High performance computing architectures analysis for gene networks inference

Anderson G. Marco[1] **, Mario A. Gazziro**[2] **, David C. Martins-Jr**[1]

[1]Centro de Matemática Computação e Cognição - UFABC

[2]Centro de Engenharia e Ciências Sociais - UFABC
Av dos Estados – 5001 – 09210-580 – Santo André – SP – Brasil

anderson.marco@ufabc.edu.br, mario.gazziro@ufabc.edu.br,

david.martins@ufabc.edu.br

Abstract. *Modeling and inference of biological systems are an important field in computer science, presenting strong interdisciplinary aspects. In this context, the inference of gene regulatory networks and the analysis of their dynamics generated by their transition functions are important issues that demand substantial computational power. Because the algorithms that return the optimal solution have an exponential time cost, such algorithms only work for gene networks with only dozens of genes. However realistic gene networks present hundreds to thousands of genes, with some genes being hubs, i.e., their number of predictor genes are usually much higher than average. Therefore there is a need to develop ways to speed up the gene networks inference. This paper presents a benchmark involving* GPUs *and* FPGAs *to infer gene networks, analysing processing time, hardware cost acquisition, energy consumption and programming complexity. Overall Titan XP GPU achieved the best performance, but with a large cost regarding acquisition price when compared to R9 Nano GPU and DE1-SOC FPGA. In its turn, R9 Nano GPU presented the best cost-benefit regarding performance, acquisition price, energy consumption, and programming complexity, although DE1-SOC FPGA presented much smaller energy consumption.*

Resumo. *A modelagem e inferência de sistemas biológicos é uma importante área de pesquisa em ciência da computação, possuindo características fortemente interdisciplinares. Nesse contexto, a inferência de redes de regulação gênica e a análise da dinâmica da expressão gerada pelas suas funções de transição são problemas relevantes. Tais problemas demandam muito poder computacional, já que algoritmos que buscam pela solução ótima possuem complexidade de tempo exponencial. Esta complexidade de tempo faz com que muitas análises sejam realizadas em redes gênicas com dezenas de genes e com cada gene tendo poucos preditores. Porém, uma rede gênica de tamanho real tipicamente possui milhares de genes, com alguns desses genes podendo ser hubs por terem um grau de entrada (número de preditores) bem acima da média. Desta forma é necessário buscar meios de acelerar o processamento da inferência de redes gênicas. Este artigo mostra uma comparação entre* GPUs *e* FPGAs*, duas arquiteturas computacionais de alto desempenho, na realização*

da tarefa de inferência de redes gênicas, comparando o tempo para o processamento, o custo energético, custo de aquisição destes hardwares e dificuldade de programação.

1. Introduction

Genes are responsible to define the features of the life forms, together with the environment. The expression of a given gene, which is understood as the concentration of RNA messengers (mRNA) created for a specific condition, can impact the expression of other genes. The interaction among genes by means of their expression can be viewed as a graph. Such graphs are known as gene regulatory networks. Gene networks inference is important to derive models to better understand the interaction among genes. Bayesian networks [Friedman et al. 2000] and its variations (e.g., Boolean networks [Kauffman 1969], probabilistic Boolean networks [Shmulevich et al. 2002] and probabilistic gene networks [Barrera et al. 2007] [Lopes et al. 2008]) are popular gene network discrete models, i.e., each gene is represented by a finite number of states (e.g., 0 or 1 in Boolean Network models and their derivatives). The inference of a Bayesian network requires the estimation of a conditional probability distribution table for each gene, representing the probabilities of a gene being on each possible state given all possible states of the best candidate gene set (predictors). Such probabilities are estimated from a given gene expression dataset. Bayesian networks inference involves the search for the best predictor set of a given gene guided by a criterion function (or fitness function). The criterion function measures the quality of prediction of the estimated conditional probability tables. The inference of gene network modeled as Bayesian networks is considered *NP*-complete with a factorial computational cost [Chickering 1996]. High performance computing methods involving parallel computing have been developed to increase the performance of this task [Borelli et al. 2013, Carastan-Santos et al. 2017].

According to [Vanderbauwhede and Benkrid 2013], since the beginning of the 21st century Field Programmable Gate Array (*FPGA*) has been used for parallel processing in many cases. This device is not a processor but a "programmable" chip to represent any entity of digital logics, being able to represent even processors. The logic circuit implemented in *FPGA* shows the type of parallel processing to be done. Therefore, the *FPGA* can do parallel processing of either shared memory or distributed memory. Pournara *et al* [Pournara et al. 2005] were the first to use *FPGA* to infer gene networks, however their implementation is restricted for networks with dozens of genes.

Along with *FPGAs GPUs* (*Graphics Processing Unit*) are very used in high performance computing. *GPUs* are formed by many small processors specialized floating point operations. *GPUs* are more easy to program, more popular and have a computational power equivalent to *FPGAs* for applications with an extensive use of floating/fixed point operations. The main tools for *GPU* programming are *CUDA* [Cook 2018] library (Nvidia proprietary), and the libraries that implement *OpenCL* [Munshi et al. 2018] *API* (Application Programming Interface). Borelli *et al* [Borelli et al. 2013] work is one of the first to use *GPU* to infer gene networks with realistic size, although the topology of the resulting networks is restricted to a maximum of two predictors per gene.

This paper aims to analyse the performances of *FPGAs* and *GPUs* in inferring gene networks. Differently from other works, whose networks inferred are not realis-

tic, the methods described in this paper can infer networks with realistic sizes and more complex topologies. Performance analyses involve cost of hardware acquisition, energy consumption and difficulty of programming, besides execution time.

2. Exhaustive search algorithm for gene networks inference

The exhaustive search for gene networks inference described here receives an input matrix $U_{m \times n}$ where columns correspond to genes and rows correspond to temporal gene expression samples, a fixed predictor set size s, and a criterion function $\mathcal{F}$. By adopting Boolean Networks as gene networks model, each expression value is binary, i,e,, $U(i,j) = \{0,1\}, i = 1..m, j = 1..n$, where 0 means sub-expression (inactive or below average activity) and 1 means super-expression (active or above average activity). Normally there are thousands of genes and only dozens of samples ($m << n$). Gene networks modeled as Boolean Networks can be understood as dynamical systems where the variables and time evolution are discrete, and variables correspond to genes. The algorithm is described as follows:

1. For each gene $Y \in \mathbf{X}$, where $\mathbf{X} = \{X_1, X_2, ..., X_n\} \in \{0,1\}^n$ is the set of all n genes, perform an exhaustive search as described in step 2.
2. For each possible set of predictors $\mathbf{Z} \subseteq \mathbf{X}$ with a given fixed size s, estimate a conditional probability table for Y values given the possible values of $\mathbf{Z}$ based on U and evaluate it by applying the given criterion function $\mathcal{F}$.
3. Return the set of predictors $\mathbf{Z}^* \subseteq \mathbf{X}$ which performs the best prediction of Y according to the given criterion function $\mathcal{F}$.

In the sequential implementation all genes are processed at an outer loop (step 1), while the inner loop (step 2) creates and evaluates the conditional probability tables of all possible candidate predictor sets with a given fixed size regarding the prediction of a gene referred at a given iteration of the outer loop.

In order to retrieve the conditional probabilities table, it requires the creation of a matrix T with 2^s rows (number of all possible values of a candidate predictor set $\mathbf{Z}$) and two columns (number of possible values of gene Y: 0 and 1), Each matrix cell stores the number of observations of a certain value of $Y = \{0,1\}$ given a certain instance of $\mathbf{Z} = \{0,1\}^s$. Equations 1 and 2 define the creation of the matrix T for a given candidate predictor set $\{g_1, g_2, ..., g_s\} \subseteq \mathbf{X}$ and a given target gene $g_0 \in \mathbf{X}$. These equations use auxiliary functions defined in Equations 3, 4 and 5.

$$T_{j,1} = \sum_{i=1}^{m} v(i, j, g_1, g_2, ..., g_s, g_0) \tag{1}$$

$$T_{j,2} = \sum_{i=1}^{m} w(i, j, g_1, g_2, ..., g_s, g_0) \tag{2}$$

$$v(i, j, g_1, g_2, ..., g_s, g0) = \begin{cases} 1 & , q(i, g_1, g_2, ..., g_s) = j \\ & \text{and } U_{(i \bmod m)+1, g0} = 1 \\ 0 & \text{, otherwise} \end{cases} \tag{3}$$

$$w(i, j, g_1, g_2, ..., g_s, g0) = \begin{cases} 1 & , q(i, g_1, g_2, ..., g_s) = j \\ & \text{and } U_{(i \bmod m)+1, g0} = 0 \\ 0 & \text{, otherwise} \end{cases} \tag{4}$$

$$q(i, g1, g2, ..., gs) = (2^{s-1}U_{i,g1} + 2^{s-2}U_{i,g2} + ... + U_{i,gs}) + 1 \quad (5)$$

Each row of matrix T refers to a certain state that the gene set $\{g_1, g_2, ..., g_s\}$ could have and the two columns refer to the values (0 or 1) that the target gene (g_0) could have. The function in Equation 5 shows the state of genes from the considered candidate predictor set at time t, with 2^s possible states indexed from 1 to 2^s. The expression value $U_{(i \bmod m)+1,g_0}$, present in Equations 3 and 4 represents the state of the target gene g_0 at the next timepoint $(t + 1)$, for $t = 1..m - 1$. In this way T represents a counting table for all possible states of the candidate predictors and the target, which can be easily converted to a conditional probability distribution table.

The counting table T can be evaluated by many criterion functions. A popularly adopted criterion function is the conditional entropy [Barrera et al. 2007, Lopes et al. 2008], in which the best predictor sets present the smallest mean conditional entropy values. Conditional entropy presents a high computational cost, since it requires the conversion of the counting table to a conditional probability table and mathematical operations with real numbers (floating point). Due to this issue, in this paper we adopted a less complex criterion function based on Bayesian classification error. The coefficient of determination (CoD) [Martins-Jr et al. 2008, Dougherty et al. 2009], also popularly used for gene networks inference, relies on Bayesian error. The Bayesian error is defined by function $E(T)$ in Equation 6.

$$E(T) = \sum_{i=1}^{m} \alpha(T, i) \quad (6)$$

$$\alpha(T, i) = \begin{cases} T_{i,1} & , T_{i,1} < T_{i,2} \\ T_{i,2} & , T_{i,1} \geq T_{i,2} \end{cases} \quad (7)$$

Once T is obtained, the Boolean function b that minimizes the classification error based on T is defined as a vector where each value corresponds to the target gene value with the smallest error for a given row of T, i.e., $b_i = 0$ if $T_{i,1} < T_{i,2}$ or $b_i = 1$ otherwise, for $i = 1..2^s$.

2.1. High performance computing implementations

In this paper we propose three implementations for the gene network inference algorithm described in the previous section, one version uses *FPGA* and the other two use *GPUs*. One *GPU* implementation uses *CUDA* library, while the other uses *OpenCL API*. *CUDA* only works with Nvidia *GPUs* while *OpenCL* has implementations for AMD *GPUs*, Nvidia *GPUs*, x86 *CPUs* and other accelerator hardwares. To analyse the performances of both *OpenCL* and *CUDA* for Nvidia *GPUs*, we developed two *GPU* implementation versions. Both *GPU* implementations require parallel programming to present good performance.

2.1.1. Parallelism using GPU

The parallelism in *GPU* can be summarized as the parallelization of the outer and the inner loop, These loops are described in Section 2. GPU code presents high performance only

if it has thousands of processes/threads running at the same time, *CUDA* and *OpenCL* organize *threads* by blocks. The *CUDA* Guide [Cook 2018] states that a good performance is obtained when the number of blocks is greater or equal than 100 and the number of threads is divisible by 64. Therefore if only the external loop is parallelized, the gene network needs to have 6400 genes in order to achieve good performance. If the two loops are parallelized, it is possible to achieve total use of GPU in a network with at least 100 genes.

Nvidia provides a spreadsheet [1] which informs how much a code for its *GPUs* is optimized (occupancy). The occupancy depends on the number of threads per block of the GPU model adopted and on the number of registers used by the code. The number of registers is known when the code is compiled with *–ptxas-options=-v* parameter. This parameter works only in *nvcc* (an Nvidia *CUDA* compiler). Our developed code to infer gene networks presents 75% of occupancy, which is considered an excellent result.

2.1.2. Parallelism using *FPGA*

The *FPGA* parallel implementation version can be summarized as the parallelization of the outer loop, the counting table T construction, and the Boolean function creation based on T and the respective Bayesian classification errors. These operations were parallelized for *FPGA* but not for *GPU*, because they require much communication, and the communication cost among processes/*threads* is smaller in *FPGAs* than in *GPUs*.

The inner loop was not parallelized in *FPGA* due to the memory communication high cost. The *FPGA* memories speed is limited to hundreds of megahertz, slower than *GPU* memories which have between 1.2 and 2.6 gigahertz. Therefore the *FPGAs* memories are a barrier for the high performance parallelism. Another reason for the absence of parallelism of the inner loop refers to the fact that *FPGA* does not support the execution of a large number of *threads*/process. The number of *threads*/processes supported by *FPGA* can support is limited by code complexity from threads/processes. *FPGAs* do not support many thread/process instances with a lot of variables and complex operations, such as logarithm operations.

The *FPGA* implementation required the creation of a communication protocol between *FPGA* and an *ARM CPU*, both in the same die. This protocol has three layers with different abstract levels, as illustrated in the diagram of Figure 1. The layer in the low level is implemented over *PIOs*, a bus communication more popular for Altera *FPGAs*. The middle layer is composed by instructions to read and write data in a *RAM* implemented over *HDL* (Hardware Description Language). The top layer is composed by data from *RAM* which was described in the middle layer. An *HDL* code is responsible to infer gene networks from data received and sent by the top layer.

3. Analyses

As described in the previous section, three parallel implementations for inference of gene networks were considered:

1. *CUDA* (only for NVIDIA *GPUs*)

[1] http://developer.download.nvidia.com/compute/cuda/CUDA_Occupancy_calculator.xls

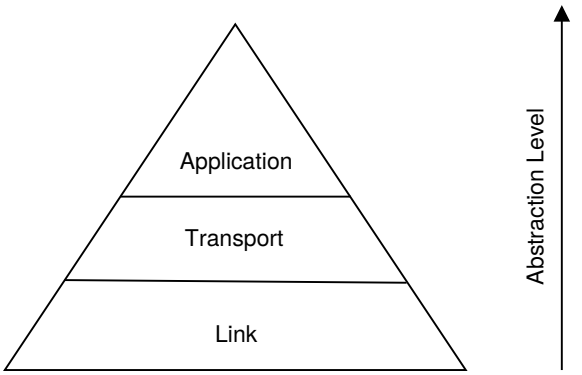


Figure 1. Diagram showing the layers of the protocol created for communication between FPGA and ARM processor, which resides in the same FPGA encapsulation.

2. *OpenCL* (for GPUs, CPUs x86 and other hardware accelerators)
3. *HDL* Verilog/System Verilog: this implementation instances a circuit on *FPGAs*, whose code presents some small parts written in C++.

We conducted the analyses taking into account an artificial gene network with 512 genes ($n = 512$). Artificial gene expression data containing 50 temporal samples was generated considering the transition functions of the artificial gene network. The number of predictors per target was fixed as 3 ($s = 3$) for the exhaustive search.

The hardwares used for the analysis are described as follows:

1. *PC* Intel Core i7-7700K CPU, 16 GB *RAM*, Nvidia *GPU* Titan XP, Linux *Kernel* 3.10.0-862.3.2.el7.x86_64, Nvidia *driver* 390.67 and *CUDA* 9.1.85.
2. *PC* AMD A10-7850K, 8 GB *RAM*, *GPU* AMD R9 Nano, Windows 10 version 1803, *GPU* driver 17.7 and AMD-APP-SDK version 3.0.130.135-GA.
3. Terasic DE1-SOC Board with Cyclone V SoC 5CSEMA5F31C6 *FPGA*, in the same die with an ARM Cortex A9 processor, 1 GB *RAM* for ARM and Linux Kernel version 4.5.0-00183-g4647b69-dirty running at ARM. This board works without *PC*, as an Arduino board.

The *CUDA* implementation was analyzed on *hardware* 1 only. *OpenCL* implementation was analyzed on *hardwares* 1 and 2, For *hardware* 2 two analyses were conducted, one for *GPU* implementation and another for the CPU implementation. *FPGA* implementation was analyzed on *hardware* 3 only.

3.1. Implementations code complexity

Lines counting was adopted as measure of implementation complexity. Lines of code for layers linking and transport in the communication protocol showed in Section 2.1.2 were ignored from counting, since those lines might be reusable by other projects.

The line counts for the implementations were 379, 416 and 2150 for *CUDA*, *OpenCL* Verilog/System Verilog, respectively. Therefore *FPGA* implementation is the most complex implementation. Moreover, the communication protocol between *FPGA* and *ARM* processor has 1446 lines, which illustrates the high complexity in implementing communication protocols between FPGA and external devices.

3.2. Execution time

The time were measured from five configuration setups, every configuration setup is composed of three component: hardware, implementation and the hardware section where the implementation was running (*GPU*, *CPU* or *FPGA*). The hardwares and the implementations were enumerated at the beginning of this section. There are five configurations setups:

- *PC* with Titan XP *GPU* (hardware 1) and *CUDA* implementation running at *GPU*.
- *PC* with R9 Nano *GPU* (hardware 2) and *OpenCL* implementation running at *GPU*.
- *PC* with Titan XP *GPU* (hardware 1) and *OpenCL* implementation running at *GPU*.
- *PC* with R9 Nano *GPU* (hardware 2) and *OpenCL* implementation running at AMD A10-7850K *CPU*.
- Terasic DE1-SOC Board (hardware 3) and *HDL* implementation running at *FPGA*.

Each configuration setups was executed ten times. Figure 2 shows the average execution time achieved for the 5 configuration setups.

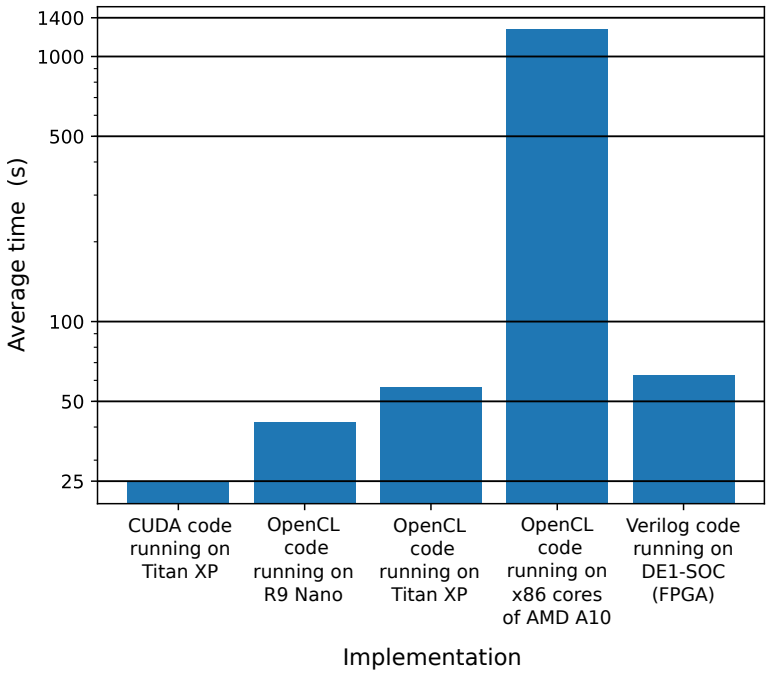


Figure 2. Average time for each considered hardware to infer gene networks (log scale). The averages were obtained from ten executions. The standard deviations were negligible (maximum of 1.5 seconds).

3.3. Energy consumption

We used a clamp meter for measuring the energy consumption from each hardware, including the auxiliary hardware necessary to work. Auxiliary hardware can be a power supply or a *CPU*. Therefore the energy consumption from each hardware was measured, except for the hardware 1 (Titan XP *GPU*) because we did not have physical access to this hardware.

There are some details about the energy consumption from each hardware:

- The energy drained from DE1-SOC board does not change in case the board is processing a gene network or the board is on standby.
- There are variations for the amount of energy drained from R9 Nano when this hardware is running the program to infer a gene network.
- The energy drained from AMD A10 changes if the hardware changes from standby to processing a gene network or if the hardware changes from processing a gene network to standby; However while the hardware is running the program to infer a gene network the energy drained does not change.

The energy consumption from DE1-SOC is close to 22 watts (Figure 3-a). The energy consumption from *OpenCL* parallel code running on AMD A10 *x86* cores is close to 118 watts (Figure 3-b). Due to changes for the amount of energy drained from R9 Nano along time (Figure 3-c), it was not possible to infer an average with small standard deviation for the energy consumption.

R9 Nano has an irregular amount of energy drained along time due to threads synchronization inside a block. The threads are divided inside blocks and each block is responsible to find the better predictor genes set for a gene. This happens for *OpenCL* and *CUDA* implementations, Next, the threads finish the processing, and the results obtained from them are shared to know which result is the best. The step which returns the best result is a sequential process. Therefore it happens for only one thread per block, while the other threads are in standby mode. In this way, R9 is not fully loaded during the final step of the algorithm, which leads to smaller energy consumption.

4. Cost of hardware acquisition

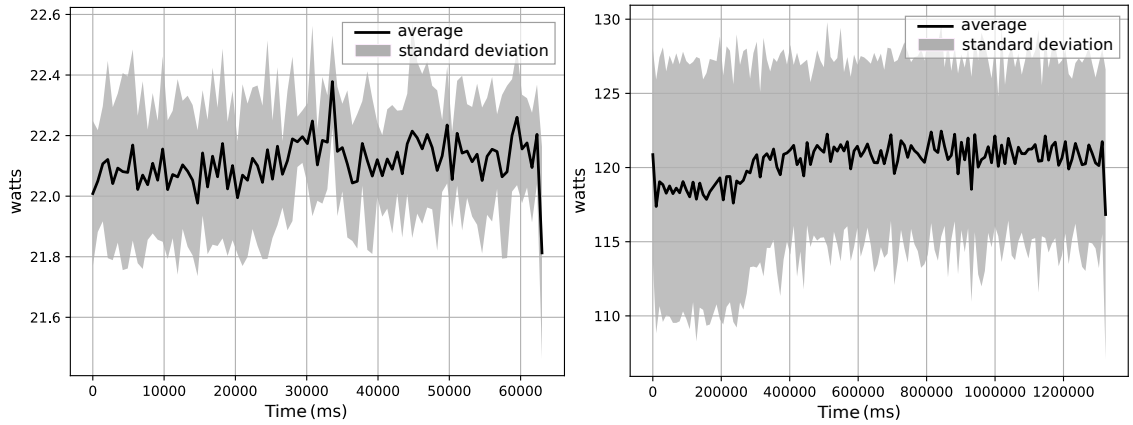
Both R9 Nano and Titan XP are hardwares that needs to be embedded in a personal computer (*PC*) to work, so we performed price searches for PCs able to support both *GPUs*. For the cost quotation of the *PC* parts and the cost of a GPU equivalent to R9 Nano, we based the search on https://www.outletpc.com/, since the R9 Nano was discontinued. Table 1 shows the price of each component required to build a PC that supports GPU R9 Nano or GPU Titan XP.

Table 1. Price in dollars of the components needed for a computer that supports an R9 Nano *GPU* or an Nvidia Titan XP *GPU*.

Component	Price
Intel Celeron processor G3900	36.79
DDR4 memory 8 GB	89.89
GIGABYTE Motherboard GA-H270M-DS3H	119.0
Corsair power supply 1000W ATX12V 80 PLUS	174.39
Thermaltake computer chassis CA-1D4-00S1NN-00	44.89
SSD SanDisk PLUS 2.5” 120GB SATA III	59.89

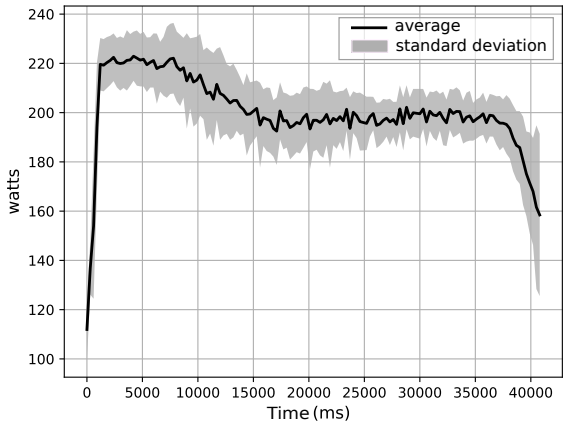
According to Tomshardware website the *GPU* Nvidia Titan XP price is 1200 dollars [2]. The price of a R9 Nano was 649 dollars when it was launched, however it is currently sold out. A GPU with similar performance nowadays is RX580, based on an

[2]https://www.tomshardware.com/reviews/nvidia-titan-xp,5066.html



(a) FPGA DE1-SOC

(b) OpenCL x86 AMD A10



(c) GPU R9 Nano

Figure 3. Plots with the average energy consumption along time for the three hardwares analyzed. Ten measurements were done for each time instant.

analysis done by UserBenchmark website [3]. Accoording to https://www.outletpc.com/ website the RX580 price is 300 dollars.

The AMD A10 processor price is 170 dollars according to https://www.outletpc.com/ website, Table 2 presents the cost of the components required to build a computer that supports the AMD A10 processor.

Table 2. Price in dollars of the components needed to build a computer that supports an AMD A10 processor.

Component	Price
DDR3 memory 8 GB	75.89
MSI motherboard A68HM AMD A68 Chipset	109.88
Thermaltake power supply Smart Series 700W	58.66
Thermaltake computer chassis CA-1D4-00S1NN-00	44.89
SSD SanDisk PLUS 2.5” 120GB SATA III	59.89

[3]http://gpu.userbenchmark.com/Compare/AMD-R9-Nano-vs-AMD-RX-580/m58413vs3923

The DE1-SOC board costs 249 dollars for the commercial version according to Terasic website. The academic version of this board, which has the same technical specifications, costs 175 dollars. DE10-Nano is a board equivalent to DE1-SOC. Its price is 110 dollars for academic version and 130 dollars for commercial version. DE10-Nano would likely work running the *FPGA* code analyzed in this paper, because both *FPGAs* have similar technical specifications.

5. Conclusion

In this paper we preseted a detailed comparison among different hardwares to infer gene networks. The experiments compared the average hardware performances for inferring gene networks with 512 genes, where 3 predictors were inferred for each gene based on a set of 50 temporal expression samples. Three exhaustive search algorithm implementations were developed, one of them with CUDA library, another one with OpenCL API and the third one with HDL Verilog/System Verilog. The CUDA implementation was done to analyse the maximum performance achieved by the algorithm on GPU Nvidia Titan XP. In its turn, the OpenCL implementation was done to measure the algorithm performance running on CPUs and R9 Nano GPU. Besides we conducted a comparison of performances between OpenCL and CUDA for Nvidia GPUs, Finally, the Verilog/System Verilog implementation consists on a digital circuit description instantiated on FPGA of DEI-SOC board. For the price analyses of GPUs we also considered the price of the personal computer (PC) necessary to function.

The results presented show that, if ignored the energy consumption, the best hardware in terms of cost-benefit for gene networks inference is GPU R9 Nano or the equivalent RX500. R9 Nano is about 50% faster than DE-1 SOC board, while the equivalent RX500 is 182% more expensive than DE-1 SOC in its commercial version and the OpenCL code implementation complexity is about 80% less than the DE1-SOC code implementation.

Nvidia Titan XP hardware achieved the best performance: 66% faster than R9 Nano. Nevertheless, Nvidia Titan XP is 109% more expensive than the GPU equivalent to R9 Nano. It is important to highlight that Titan XP only achieved a superior performance when running the CUDA implementation, since the OpenCL implementation performance was 26% worse. In terms of code implementation complexity, both were almost equivalent, with OpenCL implementation being only about 10% more complex than CUDA implementation.

The x86 cores of the AMD A10 processor are inadequate to infer gene networks, since its performance was 20 times worse than DE1-SOC, with an energy consumption of about half of the maximum consumption of R9 Nano.

From the point of view of the energy consumption the best hardware is DE1-SOC which consumes about 9 times less than the R9 Nano average consumption. In terms of acquisition cost, DE1-SOC is better than R9 Nano only if it is necessary to buy a new computer for R9 Nano/RX 580. DE1-SOC has a programming complexity much larger than R9 Nano/RX 580. Therefore DE1-SOC is feasible only if the gene networks inference algorithm runs for long periods of time (in the order of months). In such a scenario the reduced energy consumption compensates the development time and a possibly larger acquisition cost compared to R9 Nano/RX 580.

For future works, the construction of a hybrid GPU/FPGA method for gene networks inference can be considered, as well as the conduction of performance analyses of the architectures discussed here for methods of generation of expression dynamics by the inferred gene networks. The analysis and control of the gene network expression dynamics is one of the important goals in systems biology, since it is fundamental for understanding the genesis and development of diseases.

6. Supplementary material

The implementations and information related to this manuscript will be available at https://hpcgenenetworksinference.wordpress.com/ in September 2019.

Acknowledgements

This study was financed in part by CAPES (Finance Code 001), CNPq and FAPESP (proc. #2015/01587-0).

References

[Barrera et al. 2007] Barrera, J., Cesar-Jr, R. M., Martins-Jr, D. C., Vencio, R. Z. N., Merino, E. F., Yamamoto, M. M., Leonardi, F. G., Pereira, C. A. B., and del Portillo, H. A. (2007). Constructing probabilistic genetic networks of *Plasmodium falciparum* from dynamical expression signals of the intraerythrocytic development cycle. In *Methods of Microarray Data Analysis V*, chapter 2, pages 11–26. Springer.

[Borelli et al. 2013] Borelli, F. F., de Camargo, R. Y., Martins-Jr, D. C., and Rozante, L. C. S. (2013). Gene regulatory networks inference using a multi-gpu exhaustive search algorithm. *BMC Bioinformatics*, 14(S5).

[Carastan-Santos et al. 2017] Carastan-Santos, D., Camargo, R. Y., Martins-Jr, D. C., Song, S. W., and Rozante, L. C. S. (2017). Finding exact hitting set solutions for systems biology applications using heterogeneous gpu clusters. *Future Generation Computer Systems*, 67:418–429.

[Chickering 1996] Chickering, D. M. (1996). *Learning Bayesian Networks is NP-Complete*, pages 121–130. Springer New York, New York, NY.

[Cook 2018] Cook, S. (2018). *CUDA Programming*.

[Dougherty et al. 2009] Dougherty, E. R., Brun, M., Trent, J., and Bittner, M. L. (2009). A conditioning-based model of contextual regulation. *IEEE/ACM Transactions on Computational Biology and Bioinformatics*, 6(2):310–320.

[Friedman et al. 2000] Friedman, N., Linial, M., Nachman, I., and Pe'er, D. (2000). Using Bayesian Network to Analyze Expression Data. *Journal of Computational Biology*, 7:601–620.

[Kauffman 1969] Kauffman, S. A. (1969). Metabolic stability and epigenesis in randomly constructed genetic nets. *Journal of Theoretical Biology*, 22(3):437–467.

[Lopes et al. 2008] Lopes, F. M., Martins-Jr, D. C., and Cesar-Jr, R. M. (2008). Feature selection environment for genomic applications. *BMC Bioinformatics*, 9(1):451.

[Martins-Jr et al. 2008] Martins-Jr, D. C., Braga-Neto, U., Hashimoto, R. F., Dougherty, E. R., and Bittner, M. L. (2008). Intrinsically multivariate predictive genes. *IEEE Journal of Selected Topics in Signal Processing*, 2(3):424–439.

[Munshi et al. 2018] Munshi, A., Gaster, B., Mattson, T. G., Fung, J., and Ginsburg, D. (2018). *OpenCL Programming Guide*.

[Pournara et al. 2005] Pournara, I., s. Bouganis, C., and Constantinides, G. A. (2005). Fpga-accelerated bayesian learning for reconstruction of gene regulatory networks. In *International Conference on Field Programmable Logic and Applications, 2005.*, pages 323–328.

[Shmulevich et al. 2002] Shmulevich, I., Dougherty, E. R., Kim, S., and Zhang, W. (2002). Probabilistic boolean networks: a rule-based uncertainty model for gene regulatory networks. *Bioinformatics*, 18(2):261–274.

[Vanderbauwhede and Benkrid 2013] Vanderbauwhede, W. and Benkrid, K. (2013). *High-Performance Computing Using FPGAs*. Springer Publishing Company, Incorporated.