StochKit-FF: Efficient Systems Biology on Multicore Architectures

Marco Aldinucci ⋆[1], Andrea Bracciali ⋆[2], Pietro Liò ⋆[3], Anil Sorathiya[3], and Massimo Torquati[4]

[1] Computer Science Department, University of Torino, Italy `aldinuc@di.unito.it`
[2] ISTI - CNR, Italy `braccia@di.unipi.it`
[3] Computer Laboratory, Cambridge University, UK `{pl219, as883}@cam.ac.uk`
[4] Computer Science Department, University of Pisa, Italy `torquati@di.unipi.it`

Abstract. The stochastic modelling of biological systems is informative and often very adequate, but it may easily be more expensive than other modelling approaches, such as differential equations. We present StochKit-FF, a parallel version of StochKit, a reference toolkit for stochastic simulations. StochKit-FF is based on the FastFlow programming toolkit for multicores and on the novel concept of selective memory. We experiment StochKit-FF on a model of HIV infection dynamics, with the aim of extracting information from efficiently run experiments, here in terms of average and variance and, on a longer term, of more structured data.

Key words: Stochastic biological models, simulation, multicore.

1 Introduction

The immune system is an example of a complex system formed out of its intercellular and intracellular components, which organise in space and time the immune response to pathogens through a system of positive and negative regulatory nested feedbacks. The modelling of part of the immune response to HIV infection is a paradigmatic scenario illustrating the challenges that computer-based modelling and analysis present for this class of problems. The immune system can be both modelled by deterministic differential equations (ODEs) and by stochastic modelling approaches. ODEs are effective in characterizing the system dynamics when the molecular copy number of each species is sufficiently large. A stochastic model is much more accurate when the number of molecules considered is small. The numerical solvability of stochastic models is limited to pretty small dimensions (e.g. number of species) due to their exponential complexity. The behaviour of larger systems can be described by stochastic simulations, e.g. those based on the *Gillespie's algorithm*, which simulates the system dynamics

⋆ MA and AB have been partially supported by the HPC-Europa 2 *Transnational Access* programme, MA by the Regione Piemonte *BioBITs* Project AB by the CNR project RSTL-XXL, and PL by the EC's IST SOCIALNETS project (217141) and the U.S. Army Research Lab. and U.K. Ministry of Defence (W911NF-06-3-0001).

step by step. These methods, although often more accurate than the deterministic ones, can be highly demanding in terms of computational power, e.g. when considering many simulations for increasing the precision of the model. Stochastic methods represent a challenging methodological areas of system biology and play a growing role in modelling immune responses to pathogens.

We here illustrate the use of parallelism for supporting efficient and informative stochastic analysis of one such model. Multiple simulations exhibit a natural independence that would allow them to be treated in an *embarrassingly parallel* fashion. However, this is not possible whenever the results need to be concurrently combined or compared. Often, recombination is done in a post-processing phase as a sequential process whose cost in time and space depends on the number and the size of the simulation results and can be comparable to the cost of the simulation phase. Besides, independent simulations exhibit good parallel scalability only if executed onto truly independent platforms (e.g., multicomputers, clusters or grids), but they might exhibit serious performance degradation if run on multicores due to the concurrent usage of underlying resources. This effect is particularly significative for I/O-bound applications since typically I/O and memory buses are shared among cores.

We introduce StochKit-FF, a parallel version of the popular StochKit [1], aiming at supporting the execution of multiple simulations and at combining their results on cache-coherent, shared memory multicores. These architectures are currently being developed by the whole computer industry and equip many computing platforms. StochKit-FF has been designed and developed as a low-effort, efficient porting of StochKit by means of the FastFlow C/C++ programming framework, which supports efficient applications on multicore and makes it possible to run multiple parallel stochastic simulations and combine their results. This relies on *selective memory*, a novel data structure we designed to perform the online *alignment* and *reduction* of multiple streams of simulation results: different data streams are aligned according to simulation time and combined together according to a user-defined function, e.g. the average or others. By discussing the HIV case-study, we intend to show that this framework represents an efficient way for running multiple simulations and for the development of effective modelling techniques. We focus here on producing averaged values, and on more structured and informative data on a longer term project.

2 A stochastic model of the immune response to HIV

ODEs based models have long been used for immune system and viral infection modeling [2, 3]. They focus on the average behavior of large populations of identical objects and need often to be solved numerically. When considering a small number of molecules, which is highly probable if we consider immune cell interactions in a small volume, or when considering randomness and irregularities found at all levels of life, then a stochastic model is much more accurate on a mesoscale. Stochastic methods are based on the *Gillespie's algorithm*, which simulates the reactions step by step [4]. Such stochastic methods are more effective

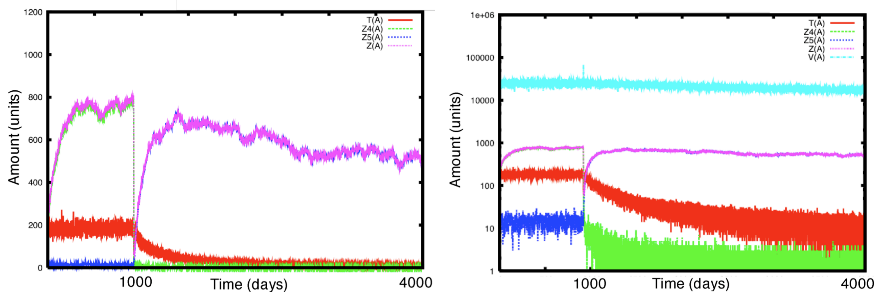

Fig. 1. a) The “noisy” immune cell dynamics over 4000 days: mutation around day 1000 and then T (mid-curve) degrades. b) A log-scale view: the high peak perturbation of $V4 + V5$ during mutation and the dynamics for the small amounts of degrading cells.

than the deterministic ones to describe the above mentioned irregularities and crucial chemical reactions. They observe emerging properties of the behaviour of a system composed of a large number of simple agents (viruses and cells), following local rules [5].

Briefly, agent behaviour consists of actions, e.g. cellular interactions, that cause a state transition of the modelled system, e.g. a variation in the amount of agents. Actions are stochastic, as their occurrence in time has an associated probability distribution, which is generally memoryless, typically negative exponential distributions with the rate as parameter. Hence the overall system behaviour can be interpreted as a Continuous Time Markov Chain (CTMC). Systemic emergent properties can be sensitive to the local presence of minimal (integer) quantities of agents/molecules/cells [6]. The combined behavior of these agents is observed in a discrete-time stochastic simulation, from given initial conditions: a single transition amongst the possible ones in the current state is selected, and the state updated accordingly. The Gillespie’s algorithm [4] determines the next transition and the time at which it occurs, exactly according to the given probability distributions. Each such possible evolution of the system is called a *trajectory*. Large computing resources may be required to correctly determine fluctuations and averages from the system simulated trajectories.

HIV and the immune response. We recapitulate here our model here, see [2, 7] for details. During the HIV infection multiple strains of the virus arise, we consider two phenotype classes, $V5$ and $V4$, which invade cells through different membrane receptors. The mutation from $V5$, initially prevailing, to the more aggressive $V4$ has been correlated to the progression to the AIDS phase. The immune response is based on the action of several cells (T, $Z5$ and $Z4$), some of which strain specific, which can also be infected by the viruses. The Tumor Necrosis Factor F induces bystander death of several cells. Infection is characterised by the progressive loss of (infected) T, $Z5$ and $Z4$ cells. Mature T cells and $Z4$ and $Z5$ cells are produced at a constant rate (i.e. the parameter of the associated probability distribution). All cells are typically also cleared out at a given rate, some of them, e.g. T, are also cleared out by the interaction with F (Tumor Necrosis Factor). $V5$ and $V4$ produce the infected cells $I5$ and $I4$, which then produce a large number of $V5$ and $V4$. The accumulation of F is proportional to the amount of $V4$. $Z4$ and $Z5$ proliferate due to infection and sustain the production of T (some of these represented dynamics are *abstractions* of more complex interactions). $V5$ strains mutates into $V4$ strains as the effect of a stochastic triggering event expected to occur around a desired time. The parameters used have been referred from literature, e.g. [2, 3] and sometimes tuned against the known behaviour of the system. Simulations start from given initial conditions, e.g. $T = 1000, Z5 = 250$ and $V5 = 100$. See Fig. 1 for a trajectory of the modelled infection dynamics.

3 Parallel Stochastic Simulations

In stochastic simulations, many trajectories might be needed to get a representative picture of how the system behaves on the whole. Processing and combining many trajectories may lead to very high compulsory cache miss-rate and thus become a memory-bound (and I/O-bound) problem. This in turn may require a huge amount of storage space (linear in the number of simulations and the observation size of the average trajectory) and an expensive post-processing phase, since data should be retrieved from permanent storage and processed. Eventually, the computational problem hardly benefits from the latest commodity multi-core architectures. These architectures are able to exhibit an almost perfect speedup with independent CPU-bound computations, but hardly replicate such a performance for memory-bound and I/O-bound computations, since the memory is still the real bottleneck of this kind of architectures. Tackling these issues at the low-level is often unfeasible because of the complexity of the code and of the need to keep the application code distinct from platform-specific performance tricks. Typically, low-level approaches only provide the programmers with primitives for flow-of-control management, synchronisation and data sharing.

Designing suitable high-level abstractions for parallel programming is a long standing problem [8]. Recently, high-level parallel programming methodologies are receiving a renewed interest, especially in the form of pattern-based programming [9, 10]. FastFlow belongs to this class of programming environments.

The FastFlow Parallel Programming Environment. FastFlow is a parallel programming framework aiming at *simplifying* the development of *efficient* applications for multicore platforms, being these applications either brand new or ports of existing legacy codes. The key vision underneath FastFlow is that effortless development and efficiency can both be achieved by raising the level of abstraction in application design, thus providing designers with a suitable set of parallel programming patterns that can be compiled onto efficient networks of parallel activities on the target platforms. The FastFlow run-time support is completely based on lock-free and memory fence-free synchronizations. This approach significantly reduces cache reconciliation overhead, which is the primary source of inefficiency in cache-coherent multicore platforms. We refer to [12, 11] for any further details. FastFlow is open source available at [11] under LGPL3.

3.1 Parallel StochKit: StochKit-FF

StochKit [1] is an extensible stochastic simulation framework developed in the C++ language. It implements the popular Gillespie algorithm, explicit and implicit tau-leaping, and trapezoidal tau-leaping methods.

StochKit-FF extends StochKit (version 1) with two main features: The support for the parallel run of multiple simulations on multicores, and the support for the online (parallel) *reduction* of simulation results, which can be performed according to one or more user-defined associative and commutative functions. StochKit v1 is coded as a sequential C++ application exhibiting several non-reentrant functions, including the random number generation. Consequently, StochKit-FF represents a significative test bed for the FastFlow ability to support parallelisation of existing complex codes. The parallelisation is supported by means of high-level parallel patterns, which could also be exploited as *parametric code factories* to parallelise existing, possibly complex C/C++ codes [11].

In particular, StochKit-FF exploits the FastFlow *farm* pattern, which implements the functional replication paradigm: a stream of independent data items are dispatched by an *Emitter* thread to a set of independent *Worker* threads. Each worker produces a stream of results that is gathered by a *Collector* thread into a single output stream [12].

In StochKit, a simulation is invoked by way of the `StochRxn()`, which realises the main simulation loop; the propensity function and initial conditions are among its parameters. StochKit-FF provides programmers with `StochRxn_ff()` function, which has a similar list of parameters, but invokes a parametric simulation modelling either a number of copies of the same simulation or a set of parameter-sweeped simulations. `StochRxn_ff()` embodies a *farm*: the emitter unrolls the parametric simulation into a stream of standard simulations (represented as C++ objects) that are dispatched to workers. Each worker receives a set of simulations, which are sequentially run by way of the `StochRxn()`, which is basically unchanged with respect to the original StochKit. Each simulation produces a stream of results, which are locally *reduced* within each worker into a single stream [13]. The collector gathers all worker streams and *reduces* them again into a single output stream. Overall, the parallel *reduction* happens in a systolic (tree) fashion via the so-called *selective memory* data structure.

Selective Memory. Together with StochKit-FF, we introduce the *selective memory* concept, i.e. a data structure supporting the on-line *reduction* of time-aligned trajectory data by way of user-defined associative functions. Selective memory distinguishes from standard *reduce* operation [13] because it works on unbound streams, and aligns simulation points (i.e. stream items) according to simulation time before reducing them: since each simulation proceed at a variable time step, simulation points coming from different simulations cannot simply be reduced as soon as they are produced. Selective memory behaves as a sliding

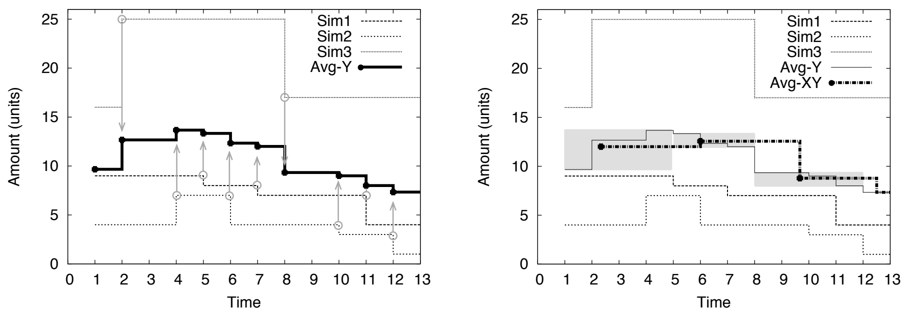

Fig. 2. Selective Memory with average. Left: a) Curve Avg-Y is derived via oversampling and time-aligned reduction (average along Y axis) of k independent simulations (arrows highlight oversampling). Right: b) Avg-XY is derived by the reduction (average along X axis) of k successive points of Avg-Y (grey boxes highlight averaging zone).

window in a buffer that follows the wavefront of generated simulation points. It keeps the bare minium amount of data from different simulations to produce a slice of simulation points that are aligned to simulation time.

The behaviour of selective memory is shown in Fig. 2 using average as combining function. Simulation points from different simulations are first averaged at aligned simulation time points: such computed average results oversampled with respect to single simulations (Fig. 2 a). This oversampling is possibly reduced by applying the same technique along time axis (Fig. 2 b). Overall, selective memory produces a combined simulation that has been adaptively sampled: time intervals exhibiting a higher variability across different simulations exhibit an higher sampling rate. Selective memory effectively mitigates the memory pressure of result logging when many simulations are run on a multicore, as it substantially reduces the output size, and thus capacity misses and the memory bus pressure.

4 Experiments and Discussion

Figure 3 a) is a focus on the immune response averaged over 16 simulations, performed on the `ness.epcc.ed.ac.uk` platform (Sun X4600 SMP - 8 x Dual-Core AMD Opteron 1218, 32 Gb memory) hosted at EPCC, University of Edinburgh. The averaged amounts and the variance of $Z4$, $Z5$, their sum Z, and T are reported. The variance of $Z4$ and $Z5$ is large till 2500 days, showing tight coupling i.e. interdependence. Then, the variance of T decreases continuously, while the one of $Z5$ decreases with the amount of $Z5$: it is not much involved in dynamics after the mutation to $V4$. Figures 3 b)-d) describe a sensitivity analysis for δ_t, which has resulted in being very influential by a large analysis of the model: it strongly impacts on the diffusion of $V_{5,4}$. In b) V is immediately (in the interval [0,100]) cleared out, T rapidly increases, and the system is very stable, with a low variance. In c) the immune response still prevails, but the system appears much perturbed. In d), well below the standard value of δ_t, the virus clearly prevails. Variance is initially high, then it stabilises towards a steady state.

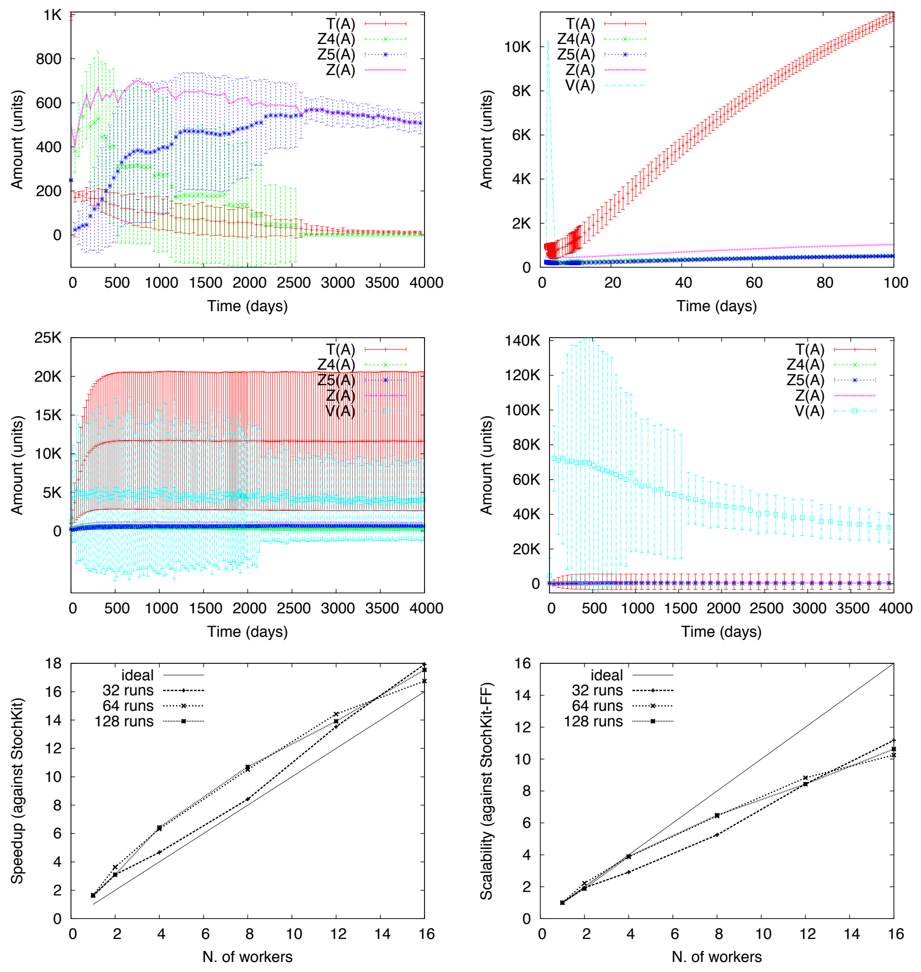

Fig. 3. Experimental results. Left to right: a) A focus on the immune response, and b) c) d) Sensitivity analysis for $\delta_t = 5.0, 3.0$ and 0.5 (average and variance for multiple trajectories (16x)). e) Speedup of StochKit-FF against StochKit. f) Scalability of StochKit-FF(n) against StochKit-FF(1), where with n is the number of worker threads.

The performances of StochKit-FF have been evaluated on multiple runs of the HIV case-study. A single run of the simulation with StochKit produces $\sim$ 150M simulation points for 4000 days of simulated time (sampled from $\sim$ 6 GBytes of raw data); multiple runs of the same simulation will need a linearly greater time and space. These simulations can be naively parallelised on a multicore platform by running several independent instances, which however, will compete for memory and disk accesses, thus lead to suboptimal performances. An additional linear time (at least) in the number and the size of outputs should be spent in the postprocessing phase for the recombination of results.

StochKit-FF mainly attacks these latter costs by online reducing the outputs of simulations, which are run in parallel. As shown in Fig. 3 e), where average and variance are used as combining functions, StochKit-FF exhibits a superlinear speedup with respect to StochKit in all tested cases. This superlinear speedup is mainly due to the fact that StochKit-FF is about two times faster than StochKit even when running with just one thread. They are mainly due to FastFlow memory allocator that is faster than standard memory allocator on the testing platform. As shown in Fig. 3 f), StochKit-FF exhibits a good scalability also when compared with the sequential (one-thread) version of StochKit-FF.

5 Concluding remarks

StochKit-FF we presented has been realised as a minimal-modification porting of a complex application supported by the FastFlow framework. StochKit-FF suitably recombines the results of efficiently run multiple stochastic simulations by exploiting the idea of selective memory. We have presented experiments, highlighting both the aspects of the emerging behaviour of a realistic model of the HIV infection and efficient performances.

References

1. Petzold, L.: StochKit web page. `http://engineering.ucsb.edu/~cse/StochKit`
2. Perelson, A., Neumann, A., Markowitz, M., Leonard, J., Ho, D.: HIV-1 dynamics in vivo: Virion clearance rate, infected cell life-span, and viral generation time. Science **271** (1996) 1582–1586
3. Sguanci, L., Bagnoli, F., Liò, P.: Modeling HIV quasispecies evolutionary dynamics. BMC Evolutionary Biology **7(2)** (2007) S5
4. Gillespie, D.: Exact stochastic simulation of Coupled Chemical Reactions. The Journal of Physical Chemistry **81(25)** (1977) 2340–2361
5. Chao, L., Davenport, M., Forrest, S., Perelson, A.: A stochastic model of cytotoxic t cell responses. Journal Theoretical Biology **228** (2004) 227–240
6. Wilkinson, D.: A stochastic model of cytotoxic T cell responses. CRC press (2006)
7. Sorathiya, A., Liò, P., Bracciali, A.: Formal reasoning on qualitative models of coinfection of HIV and Tuberculosis and HAART therapy. BMC Bioinformatics **11(1)** (2010) Asia Pacific Bioinformatics Conference
8. Cole, M.: Algorithmic Skeletons: Structured Management of Parallel Computations. Research Monographs in Parallel and Distributed Computing. Pitman (1989)
9. Intel Threading Building Blocks. `http://software.intel.com/en-us/intel-tbb`
10. Asanovic, K., Bodik, R., Demmel, J., Keaveny, T., Keutzer, K., Kubiatowicz, J., Morgan, N., Patterson, D., Sen, K., Wawrzynek, J., Wessel, D., Yelick, K.: A view of the parallel computing landscape. CACM **52**(10) (2009) 56–67
11. Fastflow project: website. (2009) `http://mc-fastflow.sourceforge.net`
12. Aldinucci, M., Meneghin, M., Torquati, M.: Efficient smith-waterman on multicore with FastFlow. In Proc. of Intl. Euromicro PDP 2010: Parallel Distributed and network-based Processing, Pisa, Italy, IEEE (February 2010) 195–199
13. Aldinucci, M., Gorlatch, S., Lengauer, C., Pelagatti, S.: Towards parallel programming by transformation: The FAN skeleton framework. Parallel Algorithms and Applications **16**(2–3) (March 2001) 87–121