In Section 2, we give a short introduction to gene regulatory networks and to S-systems, since that is the mathematical model that we adopt in this paper. Graphical processing units and CUDA are introduced in Section 3. Our proposed approach is described in Section 5. Section 6 describes the experimental setup adopted for validating our proposed approach as well as the results obtained. Our conclusions and some possible paths for future research are provided in Section 7.

1.1 Evolutionary Algorithms

The term *evolutionary algorithms* (EAs) is generically applied to a set of stochastic techniques inspired by the Neo-Darwinian theory of natural evolution. EAs are commonly used to solve search and optimization problems [10]. Although three main paradigms are normally considered within EAs (i.e., evolutionary programming [11], evolution strategies [23], and genetic algorithms [12]), generally speaking, any approach whose selection mechanism is based on a measure of quality of the solutions (the so-called *fitness*) can be seen as an EA. EAs mimic (at different levels of similarity) the evolution of a population of individuals (i.e., a set of solutions to a problem) which are subject to certain variation operators (e.g., crossover and mutation).

Differential evolution (DE) is a particular type of evolutionary algorithm originally proposed in the mid-1990s by Kenneth Price and Rainer Storn for continuous optimization [27, 20]. However, unlike simple genetic algorithms, DE does not use an encoding of solutions and, unlike evolution strategies, DE does not use a probability density function to self-adapt its parameters. Instead, DE performs mutation based on the distribution of the solutions in the current population. In this way, search directions and possible step sizes depend on the location of the individuals selected to calculate the mutation values.

There is a nomenclature scheme developed to reference the different DE variants. The most popular DE variant is called *"DE/rand/1/bin"*, where "DE" means Differential Evolution, the word "rand" indicates that individuals selected to compute the mutation values are chosen at random, "1" is the number of pairs of solutions chosen and finally "bin" means that a binomial recombination is used. The corresponding algorithm of this variant (which is the one adopted in this paper) is presented in Figure 1.

The "CR" parameter controls the influence of the parent in the generation of the offspring. Higher values mean less influence of the parent. The "F" parameter scales the influence of the set of pairs of solutions selected to calculate the mutation value (one pair in the case of the algorithm in Figure 1).

2. GENE REGULATORY NETWORKS

A gene regulatory network (GRN) describes the way in which a group of genes are interacting between them [8]. Genes can be expressed at diffent rates, known as *expression levels*. Such an expression level has an influence on the type of proteins that a gene produces. Conversely, the proteins that are generated by a gene produce reactions in a cell, and some of these reactions could constitute the expression level of the other genes. Such interactions precisely form the gene regulatory network that are the focus of study of this paper.

To see how the expression level has varied under different circumstances, biologists created a technique called DNA microarray, which is a collection of microscopic DNA spots attached to a solid surface. DNA microarrays can measure the expression levels of a large number of genes simultaneously.

The motivation for understanding how gene regulatory networks work, is that such an understanding can lead to predicting the behavior of such networks, which has lots of applications in areas such as drug design.

2.1 S-Systems

One of the most commonly used methods to model gene regulatory networks (particularly when using evolutionary algorithms in this area) is through a type of systems of ordinary differential equations called S-Systems.

The general form of an S-System for representing a gene regulatory network is the following:

$$\frac{dX_i(t)}{d(t)} = \alpha_i \left(\prod_{j=1}^{N} X_j^{g_{ij}(t)} \right) - \beta_i \left(\prod_{j=1}^{N} X_j^{h_{ij}(t)} \right) \quad (1)$$

where: N represents the number of genes involved in the gene regulatory network, $g_{ij}0$ (the G matrix) represents the values of the kinetic orders for synthesis (promotes expression), h_{ij} (the H matrix) represents the values of the kinetic orders for degradation (suppresses expression), α and β, where $\alpha > 0$ and $\beta > 0$ are rate constants for the S-System, $X_j(t)$ represent the level of expression of the gene X_j at time t.

The S-System has a major disadvantage in that it includes the large number parameters that have to be estimated. The total number of parameters in S-Systems is $2N(N+1)$, where N is the number of genes (X_i). This quadratically increases the number of parameters to infer for each gene involved in the network. Thus, the evolutionary algorithm searches the best parameters $g_{ij}0$ (the G matrix), h_{ij} (the H matrix) and vectors α and β that model the network.

3. GRAPHICS PROCESSING UNITS

Graphics processing units (GPUs) are specialized circuits design to rapidly manipulate and alter memory in such a way that graphics related tasks such as the building of images in a frame buffer are considerably accelerated. GPUs

```
1  Begin
2      G=0
3      Create a random initial population x_{i,G} ∀i, i = 1,...,NP
4      Evaluate f(x_{i,G}) ∀i, i = 1,...,NP
5      For G=1 to MAX_GEN Do
6          For i=1 to NP Do
7              Select randomly r1 ≠ r2 ≠ r3 :
8              j_rand = randint(1, D)
9              For j=1 to D Do
10                 If (rand_j[0,1) < CR or j = j_rand) Then
11                     u_{i,j,G+1} = x_{r3,j,G} + F(x_{r1,j,G} − x_{r2,j,G})
12                 Else
13                     u_{i,j,G+1} = x_{i,j,G}
14                 End If
15             End For
16             If (f(u_{i,G+1}) ≤ f(x_{i,G})) Then
17                 x_{i,G+1} = u_{i,G+1}
18             Else
19                 x_{i,G+1} = x_{i,G}
20             End If
21         End For
22         G = G + 1
23     End For
24 End
```

Figure 1: "DE/rand/1/bin" algorithm. randint(min,max) is a function that returns an integer number between min and max. rand$[0,1)$ is a function that returns a real number between 0 and 1. Both are based on a uniform probability distribution. "NP", "MAX_GEN", "CR" and "F" are user-defined parameters. "D" is the dimensionality of the problem.

were introduced in 1999, and are defined as[1]: "a single chip processor with integrated transform, lighting, triangle setup/clipping, and rendering engines that is capable of processing a minimum of 10 million polygons per second."

In recent years, a new concept that has gained increasing popularity has been to use a general purpose GPU as a modified form of a stream processor. The idea is to transform the massive floating-point computational power of GPUs into general-purpose computing power. In fact, in some applications that require massive vector operations, the use of this general purpose GPUs can yield several orders of magnitude higher performance than using a conventional CPU.

Since the year 2005, there has been a lot of interest in developing GPU-based implementations of evolutionary algorithms [29], with the obvious aim of reducing the computational time required for evaluating expensive fitness functions (e.g., in genetic programming [13]).

3.1 CUDA

In order to motivate applications of GPUs in computationally expensive applications, NVidia [5] developed an API extension of the C programming language called CUDA (Computer Unified Device Architecture), which allows the definition of specific functions from a normal C program to run on the GPU's stream processors.

The architecture of the GPUs for the CUDA model is illustrated in Figure 2 in which the cores of the GPU are grouped in blocks of cores that share a fast access memory that can only be viewed by the block of cores. However, there is also a global memory that can be accessed by all the cores of the GPU. The communication between the GPU and the CPU takes place through the PCIe port and, passing data from the CPU to the GPU must be done by copying regions of memory from the CPU's memory to the GPU's global memory, so that the data stored in the GPU cannot be viewed by the CPU and vice versa.

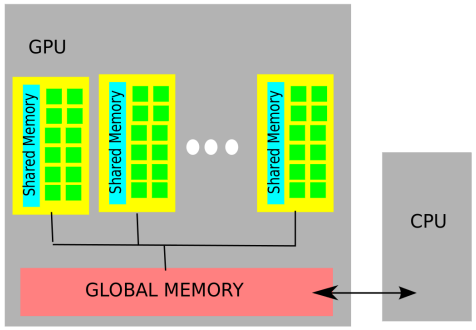

Figure 2: Nvidia's GPUs architecture.

GPUs can launch a high number of execution threads per block which is the reason why GPUs' parallelism is refereed to as *massive parallelism*.

[1]See: `http://www.nvidia.com/object/gpu.html`

4. PREVIOUS WORK

Several researchers have adopted EAs in recent years, to tackle the gene regulatory network inference problem [25, 15, 17, 26, 19, 16, 18].

A variety of evolutionary algorithms have been used to solve this problem, including genetic algorithms with binary encoding, genetic algorithms with real-numbers encoding, evolution strategies and differential evolution. In fact, even multi-objective evolutionary algorithms [26, 17] and hybrids (e.g., combinations of artificial neural networks and EAs [1]) have been adopted to tackle this problem.

All these approaches have the same basic framework to solve the GRN model inference problem:

- Each individual in the population encodes the parameters that define an S-System.
- The objective of the EA is to do reverse engineering to the problem, so that we obtain a model that can reproduce the given target data.

The most common (and perhaps simplest) fitness function adopted in the specialized literature is the mean quadratic error between the original data and the candidate solutions:

$$f = \sum_{i=1}^{N} \sum_{t=1}^{T} \left(\frac{X_{i,ca,t} - X_{i,exp,t}}{X_{i,exp,t}} \right)^2 \quad (2)$$

where: N represents the number of genes, $X_{i,ca,t}$ represents the level of expression of the gene X_i at time t that has produced a candidate solution, $X_{i,xpt}$ represents the level of expression of the gene X_i at time t of the biological experiment, and T is the number of time points.

Thus, the problem of inferring a GRN model can be viewed as an optimization problem in which the objective is to minimize equation (2).

The following is a general method to solve this problem using an EA:

1. First, the individuals in the population are randomly generated (i.e., the S-Systems encoded by each individual in the population is randomly generated)
2. The solutions are evaluated by solving the S-System using a numerical method to solve ODEs (in our case, we adopt Runge Kutta's method [2]).
3. After the S-System has been solved, the result obtained is compared with respect to the original data and the fitness value of each individual is computed using equation (2).
4. The EA operators are applied to the individuals in the population.
5. Steps 2 to 4 are repeated until reaching a termination condition (e.g., a maximum number of iterations).

5. OUR PROPOSED APPROACH

We have developed a GPU-based implementation of differential evolution that solves the gene regulatory network inference problem. The flowchart of our proposed approach is shown in Figure 3. In our implementation, we parallelize the generation of the individuals in the population, the application of the operators of the differential evolution algorithm, and the fitness function evaluations.

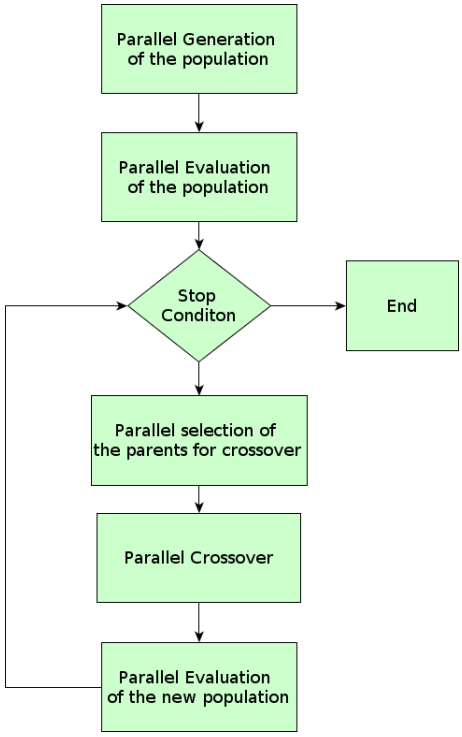

Figure 3: Flowchart of our proposed approach

5.1 Memory Allocation

At the beginning of the algorithm's execution, we reserve all the memory that the algorithm will require for each individual of the population, for the parameters of the S-System, for the matrices G and H. This follows the proposal of Veronese [9], who reported what seems to be the first GPU-based implementation of the differential evolution algorithm. The vectors α and β are mapped into one continuous linear block of memory as shown in Figure 4. The pseudocode (in CUDA) for the memory allocation process is shown in Algorithm 1.

5.2 Parallel Generation of the Population

One of the challenges in the implementation is the requirement of a big number of random parameters. For that, we allocate the memory space required in the global memory of the device. At the beginning of the algorithm's execution, we generate all the numbers that the algorithm will require. We do that in parallel using the Curand library, which is a library that was specially designed to generate random numbers in CUDA. The pseudocode for generating the population is shown in Algorithm 2.

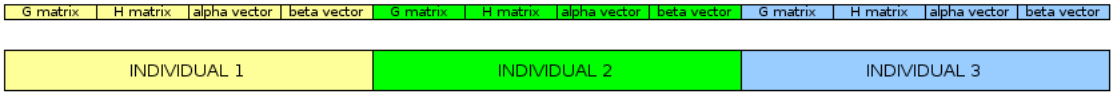

Figure 4: Memory allocation for the individuals of the population

Algorithm 1 Memory Allocation: We reserve the memory that the population needs to encode the parameters of the S-systems, the time series and the fitness of each individual.

cudaMalloc d_pop_G {reserve memory of size ngenes*ngenes*POPSIZE for matrix G of the population}

cudaMalloc d_pop_H {reserve memory of size ngenes*ngenes*POPSIZE for matrix H of the population}

cudaMalloc d_pop_alpha {reserve memory of size ngenes*POPSIZE for vector α of the population}

cudaMalloc d_pop_beta {reserve memory of size ngenes*POPSIZE for vector β of the population}

cudaMalloc d_pop_fitness {reserve memory of size POPSIZE for the fitness of the population}

cudaMalloc d_pop_X {reserve memory of size ngenes*npoints*POPSIZE for the time series of each individual of the population}

Algorithm 2 Parallel generation of the population

curandGenerateUniform(gen,popG,ngenes*ngenes*POPSIZE)
curandGenerateUniform(gen,popH,ngenes*ngenes*POPSIZE)
curandGenerateUniform(gen,popalpha,ngenes*POPSIZE)
curandGenerateUniform(gen,popbeta,ngenes*POPSIZE)
the we fit the random numbers to the search bounds of the problem
fitvals< < <grid_size,block_size > > >(popG, mingij, maxgij)
fitvals< < <grid_size,block_size > > >(popH, minhij, maxhij)

5.3 Parallel Evaluation of the Population

We implemented a parallel evaluation scheme of the population. For that sake, we assigned a thread execution per individual; thus, in our implementation, each thread performs one function evaluation. In our implementation, we launch blocks of 512 threads. The code for the parallel evaluation of the population is shown in Algorithm 3.

Algorithm 3 Parallel evaluation of the population.

callrungekutta< < < POPSIZE/512 + 1 , 512 > > >(G,H,alpha,beta,fitness,timeseries)

Each execution thread invokes a fourth order Runge Kutta's method to compute the solution of the S-System that is encoded in each individual in the population. After that, each thread will also compute the fitness (i.e., the mean squared error between the target data and the solution of the S-System) of each individual. This is done in a `__device__` function. This process is shown in Algorithm 4.

Algorithm 4 Fitness calculation.

Compute the solution of the S-System of each individual
For each solution in each thread, compute the mean squared error between the target data and the solution of the S-System

5.4 Parallel Differential Evolution operators

In our implementation, we provide a parallel crossover operator (see Algorithm 5) and a parallel selection mechanism (see Algorithm 6). These operators were implemented as suggested in [9].

Algorithm 5 Parallel crossover.

For each individual in the population, we randomly select three other individuals, using the Curand library. We perform this in a loop until fulfilling the condition that the four individuals selected are different
We perform crossover to the selected individuals using one thread for each individual, so that the operator can be applied simultaneously to all the individuals in the population

Finally, we choose the best individual between each parent and the offspring generated by the crossover operator. This is also done in parallel using a thread per individual: `kernelselection< < < (POPSIZE/512) + 1, 512 > > >(old population , new population)`.

The process is repeated until reaching the maximum number of generations.

6. EXPERIMENTAL SETUP AND RESULTS

To assess the advantages of our GPU-based implementation, we tested our algorithm using the same instance adopted by Tominaga [28]. This instance consists of a network of two genes artificially generated by the parameters shown in

Algorithm 6 Parallel selection

```
tid = threadIdx.x + (blockIdx.x * blockDim.x);
while tid < POPSIZE do
  if new individual tid fitness < old individual tid fitness then
    old individual = new individual
  end if
end while
```

i	α_i	β_i	$g_{i,1}$	$g_{i,2}$	$h_{i,1}$	$h_{i,2}$
1	3	3	0	2.5	-1	0
2	3	3	-2.5	0	0	2

Table 1: S-System Parameters of the gene regulatory network instance adopted to validate our proposed approach

Table 1. The gene expression levels of the network are shown in Figure 5, and it consists of a data set of 50 time points of expression level per gene. The parameters for the limits of the variables are the same adopted by Tominaga: for α_i and β_i the search range is [0.20.0] and for g_{ij},h_{ij}, the search space ranges in [-4.0,4.0].

For the CPU version, the tests were conducted using an Intel Core 2 Quad processor running at 2.6 GHz and having 2 GB of memory. For the GPU implementation, we adopted a GeForce GTX 460 with 336 cores running at 765 MHz and having 1 GB of memory. The parameters of the differential evolution algorithm were the following: F=0.5, crossover rate (CR) = 0.8.

6.1 Validating our implementation

In order to know if our implementation is correct, we ran some tests to see if we could exactly reproduce the target data. The results of these test is graphically shown in Figures 6 and 7. It can be seen that the estimated levels of expression for the genes one and two, are almost identical to the test data. The final fitness value of these results is $.016232\times10^{-5}$.

6.2 Results

Table 2 show a comparison of the results obtained by both, the sequential CPU version and the GPU-based implementa-

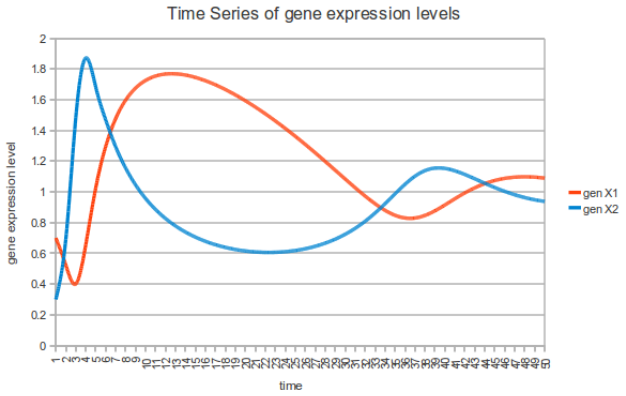

Figure 5: Time courses of the test instance for two genes

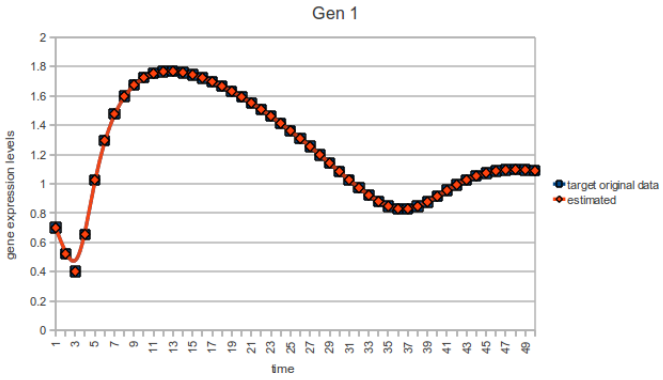

Figure 6: Target time-series and estimated time-series for gene one.

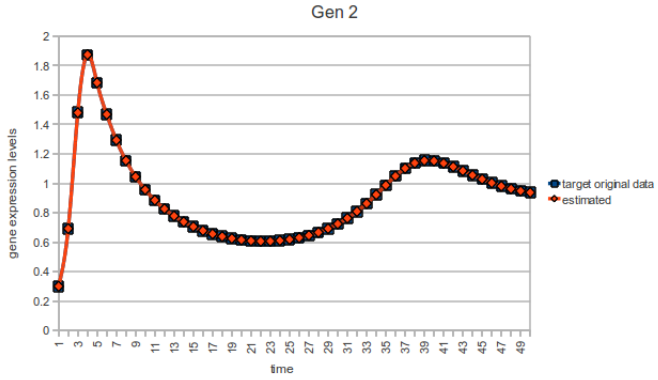

Figure 7: Target time-series and estimated time-series for gene 2

tions of our algorithm. In Table 2, we show different configurations in which we combine different population sizes with different numbers of generations. On the left handside, we show the fitness value and CPU times required (in seconds), for the sequential implementation. The same information is shown on the right handside of the table.

6.3 Discussion

Our results clearly indicate that our GPU-based implementation produces a significant time reduction for this costly problem. For the configuration in which we use the maximum numbers of individuals (4056) and generations (1024), our GPU-based implementation provided a speeded of about 145 times with respect to the sequential implementation. Thus, our GPU-based implementation required in this case of only 6.35 seconds, while the sequential implementation required 919.65 seconds. Also, it is worth noting that the final fitness values achieved are reasonably close to zero, which is the target value.

It is worth noting that in [28], Tominaga reported that a genetic algorithm required 4085 seconds to solve the exact same problem. Tominaga's implementation used a population of 1000 individuals, which ran during 1000 generations, and achieved a final fitness value similar to ours (8×10^{-5}). It is worth noting, however, that Tominaga used older hardware than us.

7. CONCLUSIONS AND FUTURE WORK

Our main conclusion is that the use of GPU-based computing is a viable alternative for solving the gene regulatory network model. Our results have shown a significant time reduction when using our GPU-based implementation, with respect to the sequential version of the same algorithm.

As part of our future work, we plan to test our GPU-based implementation using additional problems that have been reported in the specialized literature, including the use of larger networks having 5, 10 and up to 20 genes. Such examples could test the scalability of our implementation, since a sequential implementation normally requires about 10 hours to reach an acceptable solution for networks having 5 genes, while larger networks may require several days.

Acknowledgements

The first author acknowledges support from CONACyT to pursue graduate studies at the Computer Science Department of CINVESTAV-IPN. He also acknowledges support from CINVESTAV-IPN to present this paper. The second author acknowledges support from CONACyT project no. 103570.

8. REFERENCES

[1] S. Ando and H. Iba. Estimation of gene regulatory network by genetic algorithm and pairwise correlation analysis. In *The 2003 IEEE Congress on Evolutionary Computation (CEC'2003)*, pages 207–214, Canberra, Australia, December 2003. IEEE Press.

[2] U. M. Ascher and L. R. Petzold. *Computer Methods for Ordinary Differential Equations and Differential-Algebraic Equations*. SIAM, Philadelphia, USA, July 1998. ISBN 978-08987-1412-8.

[3] A. Barabasi and Z. Oltvai. Network biology: Understanding the cell's functional organization. *Nature Reviews Genetics*, 5(2):101–U15, February 2004.

[4] H. Bolouri and E. Davidson. Transcriptional regulatory cascades in development: Initial rates, not steady state, determine network kinetics. *Proceedings of the National Academy of Sciences of the United States of America*, 100(16):9371–9376, August 5 2003.

[5] N. Corporation. NVIDIA CUDA Toolkit v3.2 RC2 Release Notes for Linux, October 2010.

[6] S. Das, D. Caragea, S. M. Welch, and W. H. Hsu, editors. *Handbook of Research on Computational Methodologies in Gene Regulatory Networks*. Medical Information Science Reference, USA, 2009.

[7] C. Davidson. Identifying gene regulatory networks using evolutionary algorithms. *Journal of Computing Sciences in Colleges*, 25(5):231–237, May 2010.

[8] E. H. Davidson. *The Regulatory Genome: Gene Regulatory Networks In Development And Evolution*. Academic Press, London, UK, 2006. ISBN 978-0-12-088563-3.

[9] L. de P. Veronese and R. A. Krohling. Differential evolution algorithm on the GPU with C-CUDA. In *2010 IEEE Congress on Evolutionary Computation (CEC'2010)*, Barcelona, Spain, July 2010. IEEE Press.

[10] A. Eiben and J. Smith. *Introduction to Evolutionary Computing*. Springer, Berlin, 2003. ISBN 3-540-40184-9.

[11] L. J. Fogel. *Artificial Intelligence through Simulated Evolution*. John Wiley, New York, 1966.

[12] D. E. Goldberg. *Genetic Algorithms in Search, Optimization and Machine Learning*. Addison-Wesley Publishing Company, Reading, Massachusetts, 1989.

[13] S. Harding and W. Banzhaf. Fast Genetic Programming on GPUs. In M. Ebner, M. O'Neill, A. Ekárt, L. Vanneschi, and A. I. Esparcia-Alcázar, editors, *Genetic Programming, 10th European Conference, EuroGP 2007*, pages 90–101. Springer, Lecture Notes in Computer Science Vol. 4445, Valencia, Spain, April 2007.

[14] J. Hasty, D. McMillen, F. Isaacs, and J. Collins. Computational studies of gene regulatory networks: In numero molecular biology. *Nature Reviews Genetics*, 2(4):268–279, April 2001.

[15] S. Kimura, M. Hatakeyama, and A. Konagaya. Inference of s-system models of genetic networks using a genetic local search. In *The 2003 IEEE Congress on Evolutionary Computation (CEC'2003)*, pages 631–638, Canberra, Australia, December 2003. IEEE Press.

[16] P. Koduru, S. Das, S. Welch, and J. L. Roe. Fuzzy Dominance Based Multi-objective GA-Simplex Hybrid Algorithms Applied to Gene Network Models. In K. D. et al., editor, *Genetic and Evolutionary Computation–GECCO 2004. Proceedings of the Genetic and Evolutionary Computation Conference. Part I*, pages 356–367, Seattle, Washington, USA, June 2004. Springer-Verlag, Lecture Notes in Computer Science Vol. 3102.

[17] P. Koduru, S. Das, S. M. Welch, J. L. Roe, and E. Charbit. A Multiobjective Evolutionary-Simplex

Number of individuals	Max Generations	CPU time in seconds	CPU fitness	GPU time in seconds	GPU fitness	Speedup
32	100	0.68	4.39158	0.64	5.03668	×1.0625
64	100	1.36	5.17165	0.63	4.39576	×2.1587301587
128	100	2.66	5.23209	0.62	3.50573	×4.2903225806
256	100	5.33	5.17518	0.63	5.09926	×8.4603174603
512	100	10.71	5.15556	0.66	5.20999	×16.2272727273
1024	100	21.56	4.45661	0.65	4.78358	×33.1692307692
2048	100	42.72	4.15864	0.65	4.42818	×65.7230769231
4056	100	84.63	2.72947	0.66	4.837	×128.2272727273
32	500	3.53	1.20813	3.07	0.1357531	×1.1498371336
64	500	7.09	0.0238907	3.14	0.0110196	×2.2579617834
128	500	13.97	0.0242551	3.15	0.0172489	×4.4349206349
256	500	27.86	0.0127262	3.17	0.00386007	×8.7886435331
512	500	55.6	0.0175084	3.12	0.00935788	×17.8205128205
1024	500	110.86	0.0200963	3.18	0.00658496	×34.8616352201
2048	500	234.31	0.0184026	3.14	0.0043593	×74.6210191083
4056	500	438.5	0.0143981	3.16	0.00700156	×138.7658227848
32	1000	7.32	1.89028	6.24	0.000593101	×1.1730769231
64	1000	14.64	0.003957	6.28	0.000440774	×2.3312101911
128	1000	29.31	2.48775e-05	6.3	0.000226659	×4.6523809524
256	1000	58.21	2.82812e-05	6.29	8.89689e-05	×9.2543720191
512	1000	116.57	2.84008e-05	6.36	8.85665e-05	×18.3286163522
1024	1000	232.72	3.69976e-05	6.29	1.54926e-04	×36.9984101749
2048	1000	463.56	6.2814e-05	6.31	8.13485e-05	×73.4643423138
4056	1000	919.65	5.3545e-05	6.35	8.016232e-05	×144.8267716535

Table 2: Comparison of results and computational times (in second) between our GPU-based version and our CPU version.

Hybrid Approach for the Optimization of Differential Equation Models of Gene Networks. *IEEE Transactions on Evolutionary Computation*, 12(5):572–590, October 2008.

[18] N. Noman and H. Iba. Inference of gene regulatory networks using s-system and differential evolution. In *Proceedings of the 2005 Conference on Genetic and Evolutionary Computation (GECCO'2005)*, pages 439–446, New York, NY, USA, 2005. ACM Press.

[19] N. Noman and H. Iba. Inferring Gene Regulatory Networks using Differential Evolution with Local Search Heuristics. *IEEE/ACM Transactions on Computational Biology and Bioinformatics*, 4(4):634–647, October-December 2007.

[20] K. V. Price, R. M. Storn, and J. A. Lampinen. *Differential Evolution. A Practical Approach to Global Optimization*. Springer, Berlin, 2005. ISBN 3-540-20950-6.

[21] M. Savageau. Introduction to s-systems and the underlying power-law formalism. *Mathematical and Computer Modelling*, 11:546–551, 1988.

[22] T. Schlitt and A. Brazma. Current approaches to gene regulatory network modelling. *BMC Bioinformatics*, 8(6), 2007. Article Number S9.

[23] H.-P. Schwefel. *Evolution and Optimum Seeking*. John Wiley & Sons, New York, 1995.

[24] D. Searson, M. Willis, S. Horne, and A. Wright. S-systems and evolutionary algorithms for the inference of chemical reaction networks from fed-batch reactor experiments. In *Proceedings of the 17th International Congress of Chemical and Process Engineering (CHISA 2006)*, Prague, Czech Republic, 27-31 August 2006.

[25] C. Spieth, F. Streichert, N. Speer, and A. Zell. Optimizing Topology and Parameters of Gene Regulatory Network Models from Time-Series Experiments. In *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO 2004)*, pages 461–470. Springer, Lecture Notes in Computer Science Vol. 3102, 2004.

[26] C. Spieth, F. Streichert, N. Speer, and A. Zell. Multi-objective Model Optimization for Inferring Gene Regulatory Networks. In C. A. Coello Coello, A. Hernández Aguirre, and E. Zitzler, editors, *Evolutionary Multi-Criterion Optimization. Third International Conference, EMO 2005*, pages 607–620, Guanajuato, México, March 2005. Springer. Lecture Notes in Computer Science Vol. 3410.

[27] R. Storn and K. Price. Differential Evolution - A Fast and Efficient Heuristic for Global Optimization over Continuous Spaces. *Journal of Global Optimization*, 11:341–359, 1997.

[28] D. Tominaga, M. Okamoto, Y. Maki, S. Watanabe, and Y. Eguchi. Nonlinear Numerical Optimization Technique Based on a Genetic Algorithm for Inverse Problems: Towards the Inference of Genetic Networks. In *German Conference on Bioinformatics'99*, pages 127–140, 1999.

[29] M.-L. Wong, T.-T. Wong, and K.-L. Fok. Parallel evolutionary algorithms on graphics processing unit. In *2005 IEEE Congress on Evolutionary Computation (CEC'2005)*, volume 3, pages 2286–2293, Edinburgh, Scotland, September 2005. IEEE Service Center.