Parallel Algorithm for Dynamic Community Detection

Hugo Resende
Federal Institute of Education, Science and Technology of South Minas Gerais
Passos-MG, Brazil
hugo.resende@ifsuldeminas.edu.br

Alvaro Luiz Fazenda
Federal University of Sao Paulo
Sao Jose dos Campos-SP, Brazil
alvaro.fazenda@unifesp.br

Marcos Gonçalves Quiles
Federal University of Sao Paulo
Sao Jose dos Campos-SP, Brazil
quiles@unifesp.br

***Abstract*— Many real systems can be naturally modeled by complex networks. A complex network represents an abstraction of the system regarding its components and their respective interactions. Thus, by scrutinizing the network, interesting properties of the system can be revealed. Among them, the presence of communities, which consists of groups of densely connected nodes, is a significant one. For instance, a community might reveal patterns, such as the functional units of the system, or even groups correlated people in social networks. Albeit important, the community detection process is not a simple computational task, in special when the network is dynamic. Thus, several researchers have addressed this problem providing distinct methods, especially to deal with static networks. Recently, a new algorithm was introduced to solve this problem. The approach consists of modeling the network as a set of particles inspired by a N-body problem. Besides delivering similar results to state-of-the-art community detection algorithm, the proposed model is dynamic in nature; thus, it can be straightforwardly applied to time-varying complex networks. However, the Particle Model still has a major drawback. Its computational cost is quadratic per cycle, which restricts its application to mid-scale networks. To overcome this limitation, here, we present a novel parallel algorithm using many-core high-performance resources. Through the implementation of a new data structure, named distance matrix, was allowed a massive parallelization of the particle's interactions. Simulation results show that our parallel approach, running both traditional CPUs and hardware accelerators based on multicore CPUs and GPUs, can speed up the method permitting its application to large-scale networks.**

Keywords: community detection, massively parallel algorithm, high-performance computing on hardware accelerators

I. INTRODUCTION

Real systems, such as the Internet, social networks, fast food chains, biological networks, transport networks, and others, can be modeled via complex networks or graphs [1]. For instance, to represent a transport network, nodes can represent cities and links the paths between neighbor cities. In a second example, the Web, nodes can denote websites and links, ways of targeting such sites to other web addresses [2], [3].

Among several features and measurements that can be extracted from complex networks (see [1]), one important feature is the network modular structure named communities [4].

Community detection is a very active research topic. The community structure might reveal valuable information regarding the topology and the dynamics of the system, e.g., Internet, social networks, biological networks, among others [4], [5], [6].

Although there is no universally accepted definition for communities, most papers found in the literature consider that groups of well-connected nodes are characterized by the existence of a greater number of edges within the group than with the other network groups [6]. The Figure 1 illustrates a network with three communities. Nodes with same colors constitute the communities.

According to Danon et al. [7], community detection is a NP-Complete problem. Thus, instead of using only exact algorithms, researchers have considered a variety of approaches to tackle this task [6], [8], [9], [10]. However, dealing with large-scale networks and time-varying networks is still a significant challenge that needs to be overcome.

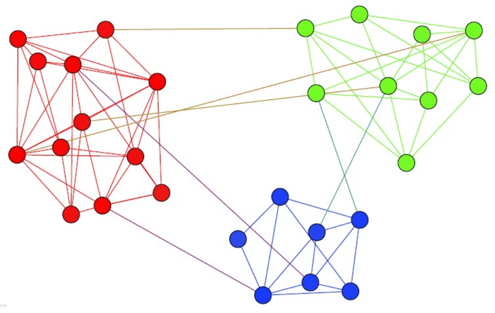
Fig. 1. A example of communities in a complex network.

Recently, a new community detection algorithm, based on a simple particle dynamic model, was proposed by Quiles et al. [10]. In their approach, each node is represented by a particle embedded in a 3D Euclidean space. The motion of particles is governed by a dynamics that takes the particles' position and the links of the graph into account. If two particles are connected through a link, they attract each other, on the contrary, they repel inversely proportional to their distance in the space. After a transient, the system reaches an equilibrium state in which particles representing densely connected nodes, or communities, are represented by clusters into the Euclidean space. Thus, by using a simple clustering algorithm, the communities are detected.

Albeit providing accurate results in both static and dynamic scenarios, owning to its high computational cost, this algorithm cannot be straight applied to large-scale networks. In this sense, high-performance computing (HPC) techniques, used in the parallelization of CPU algorithms, via OpenMP directives, and GPU, with the assistance of the CUDA platform, emerge as real alternatives in overcoming these limitations [11], [12], [13].

Here, we show that HPC techniques, via CPU and GPU, used as parallelization mechanisms for the particle community detection algorithm, allowed an escalation of the original algorithm to a higher level. Our results demonstrated that the parallel algorithm delivery results up to 50 times faster than its serial version.

The rest of this paper is organized as it follows. Section II revisits the algorithm proposed by Quiles et al. [10]. Our proposed parallel algorithm is described in Section III. Experimental results are depicted in Section IV. Some concluding remarks are drawn in Section V.

II. COMMUNITY DETECTION ALGORITHM INSPIRED ON PARTICLE DYNAMICS

In this section, we provide a detailed description of the community detection model proposed by Quiles et al. [10]. This model maps the graph nodes into a new space, named Particle Space. The model considers a straightforward and efficient dynamics, which govern the system of particles to an equilibrium state in which the cluster (or communities) are revealed when the system reaches the equilibrium.

Formally, lets assume a graph $\mathcal{G} = \{\mathcal{V}, \mathcal{E}\}$, in which $\mathcal{V}$ and $\mathcal{E}$ define the set of nodes and links, respectively. The i-th node in the network ($i = 1, \ldots, N$) is then associated to a particle's position, $\vec{x}_i(t) = (x_i, y_i, z_i)$, that evolves according to

$$\dot{\vec{x}}_i = \alpha\,\vec{F}_i^{(A)} + \beta\,\vec{F}_i^{(R)}, \tag{1}$$

where $\vec{F}_i^{(A)}$ and $\vec{F}_i^{(R)}$ are the attractive and repulsive interactions acting upon particle i and $\alpha > 0$ and $\beta > 0$ are the strengths for these interactions, respectively. Initially, each particle assumes a random initial position into the PS.

When particles i and j represent adjacent nodes in the graph, it means, the adjacency matrix ij-th entry is $A_{ij} = 1$, these particles are mutually attracted. On the other hand, particles representing unlinked nodes ($A_{ij} = 0$) repel each other.

These interactions were designed to conduct the system of particles to a stable configuration in which the clusters in the PS reveal the communities of the network. If the network has a modular structure, nodes belonging to the same community have a higher probability of sharing an edge in comparison to nodes of distinct communities. As a consequence, nodes of the same community will have a stronger attractive interaction. On the contrary, the repulsive interaction will be strong between nodes of distinct communities, or between communities. The following interaction are considered in [10]

$$\vec{F}_i^{(A)} = -\sum_{j=1}^{N} \frac{A_{ij}}{k_i}\,\frac{(\vec{x}_i - \vec{x}_j)}{\|\vec{x}_i - \vec{x}_j\|}, \tag{2}$$

$$\vec{F}_i^{(R)} = \sum_{j=1}^{N} \frac{1 - A_{ij}}{k_i}\,\frac{(\vec{x}_i - \vec{x}_j)}{\|\vec{x}_i - \vec{x}_j\|}\,e^{-\gamma\,\|\vec{x}_i - \vec{x}_j\|}, \tag{3}$$

where $\gamma > 0$ is the decay rate for the repulsive interaction as a function of the distance between particles, A_{ij} is the adjacency matrix of the graph, and k_i is the node's degree.

As stated earlier, the Particle model provides a novel and interesting approach to the community detection problem. Besides, the model can be straightforwardly applied to time-varying, or dynamic, networks.

However, a drawback of the particle system it is high computational cost. Each iteration demands the computation of the distances between every pair of particles (nodes). Thus, assuming a transient T to reach an equilibrium state, the computational complexity of the algorithm is $O(T \times N^2)$, in which N is the number of nodes.

Although having a high computational cost, the algorithm proposed by Quiles et al. has an intrinsic parallel nature. Thus, here, we have scrutinized their model to come up with a parallel equivalent algorithm.

It is worth noting that the parallelization of particle-like systems has been studied elsewhere, albeit with distinct purposes. For example, the Barnes-Hut algorithm [14], the Fast Multipole Method (FMN) [15], and the Parallel Multipole Tree Algorithm (PMTA) [16]. These approaches, to reduce the complexity cost of the integration method, have ignored the existence of interactions that might be essential in the context of community detection methods. The scrutinization of these faster integration methods to solve the Quiles et al. community detection method will be carried out in future investigations.

Next section introduces our parallel approach.

III. MASSIVELY PARALLEL ALGORITHM

As mentioned before, the community detection algorithm based on particle dynamics has a high computational cost when computing the distances between the system particles pairs. The strategy adopted here consists of developing a portable parallel algorithm to achieve a massive parallelism in the execution of this task, both for the CPU's and GPU's. To ensure the algorithm portability, we have created an auxiliary data structure consisting of an array of order $N \times N$, here called the distance matrix. This structure stores in the ij-th entry the distance between the pair of particles (i, j).

The algorithm depicted in Alg. 1 illustrate the basic steps taken into account in our parallelization approach.

Algorithm 1 Parallel Algorithm Based on Particle Dynamics

```
Input: $\mathcal{G} = \{\mathcal{V}, \mathcal{E}\}$
Output: set $C_{i=1 \cdots n_c} / \cup_{i=1}^{n_c} C_i = V$ e $\cap_{i=1}^{n_c} C_i = \emptyset$
1: initializePS();
2: $\forall$ particle $\in V$
3:     initializeDistancesParallel();
4: $\forall$ particles pair $(i, j) / i, j \in V$
5:     computePairDistancesParallel();
6:     computeGlobalDistancesParallel();
7: updatePositionsParticlesParallel();
```

The Algorithm 1 take as input an undirected and unweighted graph, and outputs a set of communities C. In line 1, the serial routine *initializePS()* assigns a random position to each particle of the system. In line 3 the parallel routine assign zero values to the distance parameters of each particle. The *computePairDistancesParallel()* procedure, shown in line 5, is responsible for evaluating the distance matrix. The *computeDistancesParallel()* (line 6) runs N parallel tasks, each one reducing a line in distance matrix to a value representing the attraction or repulsion between particles. Lastly, the *upPositionsParticlesParallel()* routine parallely updates the particle's positions.

IV. RESULTS

The community detection algorithm based on particle Dynamics is coded in C++, using the SNAP system libraries (Stanford Network Analysis Platform [17]) to deal with graph data structures. The parallelism for CPU cores uses OpenMP [18] and for GPUs uses CUDA [19] routines. The experiments were conducted on 5 different computational systems: a) a single CPU node with 20 cores shared memory configured with 2 sockets Intel Xeon E5-2630v4 2.20 GHz and 64GB for main memory; b) the same single node with a GPU NVIDIA Titan Black, 2880 Cuda cores at 889 MHz and 6GB for global memory; c) a single node with Intel Xeon Phi 7250 wich contains 68 cores at 1.40 GHz and 128GB for main memory, MCDRAM configured as cache mode; and d) a single node with IBM Power System S822LC (8335-GTB) based on POWER 8 (16 cores at 3.259 GHz with 3.857 GHz turbo performance) e) the same single node with NVIDIA NVLink connecting a GPU NVIDIA Tesla P100 (3584 Cuda cores at 1.328 GHz and 16GB for global memory).

The parallel algorithm was validated by comparing the solution obtained for different datasets with the serial program, showing a complete match between the original and parallel version. In order to check the parallel approach performance six datasets representing different networks was used, named net_i, with $i \in 1, 2, ..., 6$. The networks net_1 and net_4 have 2000 nodes/vertices, the networks net_2 and net_5 have 3000 nodes and the networks net_3 and net_6 have 4000 nodes. The networks net_1, net_2 and net_3 have approximately $n \times 20$ edges, and the datasets net_4, net_5 and net_6 have approximately $n \times 10$ edges. The first three networks ($net_1/_2/_3$) have a few number of communities with large amounts of vertices for each one, while the last ones, in contrast, have many communities, but with a few nodes.

Table I shows the elapsed time (wall-clock time) for the six networks used running on two different CPUs. The first 6 rows on table shows the performance over a single x86 node (Intel Xeon E5-2630) and the last 6 rows shows the performance over a single node with IBM Power8. Both use the same source-code implemented on C language with OpenMP. It is possible to see the worst performance using the parallel version with just one thread when comparing to the original algorithm, due to the new procedure responsible for computing the distances between adjacent nodes, according to describe in section IV. This new routine is worst than the former and serial one, however, it allows code parallelization, usually limited in a naive parallelism over the original version. Thus enabling desirable high scalability, possible to check by the time decay considering the parallel computational time from one thread elapsed time. The time decay occurs in a similar way for all datasets tested.

TABLE I
COMPUTATIONAL WALL TIME ON TWO DIFFERENT CPUS (INTEL XEON E5-2630V4 AND IBM POWER8) - ARTIFICIAL NETWORKS

	net_1	net_2	net_3	net_4	net_5	net_6
Serial	12.51	35.9	77.61	22.09	71.15	174.08
1	23.91	67.77	139.95	37.31	117.22	268.35
2	12.5	34.96	74.59	19.75	62.47	156.98
4	8.84	29.01	53.33	12.51	50.38	99.95
8	3.65	16.5	31.07	6.41	26.89	60.23
16	2.43	6.23	16.42	3.61	10.91	29.92
Serial	6.44	18.71	39.33	10.79	33.67	81.31
1	18.52	54.01	114.99	26.90	81.99	186.50
2	9.42	27.03	71.27	14.51	47.63	123.44
4	4.92	14.00	44.90	7.21	25.85	26.84
8	2.94	7.79	22.17	4.92	14.97	13.77
16	2.14	5.34	11.13	3.87	12.81	7.87

Table II shows the elapsed time evaluated on hardware accelerators for the same set of networks (except net_6). The first 7 rows shows performance running natively on a single Intel Xeon-Phi node (Intel Xeon Phi 7250) using the same C source-code with OpenMP used for CPUs. The last two rows shows the elapsed time over GPUs running a CUDA source-code, where GPU_1 means the NVIDIA Titan Black and GPU_2 the NVIDIA P100. The wall-clock times show similar pattern decay and scalability for Xeon Phi comparing to CPUs on table I. The GPU algorithm version shows the best overall performance in both GPUs, especially in NVIDIA P100.

Figure 2 shows the speedup over net_1 and net_4 (both with 2000 vertices) for all different computational resources used over the serial version (original version) running on the Intel Xeon E5-2630 as reference value. The X axis varies with 2^i, where $i = 0$ to 6. It is possible to check the scalability for all datasets, showing best performance for Intel Xeon E5-2630 than Intel Xeon Phi for same processor/core count, due to clock speed processor. The GPU shows best overall performance achieving a speedup up to 54.01 for the best case using GPU_2 (NVIDIA Tesla P100) with net_4. The best performance obtained with the parallel version

TABLE II

COMPUTATIONAL WALL TIME FOR HARDWARE ACCELERATORS (INTEL XEON-PHI, NVIDIA TITAN BLACK AND P100) - ARTIFICIAL NETWORKS

Threads	net_1	net_2	net_3	net_4	net_5
1	63.72	184.73	379.77	98.68	312.96
2	32.11	94.55	197.71	50.61	158.98
4	17.43	50.97	103.52	27.57	88.56
8	8.64	25.82	52.20	14.04	43.49
16	4.46	13.19	26.49	7.17	22.46
32	2.48	6.97	13.55	3.81	11.34
64	1.37	3.68	7.31	2.16	6.11
GPU_1	0.55	1.27	2.62	0.84	2.16
GPU_2	0.34	0.56	1.01	0.41	0.92

running on Intel Xeon or Xeon-Phi is below a desirable linear speedup, as it is possible to check on the same figure. Furthermore, similar results occurs for datasets net_2 and net_5 in figure 3, where GPU runs achieves a speedup 77.76 as best performance, with net_5 using GPU_2. In figure 4 the best speedup performance for net_3 is 77.14 also using GPU_2. The results show that larger data sets produce better performance gains on GPU. The speedup on the Power8 single node was not shown to improve the figure visibility, since the scalability is similar comparing to Intel-Xeon.

By comparing the two types of datasets (with small or large communities), it is possible to check that the results with smaller communities achieved a better performance. These behaviors are noted to a less extent for CPU multi thread runs, i.e., larger data sets with smaller communities have shown just a slight improvement on the speedup for the CPU parallel version.

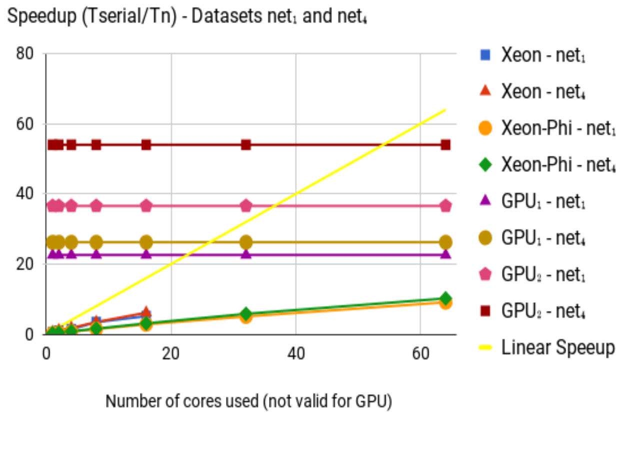

Fig. 2. Speedup values found (T_{serial}/T_n)

Is remarkable how the same parallel algorithm strategy produces a performance gain in different computational resources. However, the CPU parallel version needs future effort to improve vectorization on AVX-512, especially for Xeon Phi, to improve speedups. Furthermore, the GPU parallel version needs better effort to improve coalescent memory access and a new optimization in reduction procedures.

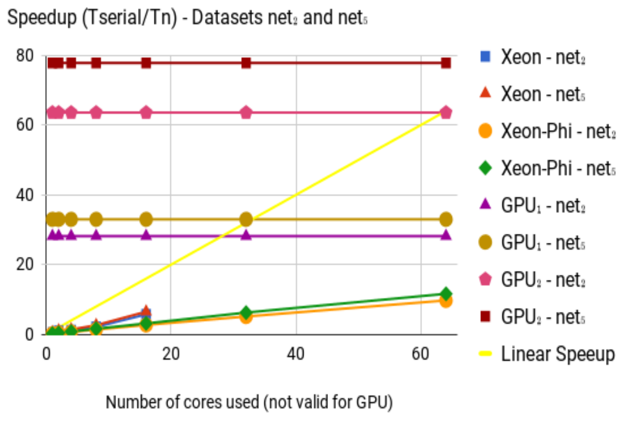

Fig. 3. Speedup values found (T_{serial}/T_n)

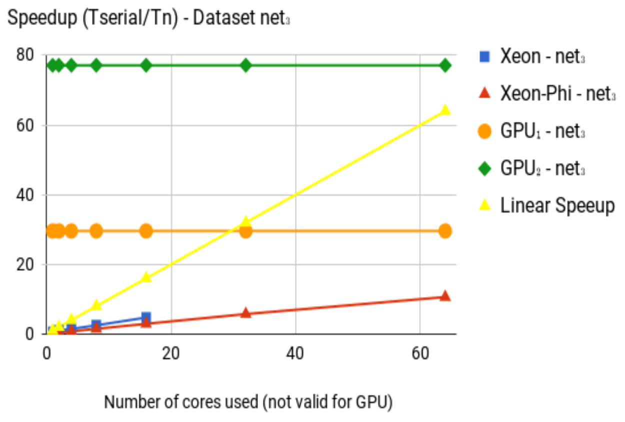

Fig. 4. Speedup values found (T_{serial}/T_n) for network net_3

The GPU parallel version takes about 98% of the time in computing routines and 2% in data transfers between the host and the device memory (and vice-versa). There are four CUDA kernel implemented, all of them are called by the host for each iterative loop until convergence: a) initParticlesGPU: responsible for initiating the distances between all vertices to zero ($O(N)$); b) coreGPUa: evaluates the distance matrix $O(N^2)$; c) coreGPUb: a reduction procedure performed in all vertices. It summarizes the distance matrix by computing the attraction and repulsion between adjacent and non-adjacent nodes, respectively ($O(N^2)$); and, d) upPositionsGPU: updates particle positions according to attraction and repulsion, adding distance factor on the three-dimensional particle coordinates ($O(N)$).

Table III shows instrumentation data results over GPU kernels evaluated on NVIDIA Titan Black (GPU_1). The most demanding kernel task (coreGPUb) is $O(N)$, in which N is the number of particles/nodes. In the procedures all tasks evaluate an inner loop over N. Thus, the complete routine is also $O(N^2)$. The small number of tasks (N) in contrast to the CoreGPUa routine is clearly a bottleneck for this procedure, especially when using an accelerator device which works better with massive parallelization like GPUs. This possible problem causes a small number of Instructions Per Cycle (IPC) metric and a flat rate for achieved occupancy. Future improvements to overcome this limitation are under investigation. The second demanding kernel (coreGPU2a) performs NxN concurrent tasks, each one evaluating a distance between two nodes. There are one order more tasks, performing less complex computation, which is desirable for a GPU device. The IPC and occupancy show the best performance. However, both routines show similar ratios for Streaming Multiprocessor (SM) activity.

TABLE III

GPU PERFORMANCE EVALUATION OVER ARTIFICIAL NETWORKS - MEAN VALUES

Kernel ID	Time(%)	SM activity(%)	IPC	Achieved Occupancy
coreGPUb	68.98	92.5	0.084	17.7
coreGPUa	27.8	99.9	0.554	46.7
upPositionsGPU	0.1	42.4	0.175	14.2
initParticlesGPU	0.02	34.8	0.113	16.9

In order to check only the scalability over an incremental number of threads, the elapsed time for parallel CPU version with just one thread was taken as a speedup reference. Figures 5 and 6 show the scalability for the datasets with larger communities and small communities evaluated on the Intel Xeon E5-2630 and the Intel Xeon Phi 7250. The results indicate better scalability for Intel accelerator Xeon Phi than the traditional dual multi-core CPU. The MCDRAM configured as a speed cache memory probably is the primary cause for the best scalability. A linear speedup line is also shown for comparative purposes. For figures 5 and 6 the results with Power8 was not shown again, since the speedup is very similar to Intel-Xeon, and the corresponding lines on the figure could be very difficult to visualize.

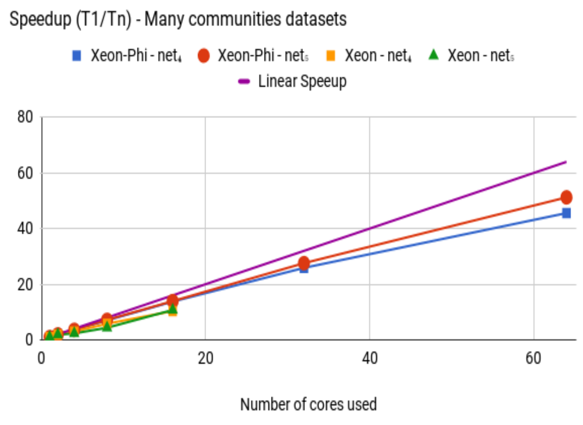

Fig. 5. Scalability for few communities datasets

V. CONCLUSIONS

In this paper, we proposed and evaluated parallel algorithms for community detection. Specifically, we have designed CPU and GPU versions of the community detection method proposed in [10]. Results showed that the parallel algorithms are equivalent to the original particle method regarding their outcome. As stated before, the particle method in its original formulation cannot be straight applied to large-scale networks owning to its high computational cost. Here, this main drawback was overcome by designing a novel parallel algorithm.

The new algorithm replaced the main procedure responsible for evaluating the distances between all nodes to each other, which was a source of concurrent race conditions, requiring critical sections to avoid it. The new matrix introduced to evaluate the distances allowed a massive parallelism, albeit decreasing the performance in a serial run.

The new parallel version has shown good scalability with a speedup near to 10 for 16 threads in a traditional dual multicore CPU and up to 50 times for 64 threads in a Intel Xeon-Phi hardware accelerator, taking the parallel version with one thread as base value for speedup evaluations. In comparison to the serial version, as the baseline value, the speedups are more modest reaching a speedup of 6.5 with dual multicore CPU using 16 threads and about 10.6 for 64 threads with a Intel Xeon Phi. One single core in Intel Xeon-Phi has less computing power than a single core in the Xeon E5-2630 v4, where the serial version was evaluated, which could explain the modest speedup even with 64 threads. Both CPU versions could improve performance with vectorization using AVX vector units. A same source-code was evaluated on a single node IBM Power8 with similar results comparing to Intel-Xeon. The GPU version, with same parallelism strategy, achieved a speedup range between 22.6 to 33.0 for the worst and best case on NVidia Titan Black and 33.6 and 77.8 on NVidia P100. Thus, according to our experiments, it is the best solution to be used in the code optimization, considering the cost/performance obtained, especially compared with Xeon-Phi.

These initial optimization experiments show the viability of the portable parallel strategy developed to create the massively concurrent tasks for different parallel computing

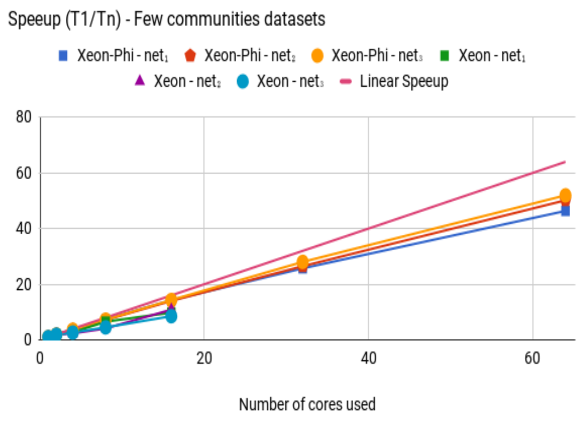

Fig. 6. Scalability for many communities datasets

resources. Future improvements on this parallel code are expected to improve even more the performance for similar shared memory machines. The main issues rely on improve SIMD vectorization for AVX vector units, and also improve a reduction procedure for GPU routine which summarizes the distance matrix (named CoreGPUb) and improves a coalescent memory access.

Future developments also include a distributed memory version using distinct hardware like Intel accelerators and GPUs, and new features like a parallel graph partition algorithm adapted to the same hardware used to compute the particle dynamics. The extension of the model to deal with directed and weighted graphs are also planned. Finally, the integration of the equations using lower-cost computational methods for N-body problem, which is commonly used in molecular dynamics problems, will also be scrutinized in our specific community detection scenario.

ACKNOWLEDGMENT

This work was partially supported by the FAPESP (Sao Paulo Research Foundation) and CNPq (National Council for Scientific and Technological Development in Brazil).

REFERENCES

[1] L. F. Costa, O. N. Oliveira, G. Travieso, F. A. Rodrigues, P. R. Villas-Boas, L. Antiqueira, M. P. Viana, and L. E. C. Rocha, “Analyzing and modeling real-world phenomena with complex networks: a survey of applications,” *Advances in Physics*, vol. 60, pp. 329–412, 2011.

[2] M. A. Porter, J.-P. Onnela, and P. J. Mucha, “Communities in networks,” *Notices of the AMS*, vol. 56, no. 9, pp. 1082–1097, 2009.

[3] S. E. Schaeffer, “Graph clustering,” *Computer science review*, vol. 1, no. 1, pp. 27–64, 2007.

[4] M. E. Newman, “The structure and function of networks,” *Computer Physics Communications*, vol. 147, no. 1-2, pp. 40–45, 2002.

[5] M. Girvan and M. E. Newman, “Community structure in social and biological networks,” *Proceedings of the national academy of sciences*, vol. 99, no. 12, pp. 7821–7826, 2002.

[6] S. Fortunato, “Community detection in graphs,” *Physics reports*, vol. 486, no. 3, pp. 75–174, 2010.

[7] L. Danon, A. Diaz-Guilera, J. Duch, and A. Arenas, “Comparing community structure identification,” *Journal of Statistical Mechanics: Theory and Experiment*, vol. 2005, no. 09, p. P09008, 2005.

[8] S. Cafieri, P. Hansen, and L. Liberti, “Locally optimal heuristic for modularity maximization of networks,” *Physical Review E*, vol. 83, no. 5, p. 056105, 2011.

[9] D. Aloise, G. Caporossi, P. Hansen, L. Liberti, S. Perron, and M. Ruiz, “Modularity maximization in networks by variable neighborhood search.” *Graph Partitioning and Graph Clustering*, vol. 588, p. 113, 2012.

[10] M. G. Quiles, E. E. Macau, and N. Rubido, “Dynamical detection of network communities,” *Scientific reports*, vol. 6, p. 25570, 2016.

[11] C. Wickramaarachchi, M. Frincu, P. Small, and V. K. Prasanna, “Fast parallel algorithm for unfolding of communities in large graphs,” in *High Performance Extreme Computing Conference (HPEC), 2014 IEEE*. IEEE, 2014, pp. 1–6.

[12] H. Meyerhenke, P. Sanders, and C. Schulz, “Parallel graph partitioning for complex networks,” *IEEE Transactions on Parallel and Distributed Systems*, 2017.

[13] D. LaSalle and G. Karypis, “Multi-threaded modularity based graph clustering using the multilevel paradigm,” *Journal of Parallel and Distributed Computing*, vol. 76, pp. 66–80, 2015.

[14] J. Barnes and P. Hut, “A hierarchical o (n log n) force-calculation algorithm,” *nature*, vol. 324, no. 6096, pp. 446–449, 1986.

[15] L. Greengard, *The rapid evaluation of potential fields in particle systems*. MIT press, 1988.

[16] J. A. Board Jr, Z. S. Hakura, W. D. Elliott, and W. T. Rankin, “Scalable variants of multipole-accelerated algorithms for molecular dynamics applications,” Technical Report TR94-006, Electrical Engineering, Duke University, Tech. Rep., 1994.

[17] J. Leskovec and R. Sosič, “Snap: A general-purpose network analysis and graph-mining library,” *ACM Transactions on Intelligent Systems and Technology (TIST)*, vol. 8, no. 1, p. 1, 2016.

[18] O. A. R. Board, “Openmp application program interface version 3.0,” http://www.openmp.org/mp-documents/spec30.pdf, 2008.

[19] NVIDIA, “Cuda c programming guide,” http://docs.nvidia.com/pdf/CUDA_C_Programming_Guide.pdf, 2017.