High Performance and Scalable Simulations of a Bio-inspired Computational Model

Sandra Gómez-Canaval
Depto. de Sistemas Informáticos
Universidad Politécnica de Madrid,
Madrid, Spain
sgomez@etsisi.upm.es

Victor Mitrana
Depto. de Sistemas Informáticos
Universidad Politécnica de Madrid,
Madrid, Spain
victor.mitrana@upm.es

Mihaela Păun
National Institute for Research and
Development of Biological Sciences
Bucharest, Romania
mihaela.paun@incdsb.ro

Stanislav Vararuk
Research Group of Modelling and
Mathematical Biocomputation
Universidad Politécnica de Madrid, Madrid, Spain
stanislav.vakaruk@alumnos.upm.es

***Abstract*—The Network of Polarized Evolutionary Processors (NPEP) is a rather new variant of the bio-inspired computing model called Network of Evolutionary Processors (NEP). This model, together with its variants, is able to provide theoretical feasible solutions to hard computational problems. NPEPE is a software engine able to simulate NPEP which is deployed over Giraph, an ultra-scalable platform based on the Bulk Synchronous Parallel (BSP) programming model. Rather surprisingly, the BSP model and the underlying architecture of NPEP have many common points. Moreover, these similarities are also shared with all variants in the NEP family. We take advantage of these similarities and propose an extension of NPEPE (named gNEP) that enhances it to simulate any variant of the NEP's family. Our extended gNEP framework, presents a twofold contribution. Firstly, a flexible architecture able to extend software components in order to include other NEP models (including the seminal NEP model and new ones). Secondly, a component able to translate input configuration files representing the instance of a problem and an algorithm based on different variants of the NEP model into some suitable input files for gNEP framework. In this work, we simulate a solution to the "3-colorability" problem which is based on NPEP. We compare the results for a specific experiment using NPEPE engine and gNEP. Moreover, we show several experiments in the aim of studying, in a preliminary way, the scalability offered by gNEP to easily deploy and execute instances of problems requiring more intensive computations.**

***Index Terms*—Bio-inspired computational model, Networks of Polarized Evolutionary Processors, Bulk Synchronous Parallel programming model, Distributed architectures, Parallel computation.**

I. Introduction

A vividly studied branch of Natural Computing investigates computational techniques, models of computation and computational devices inspired by nature. Most of these models are both computationally complete and efficient. In the last two decades, a great interest has been devoted to propose computational models inspired from nature able to solve intractable problems. Among these models, some of the most well known are based on formal language theory, such as *Membrane systems* [25], *Splicing systems*, *Insertion-deletion systems* [24], *Networks of evolutionary processors* [3].

Inspired by the evolution of cell populations, Networks of evolutionary processors (NEP) [3] offer efficient solutions to **NP**-complete problems by mimicking the massively parallel evolution of cell communities at the syntactic level. From a computational perspective, an NEP can be defined as a set of processors placed in the nodes of a virtual graph which defines an underlying architecture. As computational devices, NEPs compute by alternating two types of steps, namely evolutionary and communication steps, until a predefined stopping condition is fulfilled. During an evolutionary step, each processor node acts on local data in accordance with its predefined evolutionary rules, while during a communication step, the data are interchanged between connected nodes following a given protocol. This protocol indicates the conditions imposed by a processor when sending and simultaneously receiving data. It is worth mentioning that the general idea of the model is to show that very simple processors (based on very simple replacement operations), working synchronously in parallel and exchanging data with each other under a simple communication protocol, are able to efficiently simulate Turing machines and characterize complexity classes [18].

Recently, two new models based on NEP was proposed, namely: Networks of Polarized Evolutionary Processors (NPEP) [1] and Network of Evolutionary Processors for Optimization (NEPO) [12]. NPEP model proposes a filtering strategy based on the concept of polarization and the affinity of electrical charges. Nodes and words are polarized and the words migration from one node to another simulates the transport of molecules through an electrochemical gradient. Furthermore, NEPO defines a novel selection process based on a quantitative filtering strategy that uses an evaluation of

This work was supported by a grant of the Romanian National Authority for Scientific Research and Innovation, project number POC P-37-257.

the projections of words and not the entire words which seems able to select better the words representing the most optimal candidates to solve the instance of the problem. The reader interested in a survey of the main results regarding the strategy of communication based on polarization in NEP models is referred to [22].

Up to now, there have been reported several attempts to implement some of these models "in silico" but never "in vivo" or "in vitro". Some hardware implementations using FPGAs and GPUs have been initiated in [4], [20]. Several works have reported interesting approaches to handle the size of non-trivial problems over existing hardware solutions, see, e.g., [21].

On the other hand, the simulation of some bio-inspired computing models relies on software applications based on multicore desktop or clusters of computers [7], [8]. In addition, several software simulators for models similar to NEP using massively parallel platforms for multicore desktop computers, clusters of computers and cloud resources have been reported in [10], [23]. However, none of these software solutions are able to handle the size of non-trivial problems within a massively parallel and distributed environment and also present some problems of scalability. In spite of these efforts, all these shortcomings are some of the main reasons why bio-inspired computing models, including the NEP models, are not widely used to solve real problems [10].

Therefore, the only way to simulate the computation in such models seems to be by means of ultra-scalable simulators able to encapsulate the inherent parallelism of these computations. When these models are simulated on conventional computers, the total amount of space needed usually becomes exponential [23]. Nowadays, the emergence and consolidation of massively distributed platforms for big data scenarios make these platforms potential candidates for the development of ultra-scalable simulators able to run non-trivial bio-inspired computational models. The *Map Reduce* programming model implemented by Apache Hadoop was used in the Membrane computing area with a preliminary theoretical study for distributed implementation of P System models [7]. Subsequently, in [5] a Hadoop based implementation is used for generating test suites for P systems supported by a distributed NoSQL database for storage. In particular, these platforms can efficiently run distributed computations despite the notorious difficulty to scale and parallelize these models, often due to inherent interdependencies within graph data [16].

Some of these platforms work according to the Bulk Synchronous Parallel (BSP) processing model [28]. These BSP implementations for big data scenarios are reaching considerable levels of maturity, such as Apache Giraph [9], which features disk storage support for large graphs. Due to their inherently parallel and distributed nature, bio-inspired computational models such as NEP might benefit from the high performance and parallel computation of these distributed computing platforms. Recently in [11], was proved that, computational simulations for NPEP model can be adapted to these computing platforms by developing an engine that uses Apache Giraph on top of the Hadoop platform. With this engine, named NPEPE, it was shown that BSP model implemented by Giraph is a natural architecture to implement simulations of NPEP model. To the best of our knowledge, massively distributed platforms using BSP processing model have not been used as an alternative in the simulation approach of these bio-inspired computational models. As a continuation of the work introduced in [11], in this paper we present a highly computational framework extended from NPEPE, that is able to run any algorithm based on different variants of the NEP model. We consider that any model of NEP's family fit nicely in these distributed computing platforms. The main reason for this fact is that the computational perspective of the architecture underlying NEP is very similar to the architecture of the BSP model.

Our proposed extended framework named gNEP, takes advantage of the inherent Giraph and Hadoop parallelism and scalability to deploy and run NEP solutions (including any variant of NEP model). We have designed a flexible and reusable architecture for gNEP enabling the incorporation and adaptation of NEP models (e.g. NPEP, NEPO, and further new ones) but also other related models, e.g., Networks of Splicing Processors (NSP) [13], [19] and Networks of Genetic Processors (NGP) [2].

With respect to the NPEPE engine, gNEP provides (1) a highly flexible architecture able to easily extend software components in order to include other related NEP models (e.g. NSP, NGP, and new ones), (2) generality (i.e. it can execute any NEP, NPEP and NEPO algorithms), (3) simplicity for representing and generating the input configuration files through a translator software component, and (4) an elevated high scalability, fault tolerance and resilience.

In order to show the scalability of the proposed framework, we perform several experiments to demonstrate that NPEP algorithms and therefore, any other NEP algorithm, can be easily deployed and run using gNEP. For that, we have slightly adapted a NPEP solution for the "3-colorability" problem from [1] and implemented in [11]. These experiments were defined using different graph sizes and topologies. With them, we show that gNEP framework can correctly deploy and execute any NEP algorithm. The obtained results suggest that gNEP is able to execute bio-inspired solutions to NP-complete problems requiring high scalability for computationally intensive tasks.

The rest of the paper is organized as follows. In Section II, we briefly introduce the NPEP model. Section III describes some big data distributed computing frameworks. In particular, graph oriented and BSP based platforms are introduced. Section IV describes the gNPEP architecture, its new components and the main extended ones, and how they interact with Giraph. Section V briefly recalls the solution for the 3-colorability problem from [1]. In Section VI, we show the results obtained when executing this solution on the gNEP. In addition, we informally discuss and thoroughly analyze these results. Finally, in section VII some conclusions are presented and future work directions are discussed.

II. Networks of Polarized Evolutionary Processors - NPEP

In order to show the suitability of gNEP framework and illustrate the way in which one bio-inspired NEP model works, we have chosen the NPEP variant. To this aim, we include in this section a brief informal introduction of this model. We chose this variant because of in the SectionVI, we informally compare the deployment of an NPEP algorithm in gNPEP against its deployment in NPEPE engine.

First, following [1] we introduce and summarize some basic notation. An *alphabet* is a finite and non-empty set of symbols. Any sequence of symbols from an alphabet A is called *word* over A. The set of all words over A is denoted by A^* and the empty word is denoted by ε. A homomorphism from the monoid V^* into the monoid (group) of additive integers $\mathbf{Z}$ is called *valuation* of V^* in $\mathbf{Z}$.

We say that a rule $a \to b$, with $a, b \in V \cup \{\varepsilon\}$ and $ab \neq \varepsilon$ is a *substitution rule* if both a and b are not ε; it is a *deletion rule* if $a \neq \varepsilon$ and $b = \varepsilon$; it is an *insertion rule* if $a = \varepsilon$ and $b \neq \varepsilon$. The set of all substitution, deletion, and insertion rules over an alphabet V are denoted by Sub_V, Del_V, and Ins_V, respectively. Given a rule σ as above and a word $w \in V^*$, we define the following *actions* of σ on w:

- If $\sigma \equiv a \to b \in Sub_V$, then $\sigma(w) = \begin{cases} \{ubv : \exists u, v \in V^* \ (w = uav)\}, \\ \{w\}, \text{ otherwise.} \end{cases}$
- If $\sigma \equiv a \to \varepsilon \in Del_V$, then $\sigma^r(w) = \begin{cases} \{u : w = ua\}, \\ \{w\}, \text{ otherwise} \end{cases}$ and $\sigma^l(w) = \begin{cases} \{v : w = av\}, \\ \{w\}, \text{ otherwise.} \end{cases}$
- If $\sigma \equiv \varepsilon \to a \in Ins_V$, then $\sigma^r(w) = \{wa\}$, $\sigma^l(w) = \{aw\}$.

Note that $\alpha \in \{l, r\}$ expresses the way of applying a deletion or insertion rule to a word, namely in the left ($\alpha = l$), or in the right ($\alpha = r$) end of the word, respectively. It is worth mentioning that the action mode of a substitution rule applied to a word w: it returns the set of all words that may be obtained from w depending on the position in w where the rule was actually applied. For every evolutionary rule σ, action $\alpha \in \{l, r\}$, (α is missing when σ is a substitution rule) and $L \subseteq V^*$, we define the *α-action of σ on L* by $\sigma^\alpha(L) = \bigcup_{w \in L} \sigma^\alpha(w)$. Given a finite set of rules M, we define the *α-action of M* on the word w and the language L by $M^\alpha(w) = \bigcup_{\sigma \in M} \sigma^\alpha(w)$ and $M^\alpha(L) = \bigcup_{w \in L} M^\alpha(w)$, respectively.

A *polarized evolutionary processor* over V is a pair (M, α, π), where:

- M is a set of substitution, deletion or insertion rules over the alphabet V. Formally: $(M \subseteq Sub_V)$ or $(M \subseteq Del_V)$ or $(M \subseteq Ins_V)$. The set M represents the set of evolutionary rules of the processor. As one can see, a processor is "specialized" in one evolutionary operation, only.
- α gives the action mode of the rules of the node. If $M \subseteq Sub_V$, then α is missing.
- $\pi \in \{-, +, 0\}$ is the polarization of the node (negatively or positively charged, or neutral, respectively).

The set of polarized evolutionary processors over A is denoted by EP_A. Hereinafter we use "processor" for referring to a polarized evolutionary processor.

A NPEP can be formally defined as a 7-tuple $\Gamma = (V, U, G, N, \varphi, X_I, X_O)$:

- V and U are the input and network alphabets respectively, $V \subseteq U$.
- $G = (X_G, E_G)$ is an undirected graph with vertices X_G and edges E_G.
- $N : X_G \to EP_U$ is a mapping that associates each vertex with the corresponding processor in a given NPEP. Each processor $N(x)$, for each $x \in X_G$ contains:
  - M_x: finite set of evolutionary rules (insertion, deletion, substitution).
  - α_x: rule application mode (to the left or the right end of the word, or anywhere within the word. Substitution rules are always applied to an arbitrary position).
  - $\pi_x \in \{-, 0, +\}$ determines the polarity of the processor.
- $\varphi : U^* \to \mathbb{Z}$ is the valuation function that reveals the polarity of a word.
- $X_I, X_O \in X_G$ are the *input* and the *output* nodes, respectively.

The dynamics of the NPEP model is determined by *evolutionary* and *communication* steps performed alternatively which change the configuration of each processor. A configuration of an NPEP Γ is a mapping $C : X_G \to 2^{U^*}$ which associates a set of words with every processor at a given moment. Firstly, the input word encoding the instance of the problem is injected into X_I (the input processor) and the rest of the processors are empty. Then, a sequence of iterations is performed until a *halting* condition is reached. In each iteration, an evolutionary step followed by a communication step are executed, each one in parallel by each independent processor. The sequence of these alternative steps defines a computation.

Formally, given a word $w \in V^*$, the initial configuration of Γ on w is defined by $C_0^{(w)}(x_I) = \{w\}$ and $C_0^{(w)}(x) = \emptyset$ for all $x \in X_G \setminus \{x_I\}$.

When changing the current configuration by an evolutionary step, each component $C(x)$ of the configuration C is changed in accordance with the set of evolutionary rules M_x associated with the node x. Formally, we say that the configuration C' is obtained in *one evolutionary step* from the configuration C, written as $C \Longrightarrow C'$, iff $C'(x) = M_x^{\alpha_x}(C(x))$ for all $x \in X_G$.

When changing the current configuration by a communication step, each node processor $x \in X_G$ sends out copies of all its words but keeping a local copy of the words having the same polarity to that of x only, to all the node processors

connected to x and receives a copy of each word sent by any node processor connected with x providing that it has the same polarity as that of x. Note that, for simplicity reasons, we prefer to consider that a word migrates to a node with the same polarity and not an opposed one. Formally, we say that the configuration C' is obtained in *one communication step* from configuration C, written as $C \vdash C'$, iff

$C'(x) = (C(x) \setminus \{w \in C(x) \mid sign(\varphi(w)) \neq \pi_x\}) \cup \bigcup_{\{x,y\} \in E_G} (\{w \in C(y) \mid sign(\varphi(w)) = \pi_x\})$,

for all $x \in X_G$. Here $sign(m)$ is the sign function which returns $+, 0, -$, provided that m is a positive integer, is 0, or is a negative integer, respectively. Note that all words with a different polarity than that of x are expelled. Further, each expelled word from a node x that cannot enter any node connected to x (no such node has the same polarity as the word has) is lost.

Let Γ be a NPEP, the computation of Γ on the input word $w \in V^*$ is a sequence of configurations $C_0^{(w)}, C_1^{(w)}, C_2^{(w)}, \ldots$, where $C_0^{(w)}$ is the initial configuration of Γ on w, $C_{2i}^{(w)} \Longrightarrow C_{2i+1}^{(w)}$ and $C_{2i+1}^{(w)} \vdash C_{2i+2}^{(w)}$, for all $i \geq 0$. Note that the configurations are changed by alternative steps.

A computation as above *halts*, if there exists a configuration in which the set of words existing in the output node $\underline{Out}$ is non-empty. Given a NPEP Γ and an input word w, we say that Γ accepts w if the computation of Γ on w halts.

Let Γ be a NPEP with the input alphabet V; the *time complexity* of the finite computation $C_0^{(x)}, C_1^{(x)}, C_2^{(x)}, \ldots C_m^{(x)}$ of Γ on $x \in V^*$ is denoted by $Time_\Gamma(x)$ and equals m.

Note that a halting computation obtains at least one word in the output node. This led us to consider NPEPs as problem solvers. Informally, we say that a decision problem P is solved in time $\mathcal{O}(f(n))$ by NPEPs if there exists a family $\mathcal{G}$ of NPEPs such that for each instance p of size n of the problem one can effectively construct a NPEP $\Gamma(p) \in \mathcal{G}$ which accept in time $\mathcal{O}(f(n))$ the word encoding the given instance. This means that the word is accepted if and only if the solution to the given instance of the problem is "YES". This effective construction is called an $\mathcal{O}(f(n))$ time solution to the considered problem. Even more, the output node collects the encodings of all solutions to the problem.

If a NPEP $\Gamma \in \mathcal{G}$ constructed above accepts the language of words encoding all instances of the same size n, then the construction of Γ is called a *uniform solution*. Intuitively, a solution is uniform if for problem size n, we can construct a unique NPEP solving all instances of size n taking the (reasonable) encoding of instance as "input".

III. Giraph: a BSP Ultra-scalable Computing Platform for Big Data Scenarios

In the last decade, computing architectures were subject to a significant evolution. The requirements for data intensive and computation intensive algorithms and the emergence of Big Data scenarios quickly contributed to a significant surge of popularity for large-scale computing platforms during the last few years.

Despite of MapReduce [6] through their Hadoop implementation (as a large-scale computing framework) [26] has been extensively used in applications for several practical domains it presents well-identified drawbacks. First, lack of suitability for certain classes of applications [11]. This drawback motivated researchers to propose different alternatives, mainly in the context of iterative algorithms (e.g. Spark [30]), real-time analytics (e.g. Storm [27]). Second, the lack of graph processing in order to handle graph-like structures and highly scalable graph computations. It motivated the emergence of distributed computing platforms based on the Bulk Synchronous Parallel model (BSP) [28] (e.g. Pregel [17], Dato [15] (formerly GraphLab), Spark GraphX [29] and PowerGraph [14] and Giraph [9]). Particularly, Giraph was proposed as the open source counterpart of Google Pregel.

The BSP model implemented by Pregel is, at the most basic level, a two step process performed iteratively and synchronously: 1) one process performing computations on local data and 2) one process communicating the results. Each compute/communicate iteration is called a superstep, with synchronization of the parallel tasks occurring at the superstep barriers. Each superstep represents atomic units of parallel computation. Finally, a global check procedure determines if all compute functions are finished.

As Giraph is similar to Pregel, the implementation of BSP model follows this same procedure. In [11] was stated that this implementation is a natural architecture to implement NPEP models. Since the NEP's model family shares the same underlying architecture, it is obvious to try applying this statement to any of these models. Therefore, each process defined by the BSP model has a counterpart in the NEP model (together with its variants) definition.

The main similarities between BSP and NEP model are mentioned below:

- Massive parallelism and distributed computing underlie the paradigm of both Giraph and NEP.
- A Giraph vertex and an NEP processor can be seen as similar processing units.
- A superstep includes one evolutionary and one communication step in the NEP model.
- A global check procedure can be implemented to verify the halting condition in the NEP model.

Due to these similitudes, its open source nature and how well its model fits the problem at hand, Giraph was chosen as the basic building block of the NPEPE engine and remains valid to the extended framework proposed here (gNEP). In Section IV we provide a detailed description of how the NPEPE and gNEP designs have been done on top of Giraph.

IV. Architecture for the gNEP

We have developed the gNEP framework, an ultra-scalable computational simulator that can deploy and run bioinspired algorithms of the NEP model and its variants containing as many nodes as needed by using the Giraph framework on top of Hadoop. gNEP is generalization of NPEPE engine [11] designed in order to facilitate the simulation of other NEP

models. Some software components remain in the gNEP architecture but others have had to be re-designed and implemented. The original architecture for NPEPE is composed of *I/O* and *Computation* modules as it is depicted in the Figure 1.

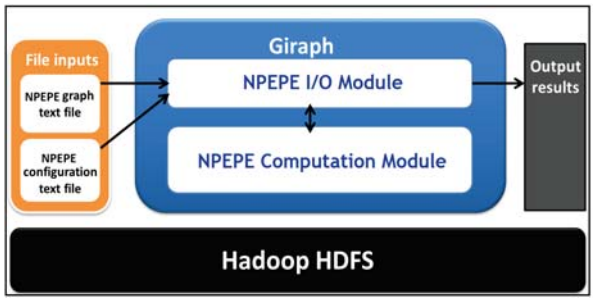

Fig. 1. NPEPE architecture.

The way in the components illustrated in Figure 1 are mapped to the Giraph execution model are described below.

In the **Computation Module** each processor of an NPEP performs a set of operations on incoming messages. These operations are implemented and managed by two subcomponents: *Processor* and *NpepComputation*. *Processor* represents a vertex instance of Giraph. Each *processor* encapsulates the evolutionary rules to be applied to incoming messages. It can be one of three types: *Input*, *Output* and *NPEP* processor. By the other hand, *NpepComputation* invokes a Giraph method such that every processor on each superstep can receive message and process them by their defined evolutionary rules. After that, each vertex can discard the processed messages, store them or send them to all connected vertices. **I/O module** is composed by three components: *NpepInputFormat* (which converts a representation of NPEP network using a text format file into a Giraph graph stored in an output text file in JSON format), *NpepOutputFormat* (which dumps the data content of each vertex/processor on an external file) and finally, *NpepWritable* (that stores the words that vertexes/processors keep for further processing). For additional details about these NPEPE modules, the reader may refer to [11].

In order to make this architecture flexible and allow simulations of any NEP family's model (and other related models) we needed to extend both I/O and Computation Modules to be used into gNEP framework. In addition, to guarantee the extensibility and re-usability of gNEP we have developed a new component named **gNEP translator**. The new architecture including these components is depicted in the Figure 2.

As we can observe in this Figure, the **gNEP Traslator** receives two input text files. The first one represents the instance of the problem to solve. The second one contains two sections: *NEP components description* and the *NEP network description*. The first section provides the values and parameters for all components in the NEP algorithm (processors, evolutionary rules, input word, filters). The second section contains the definition of the network underlying graph for the NEP algorithm.

The execution of gNEP starts when the *I/O module* parses the input files received for the gNEP Translator. Then, gNEP generates and deploys a set of Giraph elements in the *gNEP excecutor Module*. In particular, gNEP deploys the extended Giraph elements from extended NPEPE *Computation module* (e.g. nodes or messages) which can be run on top of a Hadoop platform. At the beginning, only the vertex representing the input node executes, which sends out the initial words after applying its rules to them. Hereafter, during each superstep, Giraph invokes all vertices in parallel so that they can process the received messages and send out those that make it through the specific filter. This process runs until all vertices decide to stop. Then, the halting condition for the computation is achieved through a voting process. The vertex representing the output node is configured to pass its vote by invoking a specific method when it receives a message and the other vertices have sent their vote for stopping. At the end, all words arriving to the output node are written to an output text file by the *gNEP I/O Module component*.

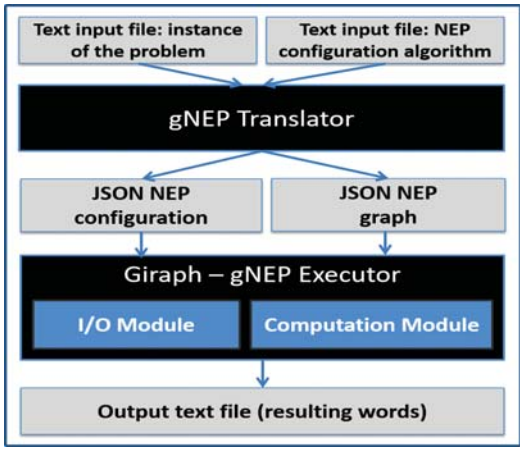

Fig. 2. gNEP architecture.

We remark that, for a same instance of a problem, *gNEP Translator Component* is able to receive several NEP configuration algorithm files (each one representing the configuration of a different NEP model algorithm e.g., NEPO, NEP, NPEP algoritm) and it can parse them into different JSON NEP graphs able to be deployed in the gNEP Executor.

Finally, we stated that, gNEP framework can directly execute any input representing an NEP algorithm. This means that gNEP is able to directly solve any NP-complete problem described by a NEP solution (i.e. reconfiguration processes and implementation of additional features, are not needed). In addition, solutions for other bio-inspired models similar to NEP can be easily deployed into gNEP framework by specializing some components of the gNEP architecture (specifically, components in gNEP Translator and Computation Modules) in order to include the new specific elements (rules, filters, etc).

V. *g*NEP DEPLOYMENTS OF 3−COLORABILITY PROBLEM SOLUTION USING NPEP

For simplicity, we use the same problem that when was introduced the NPEPE engine in [11]. It allow us to informally compare the *gNEP construction* and include some informal

comparisons (in the next section) with the experiments introduced previously. In this context, we consider relevant to briefly recall now a construction of an NPEP that provides a linear time solution to the "3-colorability" problem, which is a well-known problem that belongs to the NP-complete class. This problem has numerous applications: scheduling, register allocation, pattern matching, frequency assignment for mobile radio stations, etc. Formally, we define this problem as follows. Let $G = (V, E)$ be a graph with set of vertices $V = \{v_1, v_2, \ldots, v_n\}$ and set of edges $E = \{e_1, e_2, \ldots, e_m\}$ where each e_k is given in the form $e_k = \{v_i, v_j\}$ for some $1 \leq i \neq j \leq n$. A 3-coloring for G is a function $c : V \rightarrow \{r, g, b\}$ such that $c(v_i) \neq c(v_j)$ for every edge $e_k = \{v_i, v_j\} \in E$. A solution for the "3-colorability" problem of G can be a set containing all possible 3-colorings for G.

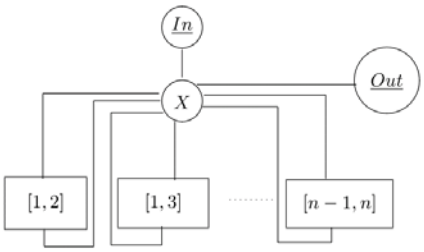

Fig. 3. General shape of the underlying graph G.

Let $Y = (B, Q)$ be a graph with set of vertices $B = \{v_1, v_2, \ldots, v_n\}$ and set of edges $Q = \{e_1, e_2, \ldots, e_m\}$, where each e_k is given in the form $e_k = \{v_i, v_j\}$, for some $1 \leq i \neq j \leq n$.

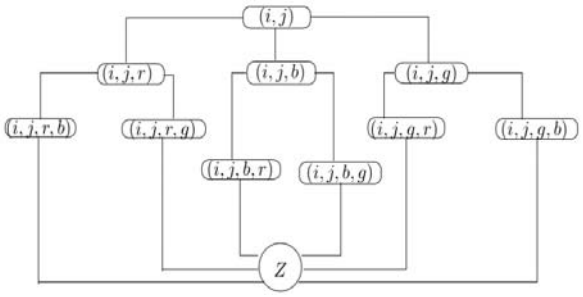

Fig. 4. The subgraph associated with each edge $\{i, j\}$.

We construct the NPEP Γ that will decide whether the graph Y can be colored with three colors, namely *red, blue, green,* as follows:

$$\Gamma = (V, U, G, \mathcal{R}, \varphi, \underline{In}, \underline{Out})$$

with

$V = \{T_0, T_1, T_2, \ldots, T_{n(n-1)/2}\} \cup$
$\{e_1, e_2, \ldots, e_{n(n-1)/2}\} \cup \{a\},$
$U = V \cup \{T'_0, T'_1, T'_2, \ldots, T'_{n(n-1)/2}\} \cup$
$\{e'_1, e'_2, \ldots, e'_{n(n-1)/2}\} \cup \{\bar{e}_1, \bar{e}_2, \ldots, \bar{e}_{n(n-1)/2}\} \cup$
$\{r_i, b_i, g_i \mid 1 \leq i \leq n\} \cup \{T''_0, F\}.$

We first give the shape of the underlying graph G in Figure 3, where each box labeled by $[i, j]$, $1 \leq i < j \leq n$, encapsulates a subgraph associated with the edge $\{i, j\}$ which is described in Figure 4.

We now define the valuation mapping φ:
$\varphi(T_k) = 0, 0 \leq k \leq n(n-1)/2,$
$\varphi(T'_j) = 1, 0 \leq j \leq n(n-1)/2,$
$\varphi(z_i) = 0, 1 \leq i \leq n, z \in \{r, b, g\},$
$\varphi(e_j) = 0, 1 \leq j \leq n(n-1)/2,$
$\varphi(z'_i) = 1, 1 \leq i \leq n, z \in \{r, b, g\},$
$\varphi(e'_j) = -2, 1 \leq j \leq n(n-1)/2,$
$\varphi(z''_i) = 3, 1 \leq i \leq n, z \in \{r, b, g\},$
$\varphi(\bar{e}_j) = 0, 1 \leq j \leq n(n-1)/2,$
$\varphi(a) = 1, \quad \varphi(F) = 1, \quad \varphi(T''_0) = -1.$
The nodes of the network are defined in Table I.

TABLE I
THE PARAMETERS OF THE NODES OF Γ.

Node	M	π
$\underline{In}$	$\bigcup_{i=1}^{n} \{a \rightarrow r_i, a \rightarrow b_i, a \rightarrow g_i\}$	+
X	$\{T_k \rightarrow T'_{k-1} \mid 1 \leq k \leq n(n-1)/2\} \cup$ $\{T_0 \rightarrow T''_0\}$	0
$(i, j), 1 \leq i \neq j \leq n$	$\{e_k \rightarrow e'_k \mid e_k = \{i, j\}\}$	+
$(i, j, z), 1 \leq i \leq n,$ for $z \in \{r, b, g\}$	$\{z_i \rightarrow z'_i\}$	−
$(i, j, z, y), 1 \leq i \leq n,$ for $z, y \in \{r, b, g\},$ $z \neq y$	$\{y_j \rightarrow y''_j\}$	0
Z	$\{T'_k \rightarrow T_k \mid 0 \leq k \leq n(n-1)/2\} \cup$ $\{z'_i \rightarrow z_i, g''_j \rightarrow g_j\} \cup$ $\{e'_k \rightarrow \bar{e}_k \mid e_k = \{i, j\}\}$	+
$\underline{Out}$	$\emptyset$	−

The reader interesting in the complexity results for this 3-colorability solution can refer to [1]. The main assumption of NPEP model and therefore of this solution is that all necessary resources to solve the problem are available all the time. However, in a silicon implementation, the computational resources are limited and therefore, achieving the maximal parallelism required by these models is still a challenge. Consequently, it is necessary to make some small modifications of the NPEP solutions in order to enable their deployment on silicon computational architectures. Specifically, following the guidelines reported in [15], it is possible to affirm that a max-parallel execution requiring coordination between vertices is not suitable for the BSP model. For this reason, we have included some adaptations to the NPEP solution introduced before. In particular, we reduce the communication between node processors when we consider it unnecessary. That is, we discard some of the words that would be rejected by target nodes because their polarity is not the same. We force the blockage of these words in order to avoid overhead during the communication step. With this modification, gNEP is able to consume less communication and computational resources thanks to the rejection of undesirable words.

VI. Experimental Results

We have conducted several experiments running the gNEP framework and the slightly adapted NPEP algorithm discussed above for input graphs of different sizes and topologies. In these experiments, we show that the gNEP can correctly deploy and execute NPEP models and therefore it is able to execute bio-inspired solutions of other related NEP models

for NP-complete problems. In addition, these results show interesting properties of the scalability of NPEP solutions.

A. Experimental Environment

For performance characterization, an 11-node Hadoop cluster was configured. 10 nodes provide both computation (Giraph workers) and storage resources (DataNode servers). The other node serves as both the MapReduce scheduler and NameNode storage manager (Giraph Master). The computers were interconnected by an Ethernet switch of 100Mbps. Each node has one Intel(R) Core(TM)2 Quad Q8400 processor clocked at 2.66GHz, 4GB of RAM and 150GB of hard drive storage. All nodes used CentOS 6.4, Hadoop framework 2.6, Giraph 1.1 and Java 1.7. The Hadoop Distributed File System was configured with 2-replication.

B. Experiments

In the first experiment, we defined a simple test for small graphs and small NPEP networks sizes in order to prove gNEP functioning. Since the experimental environment is the same as the one used in [11], we compare our results with the results obtained for the same experiment in this previous work. The experimental results are shown in Table II.

TABLE II
RESULTS OF THE EXECUTION WITH SMALL GRAPHS FOR DIFFERENT NPEP SOLUTION.

Graph size	NPEP size	Total words	Number solutions	Time NPEPE (secs)	Time gNEP (secs)
4	43	81	12	36	17
6	68	729	66	66	19
7	68	2187	192	128	20
8	79	6561	384	356	21

Second, we show the potential of the gNEP for huge graphs. We run the algorithm on graphs of an increasing number of vertices (up to 1250), all of which have exactly six solutions. In the results depicted in Table III, we show the relation between the execution time and the size of the graph.

TABLE III
RESULTS OF THE EXECUTION OF THE SOLUTION WITH 6 SOLUTIONS GRAPHS.

Graph size (nodes)	Graph links (edges)	NPEP size (nodes)	Execution time (secs)
645	25	278	0.0108
1320	50	553	0.0158
2670	100	1103	0.0305
5370	200	2203	0.1083
10770	400	4403	0.9712
13470	500	5503	2.1978
26970	1000	11003	37.0408
33720	1250	13753	86.4439

In addition, we have compared the execution time for graphs with different numbers of NPEP processors and different numbers of solutions, as can be observed in Table IV. These experiments were executed using 20 Giraph workers (10 nodes x 2 cores per node).

TABLE IV
RESULTS OF THE EXECUTION OF THE SOLUTION WITH DIFFERENT GRAPHS.

Graph size (nodes)	Graph links (edges)	NPEP size (nodes)	Solutions (words)	Execution time (secs)
24	45	498	48	35.56
25	48	531	96	35.78
25	47	520	6	38.89
50	97	1070	6	56.88
22	25	278	1990656	1752.726
23	26	289	3981312	3384.966

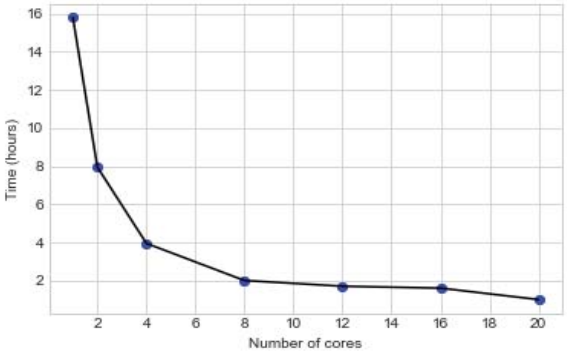

Fig. 5. Scalability for the 400 vertices and 6 solutions graph.

Finally, we show a preliminary analysis of the scalability capabilities of the simulator with our adapted solution. In particular, we show how the gNEP framework performs with respect to three key parallel computing metrics: scalability, speedup and efficiency. For this, we use an instance of the problem with 400 vertices, 797 edges and 6 solutions. Figure 5 illustrates the scalability results, showing the execution time of gNEP for different numbers of Giraph workers. The obtained results indicate that the horizontal scalability improves dramatically the execution times for this experiment when the number of cores is increased from 2 cores to 8 cores. For 8 cores we obtain an optimal value and from 10 to 20 cores, the runtime slightly improve.

Figure 6 contains the results for speedup ($\text{Speedup}_n = t_1/t_n$). These results explain how many times using more than one Giraph worker is better than using only one.

Finally, the efficiency of gNEP is illustrated by Figure 7. We define the efficiency as how well we take advantage

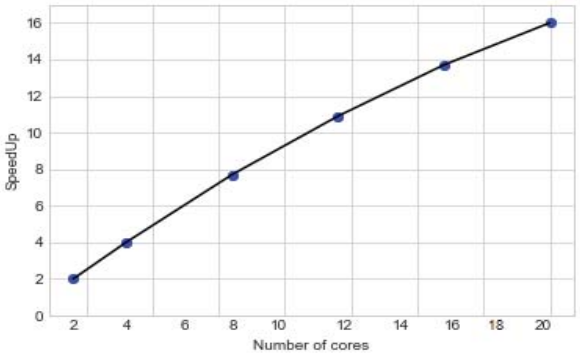

Fig. 6. Speedup for the 400 vertices and 6 solutions graph.

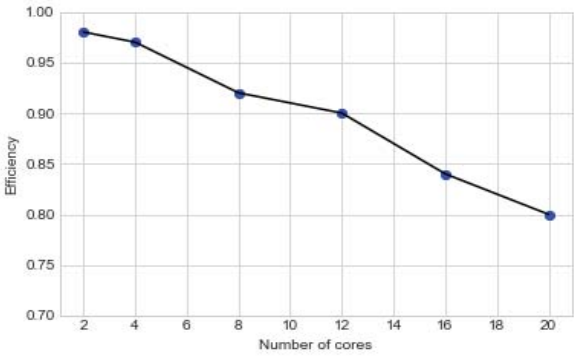

Fig. 7. Efficiency for the 400 vertices and 6 solutions graph.

of the resources with respect to the "perfect parallelization" (i.e. if we have n Giraph workers and execution time is t, then for $2n$ workers the execution time assuming "perfect parallelization" should be $t/2$). The efficiency is given by the equation $\text{efficiency}_n = \text{Speedup}_n/n$. The overall obtained results reveal a slight decrease in the efficiency (approx. 20%) when we added more cores in the computation. This fact is an expected result because resource communication and management usually degrade the efficiency when this situation occurs.

VII. Conclusions and Future Work

We have proposed an extension of the highly scalable engine NPEPE, called gNEP, which exploits massively parallel high-end computing resources using Apache Giraph on top of Hadoop. gNEP provides an ultra-scalable simulator of the several natural computing models from the Networks of Evolutionary Processors NEP family. gNEP provides extensibility, reusability, generality, simplicity and high scalability. Therefore, it can be considered as a suitable framework to simulate any NEP solution for a hard computational problem.

We have demonstrated by means of several experiments that a NPEP algorithm can be easily deployed and executed using the gNEP. The results of these experiments reveal that gNEP can be considered as a suitable framework to simulate any NEP solutions (including NPEP, NEPO and solutions from new models related with NEP) for large data sets, which entail computationally intensive tasks and require high scalability.

We consider two interest research directions for our immediate work in order to enhance the performance of the gNEP framework: 1) Adapt it to other highly-scalable big data platforms as SparkX to prove its performance; 2) Create an gNEP Hybrid Engine combining multi-core GPU clusters with the highly-scalable big data platform resulting from 1).

References

[1] F. Arroyo, S. Gómez-Canaval, V. Mitrana and S. Popescu, "Networks of polarized evolutionary processors are computationally complete", in Int. Conf. LATA, LNCS 8370, Springer, 2014, pp. 101–112.
[2] M. Campos, J. Sempere, "Accepting networks of genetic processors are computationally complete", Theor. Comp. Sci, vol. 456, 2012, pp. 18-29.
[3] J. Castellanos, C. Martín-Vide, V. Mitrana and J. Sempere, "Networks of evolutionary processors", Acta informatica, vol. 39, pp. 517–529, 2003.
[4] J. Cecilia, J. García, G. Guerrero, M. Martínez, M.J. Pérez and M. Ujaldón, "The GPU on the simulation of cellular computing models", Soft Computing, vol. 16, 2012, pp. 231–246.
[5] A. Ciobanu, F. Ipate, "Implementation of P Systems by Using Big Data Technologies", Membrane Computing", Springer, pp. 117–137, 2014.
[6] J. Dean and S. Ghemawat, "MapReduce: Simplified data processing on large clusters", Comm. of the ACM, vol. 51, 2008, pp. 107–113.
[7] L. Diez, R. Núñez, M. Cruz, and A. Ortega, "Distributed Simulation of P Systems by Means of Map-Reduce: First Steps with Hadoop and P-Lingua", Advances in Comp. Intelligence, Springer, pp. 457–464, 2011.
[8] M. García-Quismondo, R. Gutiérrez, I. Pérez, M.J. Pérez-Jiménez and A. Riscos, "An Overview of P-Lingua 2.0", in Membrane Computing, LNCS 5957, 2010, pp. 264–288.
[9] "Apache Giraph", 2014, http://giraph.apache.org/, [Consulted 02-2019].
[10] S. Gómez, A. Ortega and P. Orgaz, "Distributed simulation of NEPs based nn-demand cloud elastic computation", in Advances in Computational Intelligence, LNCS 9094, 2015, pp. 40-54.
[11] S. Gómez, B. Ordozgoiti and A. Mozo, "NPEPE: Massive natural computing engine for optimally solving NP-complete problems in Big Data scenarios", in New Trends in Databases and Information Systems, CCIS vol. 539, pp. 207–217, 2015.
[12] S. Gómez, V. Mitrana, M. Vinyals: "Solving optimization problems by using networks of evolutionary processors with quantitative filtering", J. Comput. Science, vol. 16, pp. 65–71, 2016.
[13] S. Gómez-Canaval, V. Mitrana, J. Sánchez, "Networks of splicing processors with evaluation sets as optimization problems solvers", Information Sciences, vol. 369, pp. 457–466, 2016.
[14] J. González, X. Low, H. Gu, D. Bickson and C. Guestrin, "PowerGraph: Distributed graph-parallel computation on natural graphs", Proc. Conf. on Operating Systems Design and Implementation, pp. 17–30, 2012.
[15] Y. Low, D. Bickson, J. González, C. Guestrin, A. Kyrola, J. Hellerstein, "Distributed GraphLab: A framework for machine learning and data mining in the Cloud", Proc. of VLDB, vol. 5, pp. 716–727, 2012.
[16] A. Lumsdaine, D. Gregor, B. Hendrickson, J. Berry, "Challenges in parallel graph processing", Parallel Proces. Lett., vol. 17, pp. 5–20, 2007.
[17] G. Malewicz, M. Austern, A. Bik, J. Dehnert, I. Horn, N. Leiser and G. Czajkowski, "Pregel: A system for large-scale graph processing", in Proc. of Int. Conf. on Management of Data, pp. 135–146, 2010.
[18] F. Manea, M. Margenstern, V. Mitrana and M.J. Pérez-Jiménez, "A new characterization of **NP**, **P**, and **PSPACE** with accepting hybrid networks of evolutionary processors", Th. Comp. Syst. vol. 46, pp. 174–192, 2010.
[19] F. Manea, C. Martín, V. Mitrana, "Accepting nets of splicing processors: Complexity results", Ther. Com. Sci., vol. 371, pp. 72–82, 2007.
[20] A. Maroosi, R. Muniyandi, E. Sundararajan and A. Zin, "Parallel and distributed computing models on a graphics processing unit to accelerate simulation of membrane systems", Sim. Mod. Prac. and Theory, vol. 47, pp. 60–78, 2014.
[21] M. A. Martínez-del-Amor, L. F. Macías-Ramos, L. Valencia-Cabrera, A. Riscos-Núñez and M.J. Pérez-Jiménez, "Simulating P systems on GPU devices: A survey", Fundamenta Inform., vol. 136, pp. 269–284, 2015.
[22] V. Mitrana, "Polarization: a new communication protocol in networks of bio-inspired processors", J. Membrane Comp., vol. 1, 2019, pp. 1–17.
[23] C. Navarrete, M. Cruz, E. Rey, A. Ortega and J. Rojas, "Parallel simulation of NEPs on clusters", in Int. Conf. on Web Intelligence and Intelligent Agent Technology, vol. 3, 2011, pp. pp. 171–174.
[24] Gh. Păun, G. Rozenberg and A. Salomaa, "DNA Computing: New Computing Paradigms", Ch. 7, T. in Theor. Comp. Sci., Springer, 1998.
[25] Gh. Păun, "Membrane Computing - An Introduction", Natural Computing Series, Springer, 2002.
[26] K. Shvachko, K. Hairong, S. Radia and R. Chansler, "The Hadoop distributed file system", in IEEE Symposium on Mass Storage Systems and Technologies (MSST), 2010, pp. 1-10.
[27] "Apache Storm", 2013, http://storm.apache.org, [Consulted 03-2019].
[28] L. Valiant, "A bridging model for parallel computation", Magazine Communications of the ACM, vol. 33, pp. 103–111, 1990.
[29] R. Xin, J. González, M. Franklin and I. Stoica, "GraphX: A resilient distributed graph system on spark", in Int. Workshop on Graph Data Management Experiences and Systems, 2013, pp. 1–6.
[30] M. Zaharia, M. Chowdhury, T. Das, A. Dave, J. Ma, M. McCauley, M. Franklin, S. Shenker and I. Stoica, "Resilient distributed datasets: A fault-tolerant abstraction for in-memory cluster computing", in Proc. of Conf. on Networked Systems Design and Implementation, 2012, pp. 2.