Τίτλος Έργου: «StochSocS: Συστήματα σε Ψηφίδα για Παράλληλη Στοχαστική Προσομοίωση Βιολογικών Δικτύων στη Βιολογία Συστημάτων»

(κωδικός 3828 και Κ.Α. 70/3/12367)

ΠΑΡΑΔΟΤΕΟ 2.2

Τίτλος: "Αξιολόγηση Επιδόσεων Στοχαστικής Προσομοίωσης Βιομοντέλων Αυξανόμενης Πολυπλοκότητας"

ΑΘΗΝΑ

ΙΟΥΛΙΟΣ 2015

Ευρωπαϊκή Ένωση
Ευρωπαϊκό Κοινωνικό Ταμείο
ΕΠΙΧΕΙΡΗΣΙΑΚΟ ΠΡΟΓΡΑΜΜΑ
ΕΚΠΑΙΔΕΥΣΗ ΚΑΙ ΔΙΑ ΒΙΟΥ ΜΑΘΗΣΗ
επένδυση στην κοινωνία της γνώσης
ΥΠΟΥΡΓΕΙΟ ΠΑΙΔΕΙΑΣ ΚΑΙ ΘΡΗΣΚΕΥΜΑΤΩΝ
ΕΙΔΙΚΗ ΥΠΗΡΕΣΙΑ ΔΙΑΧΕΙΡΙΣΗΣ
ΕΣΠΑ 2007-2013
πρόγραμμα για την ανάπτυξη
ΕΥΡΩΠΑΪΚΟ ΚΟΙΝΩΝΙΚΟ ΤΑΜΕΙΟ
Με τη συγχρηματοδότηση της Ελλάδας και της Ευρωπαϊκής Ένωσης

Many-Core CPUs Can Deliver Scalable Performance to Stochastic Simulations of Large-Scale Biochemical Reaction Networks

Elias Kouskoumvekakis[1], Dimitrios Soudris[2], Elias S. Manolakos[1]
[1]Department of Informatics and Telecommunications, University of Athens, Greece
[2]School of Electrical & Computer Engineering, National Technical University of Athens, Greece
Email: {eliask, eliasm}@di.uoa.gr

***Abstract*—Stochastic simulation of large-scale biochemical reaction networks is becoming essential for Systems Biology. It enables the in-silico investigation of complex biological system dynamics under different conditions and intervention strategies, while also taking into account the inherent "biological noise" especially present in the low species count regime. It is however a great computational challenge since in practice we need to execute many repetitions of a complex simulation model to assess the average and extreme cases behavior of the dynamical system it represents. The problem's work scales quickly, with the number of repetitions required and the number of reactions in the bio-model. The worst case scenario s when there is a need to run thousands of repetitions of a complex model with thousands of reactions. We have developed a stochastic simulation software framework for many- and multi-core CPUs. It is evaluated using Intel's experimental many-cores Single-chip Cloud Computer (SCC) CPU and the latest generation consumer grade Core i7 multi-core Intel CPU, when running Gillespie's First Reaction Method exact stochastic simulation algorithm. It is shown that emerging many-core NoC processors can provide scalable performance achieving linear speedup as simulation work scales in both dimensions.**

Keywords— stochastic simulation algorithms; biochemical reaction networks; First Reaction Method; Intel SCC; many-core processors; Networks on Chip; parallel algorithms

I. INTRODUCTION

Systems Biology, a rapidly emerging important multi-disciplinary field, creates challenges not only for biologists but also for computer scientists and engineers. A major challenge is the demand to capture and analyze the stochastic dynamics of large-scale biochemical reaction networks commonly used to model the behavior of cellular systems. The ever increasing complexity of such bio-models, with thousands of biochemical reactions, creates a pressing need for efficient and accurate *in silico* stochastic simulation schemes that could be employed easily and routinely by any investigator on a modern computer of reasonable cost and power consumption. The computational demands for such simulations increase dramatically as bio-models scale to the level of whole cellular subsystems or whole organism metabolic networks [1].

When an investigator does not want to sacrifice simulation accuracy, there are two approaches to follow: The first employs deterministic methods based on ordinary differential equations (ODEs), while the second and more realistic tries to mimic the way nature works and account for intrinsic and extrinsic stochasticity ("noise") of biological systems by employing exact Stochastic Simulation Algorithms (SSA). SSAs employ Markov chain processes as the stochastic model for the biochemical reactions, in order to approximate their time evolution after a series of discrete steps. Then Monte Carlo experiments can be used to repeatedly execute these algorithms, while changing their initial conditions or reaction parameters in each repetition.

The most popular exact Stochastic Simulation Algorithms were introduced by D.T Gillespie, with the Direct Method (DM) SSA being the first proposed method [2]. For networks with m reactions the algorithm has time complexity in $O(m)$. Gillespie's First Reaction Method (FRM) is an equivalent later algorithm with the same complexity which is easier to parallelize for high performance. Gibson and Bruck have also introduced an alternative exact SSA, the Next Reaction Method (NRM) [3], enjoying a reduced time complexity in $O(\log m)$. However, in contrast to Gillespie's FRM, the NRM is a lot more difficult to efficiently parallelize.

Application-specific, power efficient hardware accelerators for exact stochastic simulation try to exploit the fine grain parallelism provided by modern FPGAs. However the design of such aggressively pipelined Systems on Chip (SoC) is still a complex process where the designer should strike a good balance between the complexity of the class of bio-models supported by the SoC and the size of the FPGA device used, in terms of on-chip resources (LUTs, RAMs and DSPs). Examples of FPGA solutions are those in [4], [5]. Likewise, GPU based solutions, such as [6] and [7], exploit the massively parallel compute power of GPUs that are nowadays readily available on the average scientist's PC. Their major drawback lies however in the difficulty of generating efficient 'kernels' (software) implementation for every biomodel at hand, and in the amount of available on-chip fast access RAM (hardware), which is very limited for the needs of stochastic simulations of networks with a large number of reactions and species.

In addition, there are pure software solutions for today's commodity multi-core CPUs. They are easy to install, setup and use, however they lack the performance and power efficiency of

978-1-4673-7813-0/15/$31.00 ©2015 IEEE

the above hardware accelerators due to their serial software nature. Advancements in modern compiler technologies, in conjunction with the already perfected instruction level parallelism (ILP) of multi-core processors, help mitigate the lack of specialized massively parallel hardware. Pure software simulators exploit these in order to provide decent performance on stochastic simulations of medium size bio-models (with up to few hundreds of reactions) using SSA algorithms. Software tools that support stochastic simulation are COPASI [8], StochKit and Matlab's SimBiology toolbox.

Given the pressing need of the systems biology community for efficient exact stochastic simulation of large-scale biochemical reaction networks (with thousands of reactions and species), in this paper we focus on methods to deliver scalable performance using the emerging class of many-core CPUs. By using a fully parameterized software framework we have developed for parallel stochastic simulation on many-core and multi-core CPUs, we evaluate the performance and scalability delivered when using the FRM SSA algorithm on different underlying hardware targets and core configurations.

Our first hardware target is Intel's Single-chip Cloud Computer [9], an experimental many-core CPU with 48 Pentium cores that feature a simple in-order execution engine and arranged across a mesh-type Network on Chip (NoC) communication fabric. This processor was presented by Intel Labs as "a concept vehicle" for many-core hardware and software research and we chose it as our first target due to its massively parallel architecture that, in contrast to FPGAs and GPUs, can be exploited using well-established parallel programming models and techniques. Our second hardware target, was a powerful and more recent commodity multi-core processor featuring Intel's 4th generation Core microarchitecture, code named "Haswell", that features highly optimized out-of-order execution and HT (HyperThreading) [10], Intel's flavor of SMT (simultaneous multi-threading).

Our intention is not to compare the absolute performance of the two CPU architectures, but rather their performance and scalability characteristics on the same problem, as its size increases. Our results show that significant speedup gains can be achieved by efficiently utilizing the parallelism afforded by both many-core and multi-core CPU architectures. To the best of our knowledge this is the first attempt to implement efficiently SSA algorithms for many-core CPUs.

The rest of the paper is organized as follows: In Section II we review Gillespie's FRM SSA algorithm and show two methods on the workload division among the units of execution that can be either separate processes or threads. In Section III we provide an overview of the Intel SCC NoC processor. In Section III we present our software framework for parallel stochastic simulation. In Section IV we present and discuss the scalability evaluation of the Intel SCC NoC and Core i7 processors using our software framework. Finally, in Section V, we summarize our findings and point to future research directions.

II. The FRM SSA Algorithm and Methods of Parallelization

A stochastic biochemical reactions network model is composed of n species $\{S_1, \ldots, S_n\}$ with initial concentrations $\{X_1, \ldots X_n\}$ that interact through m reaction channels $\{R_1, \ldots, R_m\}$. To simplify the analysis we consider that all species are uniformly distributed within some volume Ω inside a cell with unit size. This assumption allows us to simplify the calculations by ignoring the spatial effects that exist in the real world. Let $X_i(t)$ be the concentration of species S_i at time t. The state of the system at time t is $\boldsymbol{X}(t) = (X_1(t), X_2(t), \ldots, X_n(t))$ with initial conditions $\boldsymbol{X_0}(t) = \boldsymbol{x}_0$ at initial time $t = t_0$.

Stochastic simulation tracks the above state vector of concentrations of the network at appropriately chosen discrete time intervals, without explicitly solving the differential equations governing the underlying system dynamics. If a reaction R_μ occurs the current state $\boldsymbol{x}$ is updated by a factor, so that $\boldsymbol{X}(t + \tau) = \boldsymbol{x} + \boldsymbol{v}_\mu$. The $\boldsymbol{v}_\mu$ state update vector is equal to $(v_{1\mu}, \ldots, v_{n\mu})$, where $v_{i\mu}$ represents the change in the molecular count of S_i due to the occurrence of the reaction R_μ. Each such reaction R_μ is also associated with a specific probability rate constant c_μ, which is proportional to the reaction rate constant k_μ and inversely proportional to the volume Ω as shown in equations 7(a) and 7(b) of [2].

The probability that a randomly chosen combination of reactant molecules can interact and form a reaction channel R_μ within the next infinitesimal time interval $[t, t + dt]$ is given by $c_\mu dt$. The propensity $a_\mu(\boldsymbol{x})$ of reaction channel R_μ at state $\boldsymbol{X}$ is calculated by multiplying the probability rate constant c_μ by the number of possible combinations of reactant molecules for R_μ in state $\boldsymbol{x}$, as shown in equations (21) to (26) of [2]. Thus for second order reactions with two reactant species S_1, S_2, it holds that:

$$\alpha_\mu = c_\mu * X_1 * X_2 \quad (1)$$

The model described above is a Markov process, where the next state is only dependent on the previous one. Simulating this model yields the trajectories of $\boldsymbol{X}(t)$. Gillespie's original stochastic simulation algorithm (SSA), named as Direct Method (DM), is based on the above formulation and speeds up the process by introducing a new function $p(\tau, \mu \mid x, t)$ which is the probability that the next reaction in the system is R_μ and it occurs within the next infinitesimal time interval $[t, t + \tau]$, given that the current state of the system is $\boldsymbol{X}(t) = \boldsymbol{x}$. This has the advantage that the simulation can advance from one time step to the next, without the need to simulate in-between times, at which no reaction occurs. In this case the update of reactants and products happens in discrete amounts and species counts can be even very low, another notable advantage relative to the deterministic simulation algorithms.

While the DM SSA algorithm works fine for small biochemical networks, it is hard to parallelize and time consuming for medium to large size bio-models. Concerned with these issues Gillespie introduced an alternative but equivalent SSA algorithm, called as First Reaction Method (FRM). In this algorithm, a putative next reaction time τ_j is calculated for every reaction channel R_j. The reaction channel R_μ with the smallest next reaction time τ_μ is determined and this reaction is then "fired" at the end of the reaction cycle (RC). Since the calculation of each putative reaction time τ_j can proceed independently of all the others, the algorithm is a good candidate for massive parallelization. On the following page we list all the steps of the FRM SSA algorithm.

1. Initialize the state of the system $\boldsymbol{X(t_0) = x_0}$.
2. For every reaction R_j:
   - Compute the propensity function $\alpha_j(\boldsymbol{X})$.
   - Determine the time τ_j to the next reaction R_j.

   $$\tau_j = \frac{1}{\alpha_j(x)} * ln\left(\frac{1}{r_j}\right) = \frac{-1}{\alpha_j(x)} * ln(r_j) \quad (2)$$

   where each r_j is a unit uniform random number.

   End for every reaction R_j.
3. Find the reaction R_μ that has the smallest putative time $\tau_\mu = \min\{\tau_j\}$.
4. Determine the new time and system state after firing reaction R_μ:

   $$t' = t + \tau_\mu \quad (3)$$

   $$\boldsymbol{X}(t + \tau_\mu) = \boldsymbol{X}(t) + \boldsymbol{v}_\mu \quad (4)$$

   Where $\boldsymbol{v}_\mu$ is the state change vector for R_μ.
5. If the new time t' exceeds the desired simulation time T_{sim} then halt the simulation.
6. Go to step 2.

The FRM SSA can be parallelized by dividing its computation workload among N processing units. Steps 2 through 5 of the FRM SSA algorithm comprise a Reaction Cycle (RC) and each RC can be processed in parallel by partitioning its m reactions on those N processing units. This workload partitioning scheme will be referred to from now on as *Single Simulation in Parallel* (SSIP) and the associated workload placed on each processing unit equals to $W_{SSIP} = m/N$ reactions. Since we usually need to perform Monte Carlo experiments with a lot of repetitions (say R) of a biomodel's simulation, we can instead distribute the R repetitions among the N processing units. This coarser grain workload partitioning scheme will be referred to as *Multiple Simulations in Parallel* (MSIP) and the corresponding workload for each processing unit will be $W_{MSIP} = R/N$ repetitions in this case.

As mentioned in Section I, parallel implementations of Gillespie's SSAs exist for FPGAs, GPUs and multi-core CPUs. However, to the best of our knowledge none exists for a many-core processor like the experimental Intel SCC NoC, despite their flexibility and increased parallel processing capabilities.

III. The Intel SCC Many-Core Processor NoC

The Intel "Single-chip Cloud Computer" (SCC) [9] is an experimental many-core Network on Chip (NoC) processor architecture consisting of a mesh of 4 x 6 = 24 "tiles", as depicted in Figure 1. Each tile includes two Pentium P54C cores, along with their 32KB L1 (devided equally for instruction and data) and 256KB L2 caches. It also contains a Mesh Interface Unit (MIU) with circuitry to connect the cores onto the network and allow them to run at different frequencies. In addition a 16KB Message Passing Buffer (MPB) memory provides fast communication between the cores of the network, for a total MPB size of 384KB.

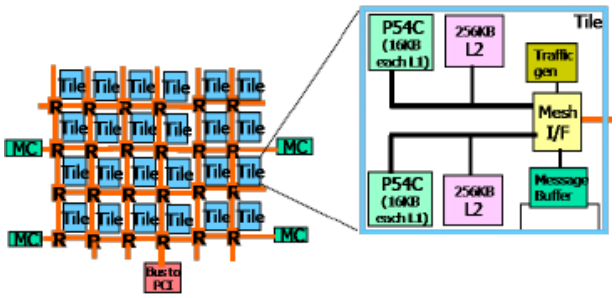

Figure 1. The Intel SCC many-cores NoC processor architecture with 24 "tiles" (2 cores per tile, along with 16KB MPB memory and a mesh interface unit), routers (R) and on-die DDR3 memory controllers (MC) [9].

A. Overview of Cores and Tiles

Each SCC core is a second generation Pentium P54C with a 32-bit x86 instruction set architecture (IA-32) that can run at frequencies from 533MHz up to 800MHz. These are small in-order cores that Intel synthesized for layout onto the chip. The absolute performance of these cores is not important since the processor was designed as a concept vehicle for researchers, to find ways to connect a large number of cores and assess how this architecture interacts with and enables application software. All of the cores act as independent computational nodes that communicate with others using either non-cache coherent shared memory through the 4 on-die DDR3 memory controllers, or through the distributed MPB memory with aid of RCCE, an MPI like library developed specifically for this purpose [11].

Every tile contains two cores and connects to a router that communicates with the Mesh Interface Unit (MIU) used to integrate the tile into the 2D mesh. The MIU packetizes and de-packetizes data to and from the mesh using a round-robin scheme for arbitration between the two cores of the tile. It also catches cache misses and decodes each core's 32-bit memory addresses into 34-bit system memory addresses, suitable for the memory controller it belongs. The memory address translation is managed through a lookup table (LUT) on each core. Each of the four memory controllers can address up to 16GB (2^{34}) bits of DDR3 memory for a maximum system memory of 64GB.

B. Memory Organization

The programmer has access to private off-chip memory which is accessed through the 4 on-chip DDR3 memory controllers. The LUTs on each core are configured in such a way so that specific regions of the DDR3 memory are only accessible through a single core. Since this corresponds to the traditional memory of a single core CPU of the past, the standard usage patterns apply and data is transferred from the core's registers, through L1 and L2 caches and finally into the DDR3 memory and vice versa. The same off-chip memory and its associated controllers provide access to shared memory. However these memory regions are mapped by the LUTs on all cores as un-cacheable regions by default in order to avoid consistency issues. This means that L1 and L2 caches are completely bypassed and data is fed directly to each core's register file. Since no cache coherence protocol is in place, it is the responsibility of the programmer to enforce partial order of operations across the cores. This sometimes requires mutually exclusive access to the shared memory regions using the test-and-set register provided for each core.

Since the processor does not offer any hardware managed cache coherency protocol, it features a new memory type to enable efficient communication between the cores. This new memory type is called the Message Passing Buffer Type (MPBT) and it consists of all the MPB buffers on the processor. Each one of the 24 tiles (with 2 cores) contains 16KB of MPB (8KB for each core) and therefore the size of this memory type is 384KB overall. A reserved bit in the page table of the cores is used to mark MPBT data. Legacy software by default does not mark memory as MPBT and runs without modifications. If the bit is set, data is cached in L1 and bypasses L2. To support software managed cache-coherence a new instruction was added to the P54C's x86 instruction set architecture which invalidates, but does not flush, all cache lines tagged as MPBT in the L1, so any subsequent accesses will go to DDR3 memory.

C. Software Platform

The most common software platform of the SCC is very similar to the one found on typical multi-core systems and it is based on a customized Linux operating system that runs separately on each core. The system includes drivers for the low-level access to the MPB memory and other hardware features of the SCC like the connection to the management console and the network communications (non MPB) between the cores of the die via an NFS file system visible among all the cores.

The lack of cache coherence between the cores suggests that the most natural and efficient programming model is the distributed one, where cores run Single Program Multiple Data (SPMD) programs and communicate by exchanging messages. For this purpose, a special library was developed and named RCCE (not an acronym) [11]. The RCCE uses the familiar to distributed message passing programmers SPMD model. The application is simultaneously launched on all cores using a helper script, called *rccerun*. This is actually a parallel ssh wrapper that connects to all the cores and executes the application that is stored on the NFS file system, which is in turn accessible from all cores. In the RCCE execution model, the application is started on each core in an unspecified order. Thus conformant programs must not depend on a particular order of execution on the cores.

A different effort in our research group has focused on the development of a skeletons library, called *rckskel,* that allows programming at a higher level masking the complexity of the hardware and allowing programmers to perform task based parallelism on the Intel SCC processor. It uses efficient task allocation, mapping, sequencing and execution patterns, in order to distribute the workload among the cores of the processor [12].

IV. Software Framework for Parallel Stochastic Simulations on Multi and Many Core Processors

We present here the software framework we have developed for performing stochastic simulations on multi-core and many-core processor architectures. The framework provisions the whole simulation flow, from configuration and loading of resources to the actual simulation scheduling and parallel execution, on the set of cores or threads the user configures during the setup phase. The supported SSA algorithm is, at the moment, the FRM SSA, both in SSIP and MSIP modes of parallel operation and is able to run on local workstations (with

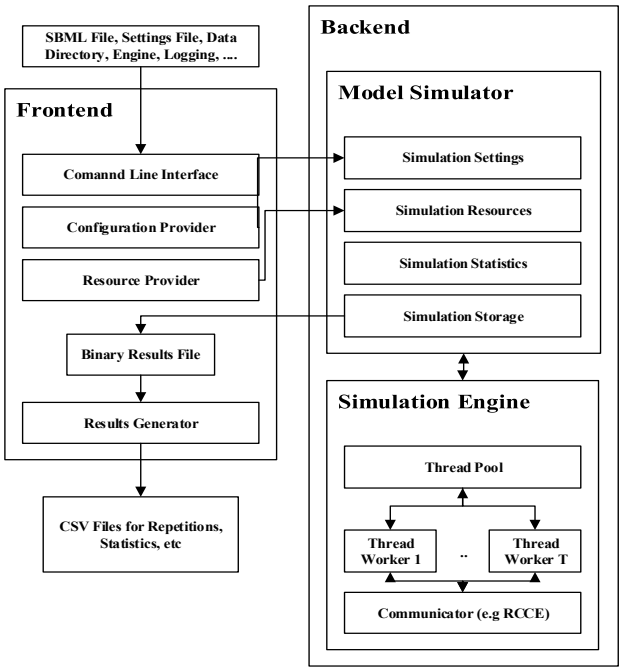

Figure 2. The software framework's main components

e.g Intel Core i5/i7 multi-core CPUs) as well as remotely onto the Intel "Single-chip Cloud Computer" (SCC) [9].

The software is written in C++ for performance reasons as well as code clarity and maintenance and employs the RCCE library for message passing communications. The compilers used were the open-source GNU GCC 4.9 and LLVM Clang 3.5. A GCC cross compiler that targets and optimizes for the x86 i586 micro-architecture of Intel SCC's Pentium cores was generated with the aid of the CrossTool-NG toolchain builder. The build system used for compiling the software was CMake. We also employed some popular open-source development libraries. *Boost* libraries were used for a variety of tasks and we also used *Google Log* in order to provide extensive logging capabilities to the application, *Google Flags* to easily declare and use command line arguments and last but not least, *Google Test* as the framework to write the unit tests for the application's classes.

A. The Frontend Components

The frontend components are those that closely interact with the end user and for this reason we chose to make them agnostic of the backend which includes components that actually perform the stochastic simulation on the underlying hardware using the selected SSA algorithm. This decoupling allows for flexibility on extending the framework with different SSA algorithms besides the FRM SSA. All of the components are shown in Figure 2.

The user controls the simulator using a command line interface (CLI) that allows him to select the simulation(s) configuration file, the binary results file location that will contain the resulting data set(s) of the simulation(s) and the data folder from which to read the simulation(s) resources. The user can also select the simulation engine to use which acts as the

TABLE I. SIMULATION SETTINGS

Parameter	Example for Intel SCC
The bio-model's name (represents the SBML filename)	ASYN
The number of reactions contained in the bio-model (*m*)	136
The number of species contained in the bio-model (*n*)	90
The SSA algorithm to use (e.g FRM)	FRM
The mode of parallelism to use (SSIP / MSIP)	SSIP
The range of cores to assign work	[00-47]
The number of threads to use on each core (if SMT/HT)	1
The number of repetitions to perform (*R*)	100
Total simulation time (in seconds) for each repetition	604800
Time interval between sampling periods (SP)	3600
Sampling period steps before aborting a repetition	1000000

hardware abstraction layer (HAL) to the underlying hardware (e.g SCC), whether to produce logs in console or in files with variable verbosity, and last but not least whether to run some unit tests prior to simulation.

The simulator's primary input is the biomodel file in standard SBML format [13] which contains all of the bio-model's reactions (*m*) and molecular species (*n*). Prior to executing the simulation, the *Resource Provider* is initiated in order to parse the original SBML file and generate the simulation's resources: the reaction table (RT) and the species table (ST). The RT table contains the *m* reactions with each one having separate reactant and product pointers (addresses) to molecular species counts stored on the ST table along with the probability rate constant c_μ, the $\boldsymbol{v}_\mu$ state vector with the stoichiometry of the reaction and the order of the reaction. We support reactions up to the 3rd order with up to three reactant and five product species. The ST table contains the initial concentrations $\{X_1, \ldots X_n\}$ of the *n* species participating in the simulation.

The configuration file is a text file that can be easily edited by the user and contains the desired simulations and their parameters. These are shown in Table I along with an example that can simulate the ASYN bio-model [15] using the FRM SSA algorithm on all 48 cores of the SCC processor for 100 repetitions with a simulation duration ("lab time") of 1 week. The user can choose the sampling (reporting) period (SP) of the simulation which has an immediate effect on the file size of the binary file that contains the complete output data set of the simulation. There is also a parameter that sets the maximum steps (reaction events) in a sampling period before aborting a repetition. This helps maintain the exact simulation in a running state even when the time steps produced (τ_μ) during an SP, are very small and cannot successfully lead to its completion.

The simulator saves the results into binary files. These files have a custom structure and contain all the repetitions of the performed simulation in a specific format readable by the *Results Generator*. Each repetition section includes the species populations on specific time intervals based on the sampling period that the user has defined on the configuration file. After each such section, the file contains repetition summary statistics like the reaction cycles (RCs) performed.

At the end of the simulation, the *Results Generator* has the ability to parse the binary files described above using multiple threads (on multi-core processors) and generate human readable, comma separated value (CSV) files with (possibly large) data-sets from the repetitions performed: the species counts at each sampling (reporting) period, as well as the average, minimum and maximum of molecular species populations, along with execution statistics such as the executed reaction cycles (simulation steps) and simulation (lab) times for each repetition. These CSV files can be in turn opened by other applications in order to further manipulate them and / or produce useful plots for the end-user.

B. The Backend Components

The actual simulation of a bio-model is performed by components that belong to this group. Each process after initialization eventually creates one or more *Model Simulator* objects that control and synchronize all of their underlying components. The *Simulation Settings* component is responsible for providing the user's configuration of the simulations that are currently in progress. The *Simulation Resources* component provides the SBML biomodel's resources (e.g. the RT and ST tables) to the worker units that actually do the computation. The *Simulation Storage* contains all the data structures for keeping the results of the simulation and traversing those on regular time intervals (based on the selected SP) allowing it to generate the resulting binary files of the simulation. The *Simulation Statistics* keeps the various statistics for all the species for all repetitions of the simulation.

The *Simulation Engine* is responsible for scheduling and dividing the work, statically at the moment, among all the worker threads that it spawns within the *Thread Pool* unit. The latter is also used for synchronizing the threads, whenever the SSA algorithm depends on such a barrier, using a *Thread Barrier* unit. Each worker thread spawned uses a *Thread Worker* derived object, like the *FRM Thread Worker*, as its underlying container holding the data structures and algorithmic functions of the selected SSA algorithm. This derived object corresponds to a *Unit of Execution* (UE), an abstraction which maps to a single thread for multi-core CPUs or a single core for multi-core ones. It is responsible for performing the computations in every repetition where each one continually creates reaction cycles and a specific reaction, having the minimum τ_j time on this UE, is selected among all the possible ones.

The communication between the UEs happen during two specific phases of the simulation. The first occurs only once, at the start of the simulation, where the master UE reads the resource files and distributes the data among the communication group (communicator) for this specific simulation. This can be done either with a broadcast or with a scatter primitive, depending on whether we want all the UEs to have a single or split representation of data respectively. The other communication phase happens when a set of *N* UEs execute the FRM algorithm in parallel SSIP mode, where at the end of each reaction cycle (RC), UE *C* will locally hold a winning reaction *R* having the minimal putative time τ_j among the *m / N* reactions assigned to it. A communication step is then needed, involving all *N* UEs, to *all-reduce* (*reduction* plus *broadcast*) the local minimum times and distribute the information of the global

"winner" reaction R_μ and time, $\tau_\mu = \min\{\tau_j\}$ (see algorithm in Section II) to all of the N UEs. Our framework realizes this *all-reduce* step on any subset of N UEs using RCCE's blocking primitives (*send* and *recv*). The method of *all-reduction* used matters only on many-core NoC processors and not on multi-core ones, since it is in the former case that different methods may have different routing latencies when passing messages between cores, while in the latter we have shared memory operations that impose almost the same latency on all cores.

Our *all-reduction* method works for any 2D mesh communicator, where we first *reduce* the local winning reactions of each row (in parallel), using a *hub* (star) communication scheme embedded into the row. When the leftmost UE of each row on the 2D mesh (let's call it $C_{i,0}$ without loss of generality), acting as the central place (*hub*) of its row, has acquired the local winning reaction of every other UE in its row, it *reduces* them to the one with the minimum time and initiates a *broadcast* of this row's winner reaction to all of the row's UEs. Subsequently, a second phase starts where the same procedure repeats along the columns of the 2D mesh, using the bottom UE of each column ($C_{0,j}$) as the master, i.e. the *reduction* target and subsequent *broadcast* initiator. At the end of this second phase every UE in the 2D mesh communicator has aquired the global winner reaction information and since it has a local copy of the model's current species table (state trajectories $\boldsymbol{X(t)}$), it can update independently the affected species counts and then proceed with the execution of the next RC. In all cases, the communication primitives used by our software framework are parameterized by the size of the communicator *(N)* and can be applied to any subset of UEs on multi-core or many-core CPUs, no matter what is its underlying physical configuration.

V. Performance Results

In this section we describe the simulation experiments performed in order to assess the throughput and performance delivered to the application when our framework is running the same biomodel, but on different CPU architectures and core configurations. We measured the running times and total reaction cycles performed during all repetitions of each simulation and report the simulation's *Throughput* (in Mega Reaction Cycles per second - MRC/sec), *Performance* (in Mega Reactions per second - MR/sec), the *Speedup* (*S*) factor achieved relatively to using one core, and the *Efficiency* (*E*) i.e the speedup over the total number of cores, for a variety of core configurations. Note that during the execution of one RC the simulator should evaluate all *m* reactions of the model.

A. Experimental Systems and Bio-Models Used

All simulation experiments were performed using the same algorithm (FRM SSA) in both SSIP and MSIP modes of operation. A first baseline for benchmarking was established by running each simulation on a single core of the SCC NoC processor, an Intel Pentium (P54C) at 533MHz running SCC Linux. A second baseline was established by running the same bio-model, using our framework setup for utilizing one core of a very powerful workstation PC with a quad-core Intel Core i7 4790K CPU running at 4GHz (when workload is placed on all cores) to 4.4GHz (when workload is placed on a single core) with 32 GB of RAM and a fast SSD under GNU/Linux OS.

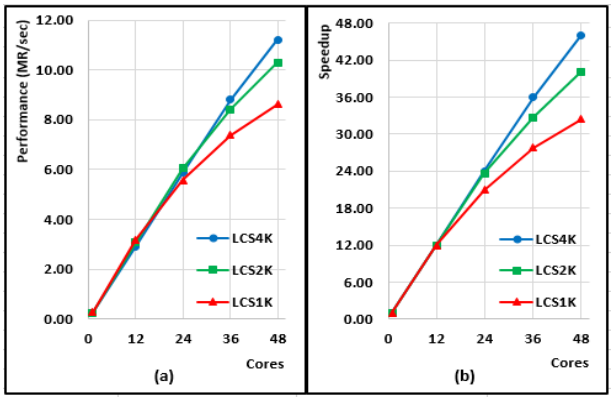

Figure 3. Intel SCC NoC scalability analysis. (a) Performance (MR/sec) and (b) Speedup while running the LCS 1K/2K/4K bio-model configurations using the parallel FRM SSA in SSIP mode.

For all simulations we used two bio-models. The first, is the well known Linear Chain System (LCS) of 2nd order reactions [14] that is commonly used as a benchmark, in the sense that we can easily scale its workload on demand. The format of its 2nd order reactions (R_i, $i = 0,1,2, \ldots, m\text{-}1$), all having two reactant and two product species, is the following:

$$S_{i \bmod m} + S_{(i+1) \bmod m} \xrightarrow{k_i} S_{(i+2) \bmod m} + S_{(i+3) \bmod m}$$

The second biomodel used was developed by our group and has been recently judged to become the "model of the month" (3/2015) of the EBI Biomodel Database [15]. It was developed to study how the polymerization of protein Alpha-synuclein (ASYN) affects different parts of the cell and disturbs the homeostasis of dopaminergic neurons, a mechanism that is believed to play a key role at the onset of Parkinson's disease. Our ASYN biomodel has $m = 136$ reactions (mass action kinetics), $n = 90$ species and is a typical (medium size) bio-model of the EBI DB, with a mix of reactions of different orders and stoichiometries.

B. SSIP Simulation Experiments

The first experiment tries to evaluate the communication cost and its impact when using the SSIP mode parallel implementation to partition the reactions of the FRM SSA among the many cores of the SCC processor. This kind of simulation can tell us how the efficiency drops with the number of cores, for a fixed problem size, and how many cores of the CPU can be used efficiently i.e. before the speedup levels off, to accelerate a single run (repetition) of the network's simulation. We used the LCS bio-model with three different configurations in terms of the reactions workload *(m):* 1024 (1K), 2048 (2K) and 4096 (4K) in the evaluation. Every setup was run a minimum of five times and we additionally performed more executions until we couldn't observe any noticeable change on the simulation times.

We observed performance and speedup gains while increasing the cores count up to a certain point, as shown in Figure 3. Detailed results for the LCS4K (model simulation time T_{sim}= 10ms) are provided in Table II, where speedup, throughput and performance figures are calculated based on the actual time taken to complete the simulation on a subset of the SCC cores (using an increment of 12 cores). The results demonstrate that as

TABLE II. INTEL SCC NOC 48-CORE CPU. STOCHASTIC SIMULATION PERFORMANCE SCALING (PARALLEL FRM-SSA, LCS4K MODEL)

Intel SCC Cores	Time (sec)	Speedup	Efficiency	Throughput (RC/s)	Performance (MR/s)
1	6883	1.00	1.00	60	0.24
12	574	11.99	0.99	714	2.92
24	287	23.99	0.99	1432	5.86
36	191	35.98	0.99	2149	8.80
48	150	46.03	0.96	2740	11.22

TABLE III. INTEL CORE I7 4-CORE CPU, STOCHASTIC SIMULATION PERFORMANCE SCALING (PARALLEL FRM-SSA, LCS4K BIOMODEL)

Intel Core i7 Threads	Time (s)	Speedup	Efficiency	Throughput (RC/s)	Performance (MR/s)
1	302	1.00	1.00	1359	5.57
2	154	1.97	0.98	2673	10.95
4	80	3.80	0.95	5161	21.14
6	91	3.34	0.56	4539	18.59
8	72	4.21	0.53	5716	23.41

the LCS bio-model's complexity increases in terms of reactions, we observe improved speedup for the same number of cores. For example for the LCS4K model speedup remains linear and efficiency is almost perfect (1.0), all the way up to 36 cores. Even when all 48 cores are used the efficiency is very good and almost linear (0.96). Speedup on smaller models levels off earlier, as expected. This is attributed to the fact that for a fixed problem size *(m)* as the number of cores increases the communication costs increase, while the computation workload per core actually decreases due to the reactions partitioning (SSIP mode). So for every stochastic simulation of fixed problem size there is a number of cores beyond which communication dominates computation leading to leveling performance. Nevertheless the SCC NoC can deliver high performance (more than 8MR/sec) at an efficiency larger than 66%, even in models of m = 1024 reactions (the smallest size considered).

The second experiment was similar with the first, but now using a different target machine, one that contains the Core i7 multi-core processor associated with the second baseline. This experiment allowed us to assess not only the expected performance of a modern multi-core CPU readily available for scientists and engineers (please remember that the SCC is an experimental research concept CPU), but also how it behaves as the number of cores increase along with the problem size. We should remark that in this case our software framework utilizes shared memory among the running threads since it does not have access to a hardware memory type, such as the MPB of the SCC processor.

The Intel's Core i7 multi-core processor implements HT (HyperThreading) [10], Intel's flavor of SMT (simultaneous multi-threading), in order to run up to two concurrent threads on each core's hardware, for a total concurrent execution of 8 threads and thus we can perform immediate comparisons between the many-core and multi-core processors. As shown in Table III we observe the same increasing speedup on this multi-core processor when running the LCS4K biomodel (T_{sim} = 10msec) on up to the 4 threads configurations. This is to be expected since the parallel FRM SSA algorithm's implementation using our software framework remains the same and there is still the same communication overhead that does not dominate the computation times as long as the bio-model has a high enough complexity (number of reactions).

The steep speedup drop (efficiency loss) when using more than 4 threads is attributed to the fact that this is a quad-core processor that implements HT [10]. Since all threads operate on exactly the same type of instruction workload (SPMD) it cannot take advantage of each core's superscalar architecture that needs varied workload placement (e.g integer and floating point arithmetic at the same time) to show performance advantages when utilizing more threads than cores. Of course throughput and performance figures are way ahead of the many-core SCC processor due to the fact that this multi-core CPU contains latest generation cores, featuring a highly superior, out-of-order micro-architecture and clocked at almost 8X the frequency of the SCC. We believe that even more performance can be exploited from next generation multi-core processors, should they start introducing hardware memory structures like the MPB of the SCC NoC processor alongside their cores, for increased communication efficiency among them, without resorting to shared memory and its unavoidable locks or cache-coherency protocol overheads.

C. MSIP Simulation Experiments

In the third experiment we used the SCC to perform many repetitions of the ASYN biomodel in MSIP mode of operation, exploiting all 48 cores of the SCC, each one with a workload of *R/N* repetitions. This kind of simulation is useful to assess the maximum expected performance a CPU can deliver, since after simulation initialization, where broadcasting of biomodel resource data occurs, the simulation repetitions proceed independently in parallel and there is no communication among the cores. All 48 SCC cores finished the ASYN model's simulation (T_{sim} = 604800 secs = 1 Week, 48 repetitions) that consisted of a total of 0.003 million RCs in T_{real} = 54 secs. The achieved throughput was 0.41 MRC/sec and the performance, when considering the model's complexity (m = 136), reached 19.06 MR/sec. This is an estimate of the maximum performance that the Intel SCC many-core processor running at 533MHz can deliver during a stochastic simulation with the FRM SSA algorithm. In comparison, our framework running 8 threads on Intel Core i7 performed the same 48 repetitions in 29 secs, giving a speadup of only S = 1.86 relatively to the SCC NoC CPU which features simple in-order, low frequency Pentium cores. All results are summarized in Table IV.

TABLE IV. PERFORMANCE COMPARISON OF INTEL SCC NOC VS. INTEL CORE I7 CPUS (FRM SSA MSIP MODE - 48 REPETITIONS)

Setup	Time (s)	Speedup	Throughput (MRC/s)	Performance (MR/s)
1 Core of SCC NoC	2495	1	0.003	0.41
48 Cores of SCC NoC	54	46.25	0.140	19.06
1 Thread on Core i7	147	1	0.051	7.00
8 Threads on Core i7	29	5.06	0.258	35.11

We also experimented with the ASYN bio-model in SSIP mode of operation but measured low performance when dividing the reaction workload on more than 10 cores. Scalability and efficiency were suboptimal due to the medium size workload of this model, that is not high enough to keep more than 10 cores busy between each communication phase. For this reason and for such medium size models we recommend the utilization of small subsets of cores (clusters), to compute reaction cycles in parallel (SSIP mode) and then employ many such clusters, to compute many simulation repetitions in parallel (MSIP mode). This hybrid *C x N* scheme, where *N* is the number of cores combined in SSIP mode and *C* is the number of SSIP simulation runs combined in MSIP mode, might prove to be ideal for exploiting in the best way many-core CPUs. We plan to further research such setups, in order to find the optimal balance and thus throughput and performance, as the number of repetitions and the model complexity increase at the same time, which is the most practical scenario for large scale biomolecular network stochastic simulations in systems biology.

VI. Conclusions

In silico investigation of the stochastic dynamics of large scale biological system models is an important problem in systems biology, with applications in drug design and personalized medicine. Stochastic simulation is the method of choice since it allows accounting for the "noise" inherent in biological systems and assessing their behavior even in the very low species counts regime. However as models become more complex (bio-molecular interactions increases to thousands) it is also required to run a large number of repetitions of the bio-model simulations to assess not only the average behavior but also the sensitivity and robustness of a system under different conditions.

We have developed a parallel implementation of the FRM SSA algorithm that can divide the bio-model reactions, or the simulation repetitions, workload on a user selected number of cores. Experimental results demonstrate that both processor architectures considered have the potential to deliver high performance and linear speedup as long as the problem scales on both dimensions i.e. in terms of the biomodel's complexity as well as in terms of the number of required repetitions of the simulation. Our software framework can be extended to incorporate more stochastic simulation algorithms, e.g. Gibson and Bruck's NRM SSA, more hardware targets, and hybrid configuration setups. These features, if properly combined, make the framework a solid research 'vehicle' for the stochastic simulation landscape and allow for experimentation and performance assessment with a multitude of setups and parameters.

Acknowledgments

This research is implemented in the framework of operation "ARISTEIA II", which is co-funded by the European Union (European Social Fund) and national resources through the Operational Programme "Education and Lifelong Learning", (NSRF 2007-2013). We would also like to thank MicroLab - ECE - National Technical University of Athens, for allowing use of their SCC infrastructure, performed under the supervision of Mr. D. Rodopoulos and Prof. D. Soudris.

References

[1] M. Tomita, "Whole-cell simulation: a grand challenge of the 21st century," *TRENDS in Biotechnology,* vol. 19, no. 6, pp. 205-210, 2001.

[2] D. T. Gillespie, "Stochastic simulation of chemical kinetics," *Annu. Rev. Phys. Chem.,* vol. 58, pp. 35-55, 2007.

[3] M. A. Gibson and J. Bruck, "Efficient exact stochastic simulation of chemical systems with many species and many channels," *The Journal of Physical Chemistry A,* vol. 104, no. 9, pp. 1876-1889, 2000.

[4] E. Logaras, O. Hazapi and E. Manolakos, "Python to accelerate embedded SoC design: a case study for systems biology," *ACM Transactions on Embedded Computing Systems (TECS),* vol. 13, no. 4, pp. 84:1-84:25, 2014.

[5] L. Macchiarulo, "A massively parallel implementation of Gillespie algorithm on FPGAs," in Proc. of 30th Annual IEEE Int'l Conf. in Medicine and Biology Society, pp. 1343-6, 2008.

[6] G. Klingbeil, R. Erban, M. Giles and P. K. Maini, "StochsimGPU: parallel stochastic simulation for the Systems Biology Toolbox 2 for MATLAB," *Bioinformatics,* vol. 27, no. 8, pp. 1170-1171, 2011.

[7] Y. Zhou, J. Liepe, X. Sheng, M. Stumpf and C. Barnes, "GPU accelerated biochemical network simulation," *Bioinformatics,* vol. 27, no. 6, pp. 874-6, 2011.

[8] S. Hoops, S. Sahle, R. Gauges, C. Lee, J. Pahle, S. Natalia, M. Singhal, L. Xu, P. Mendes and U. Kummer, "COPASI - A complex pathway simulator," *Bioinformatics,* vol. 22, no. 24, pp. 3067-3074, 2006.

[9] J. Howard, et al., "A 48-Core IA-32 Processor in 45 nm CMOS Using On-Die Message-Passing and DVFS for Performance and Power Scaling," *IEEE Journal of Solid-State Circuits,* vol. 46, no. 1, pp. 173-183, 2011.

[10] S. Saini, H. Jin, R. Hood, D. Barker, P. Mehrotra and R. Biswas, "The impact of hyper-threading on processor resource utilization in production applications," in 18th Int'l Conference on High Performance Computing (HiPC), pp. 1-10, 2011.

[11] "RCCE," Intel, [Online]. Available: http://www.intel.com/content/www/us/en/research/intel-labs-rcce-single-chip-cloud-brief.html. [Accessed 10 2 2015].

[12] A. Sharma, A. Papanikolaou and E. S. Manolakos, "Accelerating All-to-All Protein Structures Comparison with TMalign," in Proc. IEEE Int'l Symp. on Parallel & Distributed Processing, Workshops (IPDPSW), pp. 510-9, 2013.

[13] M. Hucka, et al., "The Systems Biology Markup Language (SBML): A Medium for Representation and Exchange of Biochemical Network Models," *Bioinformatics,* vol. 9, no 4, pp. 524-531, 2003.

[14] Y. Cao, H. Li and L. Petzold, "Efficient formulation of the stochastic simulation algorithm for chemically reacting systems," *The Journal of Chemical Physics,* vol. 121, no. 9, pp. 4059-4067, 2004.

[15] E. Ouzounoglou, D. Kalamatianos, E. Emmanouilidou, M. Xilouri, L. Stefanis, K. Vekrellis and E. S. Manolakos, "In silico modeling of the effects of alpha-synuclein oligomerization on dopaminergic neuronal homeostasis," *BMC Systems Biology,* vol. 8, no. 1, p. 54, May 2014.
Bio-model (SBML) available online: https://www.ebi.ac.uk/biomodels-main/static-pages.do?page=ModelMonth/2015-03

From: **Waleed Smari** <smari@arys.org>
Date: Mon, Jun 22, 2015 at 6:15 AM
Subject: Outstanding Paper Nomination HPCS 2015-URGENT
To:

Hello,

Congratulations. Your paper has been nominated for the HPCS 2015 Outstanding Paper Award.

Can you please forward to me the acceptance notification email of your HPCS 2015 paper along with all the reviews and comment you received, at your earliest convenience (by this Tuesday noon New York time at the latest)?

We will need these for the records of the Awards Committee since your paper is under consideration for the Outstanding Paper Award of the HPCS 2015 Conference.

We appreciate your prompt help with this matter.

Thank you very much.

Best Wishes.

--ws./
On Behalf of the HPCS 2015 Organizers