Journal of Supercomputing manuscript No.
(will be inserted by the editor)

Gillespie's Stochastic Simulation Algorithm on MIC Coprocessors

Andrea Tangherloni · Marco S. Nobile · Paolo Cazzaniga · Daniela Besozzi · Giancarlo Mauri

Received: date / Accepted: date

Abstract To investigate the behavior of biochemical systems, many runs of Gillespie's Stochastic Simulation Algorithm (SSA) are generally needed, causing excessive computational costs on Central Processing Units (CPUs). Since all SSA runs are independent, the Intel Xeon Phi coprocessors based on the Many Integrated Core (MIC) architecture can be exploited to distribute the workload. We considered two execution modalities on MIC: one consisted in running exactly the same CPU code of SSA, while the other exploited MIC's vector instructions to reuse the CPU code with only few modifications. MIC performance was compared with Graphics Processing Units (GPUs), specifically implemented in CUDA to optimize the use of memory hierarchy. Our results show that GPU largely outperforms MIC and CPU, but required a complete redesign of SSA. MIC allows a relevant speedup, especially when vector instructions are used, with the additional advantage of requiring minimal modifications to CPU code.

Keywords High performance computing · MIC · GPU · Parallel computing · Stochastic simulation algorithm · Stochastic modeling

A. Tangherloni, M. S. Nobile, D. Besozzi, G. Mauri
Dept. Informatics, Systems and Communication, University of Milano-Bicocca (Italy)
E-mail: {tangherloni/nobile/besozzi/mauri}@disco.unimib.it

P. Cazzaniga
Dept. Human and Social Sciences, University of Bergamo (Italy)
E-mail: paolo.cazzaniga@unibg.it

M. S. Nobile, P. Cazzaniga, D. Besozzi, G. Mauri
SYSBIO.IT Centre of Systems Biology (Italy)

1 Introduction

Nowadays, the synergistic integration between experimental laboratory research and computational analysis is deepening the comprehension of the functioning of complex biological systems [1]. In this context, mechanism-based mathematical models and simulation algorithms play a fundamental role in efficiently and reliably reproducing the temporal evolution of the system under investigation, assuming either the stochastic or the deterministic modeling approach [24]. One of the most used method to analyze the dynamics of stochastic mechanistic models is Gillespie's Stochastic Simulation Algorithm (SSA), which is based on the stochastic formulation of chemical kinetics [7].

SSA was proven to be equivalent to the Chemical Master Equation [8], so that it is able to achieve an exact description of the temporal evolution of the system. In general, large batches of independent SSA runs need to be executed to determine the distribution of the system states, to collect statistically significant results or to investigate the behavior of the system under different conditions (as in the case of, e.g., parameter estimation [19] or sensitivity analysis [21]). For computational analyses of this type, the computational burden can quickly become excessive when SSA is executed on Central Processing Units (CPUs).

However, since all SSA runs can be executed independently, parallel architectures can be exploited to distribute the workload and reduce the overall running time. Algorithm parallelization is usually realized by means of multi-threading [23], distributed computing on clusters [10], custom circuitry produced with Field Programmable Gate Array (FPGA) [14] or general-purpose Graphics Processing Unit (GPU) computing [20,17,18]. These parallel technologies generally require a custom implementation of the algorithm, since most of the time CPU code cannot be directly ported on the parallel architecture; in addition, distributed architectures need the definition of an appropriate scheduler to manage the parallel execution of processes. An alternative solution to algorithm parallelization is represented by the family of Intel Xeon Phi coprocessors, based on the Many Integrated Core (MIC) architecture, which allow to directly compile and execute on the coprocessors the code implemented for Intel CPUs.

This work extends the analysis we previously presented in [4], by comparing the performance of MIC coprocessors with respect to both CPUs and GPUs. To this purpose we considered, on the one hand, two execution modalities of SSA for MIC: the first consisted in directly executing the same CPUs source code of SSA, while the second exploited vector instructions, a peculiar capability of MIC that allows to reuse any existing CPUs source code with only few modifications. On the other hand, we developed an *ad hoc* implementation of SSA for GPUs, optimizing the use of the memory hierarchy. To evaluate the running time of these sequential and parallel SSA implementations, we executed an increasing number of stochastic simulations of a mechanism-based model of prokaryotic gene regulation [16]. A family of synthetic models with different size (i.e., different number of molecular species and reactions) was then used as test case to investigate the benefits of exploiting MIC's vector instructions and offload capabilities. In addition to the computational time, we evaluate the costs and power consumption of the hardware employed, and discuss the effort to port the existing code on parallel architectures.

Previous works already focused on the comparison of the performance of Intel Xeon Phi coprocessors against other parallel architectures, showing different results according to the specific problem under investigation. In the context of the simulation of spin systems, a comparison between Intel Xeon Phi 5110P and Nvidia Tesla K20s video card was presented in [2], highlighting that a careful implementation of the C code allows the MIC to compete with the GPU. On the contrary, in [5] it was shown that a Nvidia Tesla K20x outperforms an Intel Xeon Phi 5110P for the parallelization of non-bonded electrostatic computation for Virtual Screening; this work pointed out, in particular, the importance of OpenMP source code optimization. The work presented in [9] described the performance comparison among a multi-core Intel CPU, an Nvidia Tesla K20c GPU and an Intel Xeon Phi 7120P coprocessor for the execution of a tracking algorithm based on the Hough transform: the results highlighted that in this case the CPU performs better than both GPU and MIC coprocessors. Moreover, the authors suggested that an implementation with offloaded calculations to the coprocessors might help in achieving better performances. A multi-threaded version of an algorithm to tackle the tensor transpose problem was then presented in [13]. In this case, the multicore CPU and the MIC achieved a relevant speedup with respect to the GPU, since the optimization of L1 cache is easier than the implementation of a coalesced global memory access on the GPU. As a final example, a comparison of the acceleration on an Intel Xeon Phi 5110P and a Nvidia Tesla K20x for protein docking calculation based on the fast Fourier transform was introduced in [22]. The GPU resulted to be 5 times faster than the MIC, considering the comparable implementation costs required by these architectures.

In this work we show that, for the problem of executing increasing batches of SSA runs, GPU largely outperforms the other architectures thanks to its large number of computing units. Nevertheless, GPU required a complete redesign and specific programming of this simulation algorithm. On the other side, we show that MIC coprocessors allow a relevant speedup, especially when the simulated biochemical system is large and MIC vector instructions are used, and have the additional advantage of requiring minimal modifications to CPU code.

The paper is structured as follows. In Section 2 we briefly introduce the formalism of mechanism-based stochastic models and the functioning of Gillespie's Stochastic Simulation Algorithm. We also provide the definition of the biochemical systems that will be considered as case studies to determine the computational performances of CPU, GPU and MIC. In Section 3 we show the speedup obtained by Xeon Phi and Nvidia Tesla K80 with respect to CPU, and discuss the main benefits of MIC architecture. Finally, in Section 4 we conclude the work with some final remarks about the selection of the proper architecture to execute SSA.

2 Mechanism-based Modeling and Gillespie's Stochastic Simulation Algorithm

According to the stochastic formulation of chemical kinetics [7], a mechanism-based model of a biochemical system can be formalized by specifying the set $\mathcal{S} = \{S_1, \ldots, S_N\}$ of molecular species occurring in the system, and the set $\mathcal{R} = \{R_1, \ldots, R_M\}$ of chemical reactions taking place between these species. Each reaction can be formally defined as R_j: $\sum_{i=1}^{N} \alpha_{ji} S_i \xrightarrow{c_j} \sum_{i=1}^{N} \beta_{ji} S_i$, where $\alpha_{ji}, \beta_{ji} \in \mathbb{N}$ are the stoichiometric coefficients associated, respectively, to the i-th reactant and to the i-th product of the j-th reaction, with $i = 1, \ldots, N$, $j = 1, \ldots, M$. The value $c_j \in \mathbb{R}^+$ is the so-called stochastic constant, encompassing all physical and chemical properties of reaction R_j.

Mechanism-based stochastic models are able to account for any macroscopic effect in the behavior of biochemical systems caused by biological noise, that is, the random collision and reaction events among the molecular species that are present in very low amounts inside cells. In these cases, the classical deterministic modeling approach can fail in capturing the effects of biochemical stochastic processes [24]. The seminal procedure used to describe the dynamics of stochastic models is Gillespie's Stochastic Simulation Algorithm (SSA) [7], which is able to realize an exact reproduction of the temporal evolution of biochemical networks, under the following assumptions: (i) reactions and species are contained within a single volume, whose physical conditions (e.g., pressure, temperature) remain constant during the whole time of simulation; (ii) the reaction volume is well-stirred, that is, molecules are uniformly distributed in space; (iii) the amount of each molecular species S_i is discrete, i.e., it is represented by an integer number $x_i \in \mathbb{N}$.

Briefly, SSA works as follows. Given the state of the system at time t, represented by the vector $\mathbf{x} = \mathbf{x}(t) \equiv (x_1(t), \ldots, x_N(t))$, SSA first identifies the reaction to execute in the next time interval $[t, t+\tau)$. To this aim, the probability of each reaction R_j to occur in the next infinitesimal time step $[t, t+dt)$ has to be evaluated. This probability is proportional to the so-called *propensity function* of reaction R_j, defined as $a_j(\mathbf{x}) = c_j \cdot d_j(\mathbf{x})$, where $d_j(\mathbf{x})$ is the number of distinct combinations of the reactant molecules in R_j occurring in state $\mathbf{x}$. Then, SSA computes the time τ before a reaction takes place: $\tau = \frac{1}{a_0(\mathbf{x})} \ln\left(\frac{1}{\rho_1}\right)$, where $a_0(\mathbf{x}) = \sum_{j=1}^{M} a_j(\mathbf{x})$ and ρ_1 is a random number sampled in [0,1] with a uniform probability. The reaction R_j to be actually executed is then chosen by taking the smallest integer in $[1, M]$ such that $\sum_{j'=1}^{j} a_{j'}(\mathbf{x}) > \rho_2 \cdot a_0(\mathbf{x})$, where ρ_2 is a second random number sampled in [0,1] with a uniform probability. The interested reader is referred to [7] for more details about SSA.

In what follows, we will consider two different test cases of mechanism-based stochastic models to the aim of evaluating the computational performance of MIC, CPU and GPU when executing large batches of SSA runs.

The first test case is a stochastic model describing the gene expression regulation network in prokaryotes (PGN, in short). In the PGN model, a gene is transcribed into messenger RNA and translated into a protein. The gene itself is inhibited by the binding with a dimer of the protein. Full details of this model, consisting in 5 species and 8 reactions, can be found in [16].

The second test case is a family of synthetic stochastic models of increasing size (SynSM, in short), which are randomly generated according to the methodology proposed in [20]. Namely, SynSM are characterized by a number of species N and of reactions M ranging from (20×20) to (240×240); the values of the stochastic constants are randomly sampled with uniform distribution in $(0, 1)$. SynSM are specifically exploited here to evaluated the impact of the size of the model on the performance of the MIC architecture.

3 Computational Results

In this section we start by showing the comparison between the computational performance achieved with Intel Xeon Phi coprocessors, CPUs and GPUs for the execution of an increasing number of SSA runs for the PGN model. Then, we investigate some specific MIC features, such as vector instructions and offload capability, to simulate a set of SynSM with increasing size. To evaluate the computational costs of running sequential and parallel SSA runs, we exploited GALILEO, a supercomputer created by the Italian consortium CINECA that combines multiple state-of-the-art accelerators. Namely, it consists of: (*a*) 768 Intel Xeon Phi 7120P coprocessors, each one with 61 cores, 1.238 GHz; (*b*) 516 compute nodes, each one equipped with 2 Intel Xeon Haswell E5-2630 v3 (8 cores, 2.40 GHz), for a total of 8256 cores; (*c*) 20 Nvidia Tesla K80 GPUs (4992 cores with a dual-GPU design, 560 MHz). Thanks to its peculiar hybrid architecture, this supercomputer represents an ideal machine for a direct comparison between the three architectures considered here.

To exploit GALILEO, we developed three different implementations of SSA. The first implementation was designed for the x86 architecture, that is, the Intel Xeon Haswell E5-2630 v3 and the Intel Xeon Phi 7120P. The second implementation consisted in reusing the CPU code, modified to exploit the advantages of MIC's vector instructions. The third version was specifically developed for the Nvidia Tesla K80 architecture, and optimized to fully exploit the memory hierarchy of the GPU: the state of the biochemical system is stored in the shared memory, while the matrices of stoichiometric coefficients are stored in the constant memory. All SSA implementations were written using the `C++` language and exploit the MRG32K32a pseudorandom numbers generator [12], which is available in the Intel Math Kernel Library for the CPU and MIC, and in the CURAND Library for the GPU.

Simulation of the PGN model. For each architecture and its respective SSA implementation, every SSA run of the PGN model was executed for a total simulation time of $t = 80$ time units, storing the amount of all molecular species along 16 time points of the dynamics. Figure 1 reports the running time (in seconds) required to execute increasing batches of SSA runs on the

MIC (light gray bars), MIC using vector instructions (dark gray bars), CPU (black bars) and GPU (white bars).

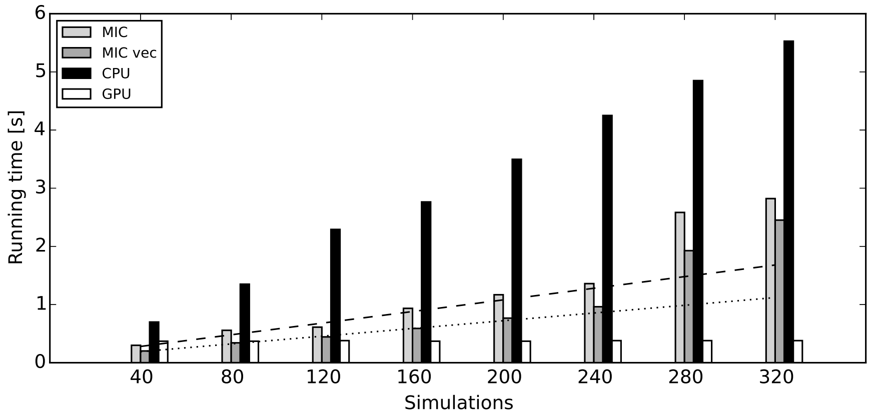

Fig. 1 Comparison of the running time to execute an increasing number of SSA runs of the PGN model on the three architectures: MIC (light gray bars), MIC using vector instructions (dark gray bars), CPU (black bars) and GPU (white bars). The dotted and dashed lines represent the linear regression for the estimated running time on the MIC with and without the use of vector instructions, respectively, assuming a number of cores larger than 60. The estimated values highlight the actual drop of performances when the number of parallel threads is larger than 240 (i.e., the maximum number of threads concurrently executable on the MIC). When more than 80 simulations are performed the GPU outperforms the other architectures, and the running time remains basically constant up to 320 parallel SSA runs, thanks to the large number of available cores.

These results show that the CPU running time linearly increases with the number of simulations, and that the GPU largely outperforms the other architectures. In particular, the GPU running time remains almost constant while increasing the number of parallel SSA runs: this is due to the high number of cores available on the GPU used in this work, which allows to distribute the simulations over individual computing units. During this batch of tests, anyway, the GPU Nvidia K80 was far from a full usage of its computing and memory resources. Thus, although for 320 parallel SSA runs the GPU achieved a speedup of 15× with respect to the CPU, we argue that its computational performances would be even better by running a larger number of simulations. It is also worth noting that, despite the branching of CUDA threads due to the stochastic nature of all (independent) simulations, the simplicity of SSA makes coarse-grain simulation perfectly suitable for GPU's architecture.

In the case of MIC architecture we observed an acceleration with respect to the sequential SSA execution on the CPU. According to our results, in the case of 240 SSA runs, the speedup achieved by this architecture—without the use of vector instructions—is 4.1× with respect to the CPU (Figure 1, light gray bars). Since Xeon Phi coprocessors can execute up to 4 concurrent threads on each of the 60 available cores, the acceleration provided by MIC scales up to 240 simulations only. In order to highlight this limitation, we estimated by linear regression the theoretical running time of MIC to execute 280 and 320 simulations, using the running times obtained in the case of $1, \ldots, 240$ simulations. In Figure 1 we show the estimated running times with and without the use of MIC's vector instructions (dotted and dashed lines, respectively). We observe that, using both SSA execution modalities on MIC, the measured running times with 280 and 320 simulations are higher than expected because of the limitation of resources.

The use of MIC vector instructions allows an additional improvement of computational performances (Figure 1, dark gray bars), and highlights that this peculiar capability of MIC is necessary to fully leverage the computational power of Intel Xeon Phi coprocessors. More precisely, in the case of 15 simultaneous simulations (Figure 2), the running time of the MIC with and without the use of vector instructions changes from 0.15 to 0.27 seconds, respectively, corresponding to an overall reduction of about 42% when vector instructions are enabled. Figure 2 also shows the break-even between MIC, CPU and GPU. It is worth noting that when only a few SSA runs need to be executed, the CPU outperforms both GPUs and MICs, thanks to its higher clock frequency. The MIC is faster than the CPU when about 15 SSA runs are performed (10 when using vector instructions). The break-even between CPU and GPU is around 20 simulations.

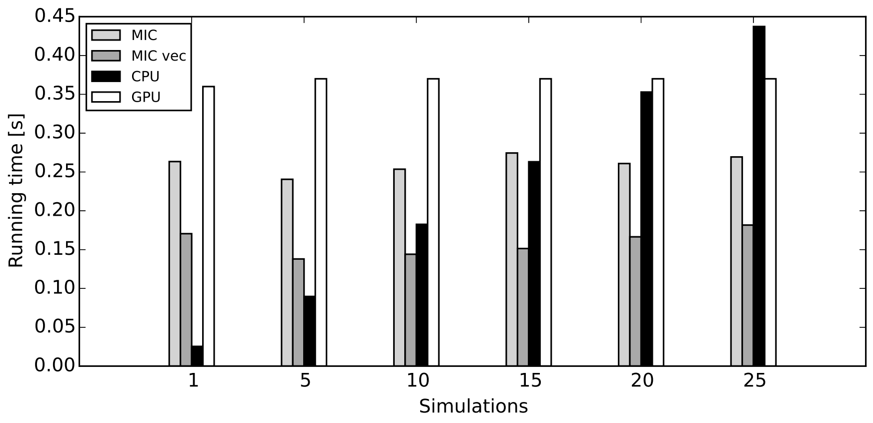

Fig. 2 Break-even of the running time to execute a limited number of SSA runs of the PGN model on the three architectures: MIC (light gray bars), MIC using vector instructions (dark gray bars), CPU (black bars) and GPU (white bars). When only a few simulations are executed, the CPU outperforms the other architectures thanks to its higher clock frequency. The MIC becomes advantageous with respect to CPU execution when about 10 simulations are performed, using vector instructions. Without vectorialization, the MIC is more performant than CPU when at least 15 simulations are run. Similarly, the GPU outperforms the CPU when about 20 simulations are executed.

Simulation of SynSM. In the second batch of tests, we specifically focused on MIC architecture to investigate the impact of the size of the simulated model. To evaluate the performances of the Xeon Phi coprocessor, we randomly created a set of synthetic models of increasing size, i.e., 20×20, 40×40, 80×80, 160×160, whose dimensions correspond to the number of chemical species and the number of reactions, respectively. The models were simulated using the SSA implementation, with and without the vector instructions enabled. For every model, each SSA run was executed for a total simulation time of $t = 100$ time units, storing the amount of 10 molecular species along 11 time points of the dynamics.

The results, summarized in Figure 3, show two relevant trends. First, although the running time constantly increases, it is not directly proportional to the size of the model: a single simulation takes 0.11 seconds for size 20×20, and 0.74 seconds for size 160×160. This means that a biochemical system that is 8 times bigger (160×160 vs. 20×20) requires a running time of $6.7\times$. If we consider the case of 240 simulations, the running times are 0.6 and 1.5 seconds, respectively, corresponding to an increment of just $3.4\times$. The second observation is that, by using the vector instructions, the scalability is strongly improved: bars with cross in Figure 3 show that, in the case of 240 simulations, if we compare sizes 20×20 and 160×160, we see that the running time of the largest model is only $2.2\times$ greater than the smallest one. In the same situation, but without using vector instructions (empty bars), the running time of the largest model is only $3.38\times$ greater than the smallest one. Stated otherwise, the use of vector instructions achieves better computational performances when the size of SynSM increases.

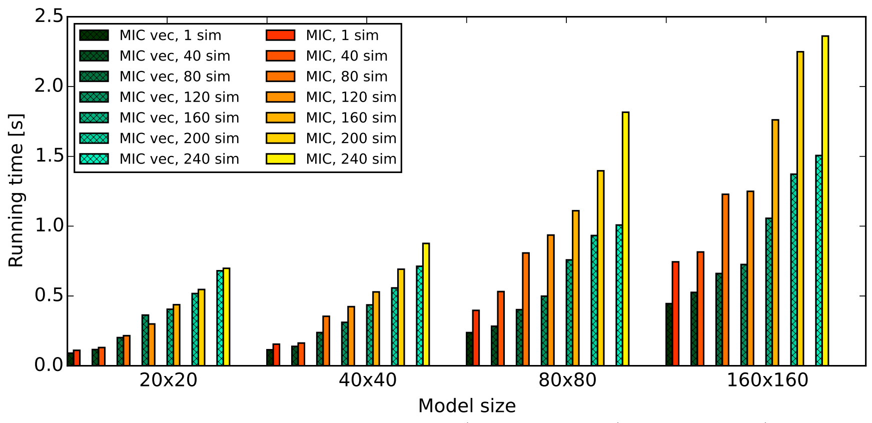

Fig. 3 Comparison of MIC running times with (bars with cross) and without (empty bars) the use of vector instructions to execute an increasing number of stochastic simulations (from 1 to 240) of synthetic models, having a number of species N and of reactions M equal to $(20 \times 20), (40 \times 40), (80 \times 80), (160 \times 160)$.

MIC's offload capability. As last test, we exploited the explicit offload capabilities of the Xeon Phi coprocessors. In order to do so, we modified the SSA implementation in two ways: (*i*) we linearized all data structures, so that the arrays could be automatically transferred to the Xeon Phi; (*ii*) we added the offload compiler pre-directives, which mark the regions of the source code that must be offloaded (in our case, the initialization of the system, the calculation of the propensity functions and the system's states update). According to our results, even in the case of "large" stochastic models of biochemical systems (e.g., 240×240 chemical species and reactions) the offload does not provide a relevant speedup. The rationale behind this is that SSA is a relatively simple algorithm, so that the reduced number of calculations distributed on MIC's cores—combined with the overhead due to MIC initialization and data transfers—affect the overall performances, making CPU more efficient for this task. Although it is possible that MIC could provide a more relevant speedup to simulate larger models, it is usually not trivial to define stochastic models of such complexity, due to the lack of quantitative parameters and of initial molecular amounts that usually affect the availability of large scale mechanistic models of biochemical systems.

4 Conclusion

In this work we investigated the application of MICs for the stochastic simulation of biochemical systems. To this aim, we performed several tests and compared MIC with two alternative architectures: CPU and GPU. According to our results, MICs provide better performances than CPUs when more than 10 SSA runs are executed, although GPUs outperform both MICs and CPUs when more than 80 SSA runs are executed. It is worth noting that the running time on GPUs remains constant until computing resources (i.e., the cores) are available; in the case of MIC's, the performances scale well until the maximum number of supported threads (i.e., 240) is reached. Moreover, MIC's performances in native mode are strongly improved when vector instructions are exploited, especially in the case of larger synthetic stochastic models. We also tested the *offload* capability of the Xeon Phi coprocessors, that is, the possibility of automatically distributing independent calculations over multiple threads. Our results showed that the offload of the SSA algorithm does not provide a speedup, since the portions of code that can be offloaded consist in a few instructions only; therefore, the offload parallelization is affected by overheads and data transfers.

The empirical analyses shown in this work might facilitate the selection of the proper parallel architecture and SSA implementation when a large number of mutually independent simulations are needed, as is the case of many computationally expensive tasks typically carried out for in-depth investigations of biological systems (see, e.g., [1,3,19,16,21]). However, we highlight that some additional issues should be considered for the selection of a proper parallel architecture. Specifically, we discuss hereby also the efforts of code porting, the power consumption of the three architectures and the financial costs items.

Concerning the cost of code porting, in the case of GPGPU computing on Nvidia video cards, the effort necessary to re-implement any algorithm using the CUDA programming technique is relevant. Since this issue would increase the time-to-market, it should be seriously taken into account. On the contrary, Xeon Phi coprocessors are supposed to be fully compatible with CPUs based on the x86 ISA; however, according to our experience, the code has to be slightly adapted in order to be correctly executed on MIC (considering the *native mode*). On the other hand, a comparison between the three architectures should also take into consideration the evaluation of costs, power consumption and theoretical peak performance. At the time of writing, the CPU Intel Xeon Haswell E5-2630 v3 has a cost of around €600, with a power consumption of 80W and theoretical peak performance in double precision of about 500 GFlops, which is only reachable when fused multiply-add (FMA) and advanced vector instructions (AVX) are simultaneously exploited. The characteristics of the other devices are: €5800, 300W and 1208 GFlops for Intel Xeon Phi 7120P; €4200, 300W and 2910 GFlops for Nvidia Tesla K80. According to these data, the same theoretical peak of the Tesla K80 GPU can be achieved using either 6 CPUs or 3 Xeon Phi 7120P, with the consequent increment in terms of financial cost and power consumption. However, to fully leverage the computational power of the CPU, the implementation would require the extensive use of FMA, AVX and multi-threading, requiring relevant modifications of the code.

As a final remark, we highlight that the performance of GPUs and MICs are affected by the Error Correcting Code (ECC), used to avoid any error caused by natural radiations [6]. This functionality is enabled on both accelerators on the GALILEO supercomputer and introduces a relevant overhead, mainly due to bits verification. According to the tests performed by Fang *et al.*, the ECC on MIC coprocessors causes a bandwidth reduction greater than 20% [6]. Tesla GPUs exploit ECC over the whole memory hierarchy, including the global memory, L1 and L2 caches, and registers [15]. Also in the case of GPUs the bandwidth reduction is around 20% [11], so we expect that the speedup obtained using GPUs and MIC for stochastic simulations could be further improved by disabling the ECC.

Acknowledgements We acknowledge the CINECA award under the ISCRA initiative, for the availability of high performance computing resources and support.

References

1. Aldridge, B.B., Burke, J.M., Lauffenburger, D.A., Sorger, P.K.: Physicochemical modelling of cell signalling pathways. Nat. Cell Biol. **8**(11), 1195–1203 (2006)
2. Bernaschi, M., Bisson, M., Salvadore, F.: Multi-Kepler GPU vs. multi-Intel MIC for spin systems simulations. Comput. Phys. Commun. **185**(10), 2495–2503 (2014)
3. Besozzi, D., Cazzaniga, P., Pescini, D., Mauri, G., Colombo, S., Martegani, E.: The role of feedback control mechanisms on the establishment of oscillatory regimes in the Ras/cAMP/PKA pathway in *S. cerevisiae.* EURASIP J. Bioinform. Syst. Biol. **2012**(10) (2012)

4. Cazzaniga, P., Ferrara, F., Nobile, M.S., Besozzi, D., Mauri, G.: Parallelizing biochemical stochastic simulations: a comparison of GPUs and Intel Xeon Phi processors. In: V. Malyshkin (ed.) Proc. 13th Int. Conference on Parallel Computing Technologies (PaCT 2015), *LNCS*, vol. 9251, pp. 363–374 (2015)
5. Fang, J., Varbanescu, A.L., Imbernon, B., Cecilia, J.M., Perez-Sanchez, H.: Parallel computation of non-bonded interactions in drug discovery: Nvidia GPUs vs. Intel Xeon Phi. In: Proc. 2nd Int. Work-Conference on Bioinformatics and Biomedical Engineering (IWBBIO'14) (2014)
6. Fang, J., Varbanescu, A.L., Sips, H., Zhang, L., Che, Y., Xu, C.: Benchmarking Intel Xeon Phi to guide kernel design. Tech. Rep. PDS-2013-005, Delft University of Technology, The Netherlands (2013)
7. Gillespie, D.T.: Exact stochastic simulation of coupled chemical reactions. J. Phys. Chem. **81**(25), 2340–2361 (1977)
8. Gillespie, D.T.: A rigorous derivation of the chemical master equation. Physica A **188**(1), 404–425 (1992)
9. Halyo, V., LeGresley, P., Lujan, P., Karpusenko, V., Vladimirov, A.: First evaluation of the CPU, GPGPU and MIC architectures for real time particle tracking based on Hough transform at the LHC. J. Instrum. **9**(04) (2014)
10. Kent, E., Hoops, S., Mendes, P.: Condor-COPASI: high-throughput computing for biochemical networks. BMC Syst. Biol. **6**(1), 91 (2012)
11. Kraus, J., Pivanti, M., Schifano, S.F., Tripiccione, R., Zanella, M.: Benchmarking GPUs with a parallel Lattice-Boltzmann code. In: 25th Int. Symposium on Computer Architecture and High Performance Computing, pp. 160–167. IEEE (2013)
12. L'Ecuyer, P., Simard, R., Chen, E.J., Kelton, W.D.: An object-oriented random-number package with many long streams and substreams. Oper. Res. **50**(6), 1073–1075 (2002)
13. Lyakh, D.I.: An efficient tensor transpose algorithm for multicore CPU, Intel Xeon Phi, and NVidia Tesla GPU. Comput. Phys. Commun. **189**, 84–91 (2015)
14. Macchiarulo, L.: A massively parallel implementation of Gillespie algorithm on FPGAs. In: Int. Conference of the IEEE on Engineering in Medicine and Biology Society, pp. 1343–1346 (2008)
15. Nickolls, J., Dally, W.J.: The GPU computing era. IEEE Micro **30**(2), 56–69 (2010)
16. Nobile, M.S., Besozzi, D., Cazzaniga, P., Mauri, G., Pescini, D.: A GPU-based multi-swarm PSO method for parameter estimation in stochastic biological systems exploiting discrete-time target series. In: M. Giacobini, L. Vanneschi, W. Bush (eds.) Evolutionary Computation, Machine Learning and Data Mining in Bioinformatics. Proc. 10th European Conference, EvoBIO 2012, *LNCS*, vol. 7246, pp. 74–85 (2012)
17. Nobile, M.S., Besozzi, D., Cazzaniga, P., Mauri, G., Pescini, D.: cupSODA: a CUDA-powered simulator of mass-action kinetics. In: V. Malyshkin (ed.) Proc. 12th Int. Conference on Parallel Computing Technologies, *LNCS*, vol. 7979, pp. 344–357 (2013)
18. Nobile, M.S., Cazzaniga, P., Besozzi, D., Mauri, G.: GPU-accelerated simulations of mass-action kinetics models with cupSODA. J. Supercomput. **69**(1), 17–24 (2014)
19. Nobile, M.S., Cazzaniga, P., Besozzi, D., Pescini, D., Mauri, G.: Reverse engineering of kinetic reaction networks by means of Cartesian Genetic Programming and Particle Swarm Optimization. In: IEEE Congress of Evolutionary Computation, pp. 1594–1601 (2013)
20. Nobile, M.S., Cazzaniga, P., Besozzi, D., Pescini, D., Mauri, G.: cuTauLeaping: A GPU-powered tau-leaping stochastic simulator for massive parallel analyses of biological systems. PLoS ONE **9**(e91963) (2014)
21. Saltelli, A., Ratto, M., Andres, T., Campolongo, F., Cariboni, J., Gatelli, D., Saisana, M., Tarantola, S.: Global Sensitivity Analysis: The Primer. Wiley-Interscience (2008)
22. Shimoda, T., Suzuki, S., Ohue, M., Ishida, T., Akiyama, Y.: Protein-protein docking on hardware accelerators: comparison of GPU and MIC architectures. BMC Syst. Biol. **9**(Suppl 1), S6 (2015)
23. Tian, T., Burrage, K.: Parallel implementation of stochastic simulation of large-scale cellular processes. In: 8th Int. Conference on High-Performance Computing in Asia-Pacific Region, pp. 621–626 (2005)
24. Wilkinson, D.: Stochastic modelling for quantitative description of heterogeneous biological systems. Nat. Rev. Genet. **10**(2), 122–133 (2009)