PRISM-PSY: Precise GPU-Accelerated Parameter Synthesis for Stochastic Systems *

Milan Češka[1], Petr Pilař[2], Nicola Paoletti[1], Luboš Brim[2], and Marta Kwiatkowska[1]

[1] Department of Computer Science, University of Oxford, UK
[2] Faculty of Informatics, Masaryk University, Czech Republic

Abstract. In this paper we present PRISM-PSY, a novel tool that performs precise GPU-accelerated parameter synthesis for continuous-time Markov chains and time-bounded temporal logic specifications. We redesign, in terms of matrix-vector operations, the recently formulated algorithms for precise parameter synthesis in order to enable effective data-parallel processing, which results in significant acceleration on many-core architectures. High hardware utilisation, essential for performance and scalability, is achieved by state space and parameter space parallelisation: the former leverages a compact sparse-matrix representation, and the latter is based on an iterative decomposition of the parameter space. Our experiments on several biological and engineering case studies demonstrate an overall speedup of up to 31-fold on a single GPU compared to the sequential implementation.

1 Introduction

Model checking of *continuous-time Markov chains* (CTMCs) against *continuous stochastic logic* (CSL) formulae [1, 27] has numerous applications in many areas of science. In biochemistry, there is an interest in analysing hypotheses (formulated using CSL) about reaction networks that can be adequately modelled as CTMCs governed by the Chemical Master Equation [22, 28, 9]. In engineering disciplines, CTMCs are used to study various reliability and performance aspects of computer networks [5], communication [19] and security protocols [29].

Traditionally, stochastic model checking techniques assume that model parameters – namely, the transition rate constants – are known a priori. This is often not the case and one has to consider ranges of parameter values instead, for example, when the parameters result from imprecise measurements, or when designers are interested in finding parameter values such that the model fulfils a given specification. Such problems can be effectively formulated in the framework of parameter synthesis for CTMCs [24, 10, 12]: given a time-bounded CSL formula and a model whose transition rates are functions of the parameters, find parameter values such that the satisfaction probability of the formula meets a given threshold, is maximised, or minimised. In [10, 12] we developed synthesis algorithms that yield answers that are precise up to within an arbitrarily small tolerance value. The algorithms combine the computation of probability bounds with the refinement and sampling of the parameter space.

* This work has been supported by the ERC Advanced Grant VERIWARE, and the Czech Grant Agency grants GA16-24707Y (M. Češka) and GA15-11089S (L. Brim).

The complexity of the synthesis algorithms depends mainly on the size of the underlying model and on the number of parameter regions to analyse in order to achieve the desired precision. However, existing techniques do not scale with the model size and the dimensionality of the parameter space. For instance, as reported in [12], the synthesis of two parameters for a model with 5.1K states requires the analysis of 5K parameter regions and takes around 3.6 hours.

In the last years, many-core graphical processing units (GPUs) have been utilised as general purpose, high-performance processing resources in computationally-intensive scientific applications. In light of this development, we redesign the synthesis algorithms using matrix-vector operations so as to ensure effective data-parallel processing and acceleration of the synthesis procedures on many-core architectures. The novelty of our approach is a two-level parallelisation scheme that distributes the workload for the processing of the state space and the parameter space, in order to optimally utilise the computational power of the GPU. The state space parallelisation builds on a sparse-matrix encoding of the underlying parametric CTMC. The parameter space parallelisation exploits the fact that our synthesis algorithms require the analysis of a large number of parameter regions during the parameter space refinement.

In this paper we present our new publicly available tool PRISM-PSY[3] that implements the data-parallel algorithms together with a number of optimisations of the sequential algorithms, and employs the front-end of the probabilistic model-checker PRISM [26]. We systematically evaluate the performance of PRISM-PSY and demonstrate the usefulness of our precise parameter synthesis methods on several case studies, including survivability analysis of the Google File System [16, 2]. Our experiments show that the data-parallel synthesis achieves on a single GPU up to a 31-fold speedup with respect to the optimised sequential implementation and that our algorithms provide good scalability with respect to the size of the model and the number of parameter regions to analyse. As a result, PRISM-PSY enables the application of precise parameter synthesis methods to more complex problems, i.e. larger models and higher-dimensional parameter spaces.

The main contributions of this paper can be summarised as follows: 1) improvement of the sequential algorithms of [10, 12], leading in some cases to more than 10-fold speedup; 2) formulation of a backward variant of the parametric transient analysis of [10] using matrix-vector operations, which enables data-parallel implementation; 3) combination of the state space and parameter space parallelisation in order to fully utilise the available computational power; 4) development of the PRISM-PSY tool that enables precise parameter synthesis on many-core architectures and 5) systematic experimental evaluation of the tool.

Related work. The parameter synthesis problem for CTMCs and bounded reachability specifications was first introduced in [24], where the authors resort to the analysis of the polynomial function describing how the reachability probability depends on the parameter values. Due to the high degree of the polyno-

[3] `http://www.prismmodelchecker.org/psy/`

mials (determined by the number of uniformisation steps), only an approximate solution is obtained through the discretisation of the parameter space.

The function describing how the satisfaction probability of a linear time-bounded formula depends on the parameter values can be approximated through statistical methods. A technique based on Gaussian Process regression is presented in [6] and implemented in the U-check tool [7]. In contrast to our approach, statistical methods cannot provide guaranteed precision, and thus are not suitable for safety-critical applications.

Parameter synthesis has also been studied for discrete-time Markovian models and unbounded temporal properties [23, 15]. The synthesis algorithms are based on constructing a rational function describing the satisfaction probability by performing state elimination. This approach is implemented in the tool PROPhESY [18] that supports incremental parameter synthesis using SMT techniques, but is not suitable for time-bounded specifications and CTMCs.

Our tool builds on methods for the efficient GPU parallelisation of matrix-vector multiplication [4] and probabilistic model checking [33, 8]. In our previous work [3], we showed how the algorithms for LTL model checking can be redesigned in order to accelerate verification on GPUs.

2 Precise Parameter Synthesis

In this section we summarise the parameter synthesis problem for CTMCs and time-bounded CSL properties originally introduced in [12]. We also describe the sequential synthesis algorithms of [10, 12] and the improvements implemented in the PRISM-PSY tool, which provide the foundation for the new data-parallel algorithms (Section 3) and the baseline for evaluating the parallelisation speedup.

2.1 Problem formulation

Parametric continuous-time Markov chains (pCTMCs) [24] extend the notion of CTMCs by allowing transition rates to depend on parameters. We consider pCTMCs with a finite set of states S and a finite set K of parameters ranging over closed real intervals, i.e, $[k^{\perp}, k^{\top}] \subseteq \mathbb{R}$ for $k \in K$. These induce a rectangular parameter space $\mathcal{P} = \bigtimes_{k \in K} [k^{\perp}, k^{\top}]$. Subsets of $\mathcal{P}$ are referred to as *parameter regions* or *subspaces*. Given a pCTMC and a parameter space $\mathcal{P}$, we denote with $\mathcal{C}_{\mathcal{P}}$ the set $\{\mathcal{C}_p \mid p \in \mathcal{P}\}$, where $\mathcal{C}_p$ is the instantiated CTMC obtained by replacing the parameters in the parametric rate matrix $\mathbf{R}$ with the valuation p.

In the current implementation of the tool, we support only linear rate functions of the following two forms: for any $s, s' \in S$, $\mathbf{R}(s, s') = \sum_{k \in K} k \cdot a_{k,s,s'}$ (parametric rate) or $\mathbf{R}(s, s') = b_{s,s'}$ (constant rate) where $a_{k,s,s'}, b_{s,s'} \in \mathbb{R}_{\geq 0}$. Such rate functions are sufficient to describe a wide range of systems, from biological to computer systems, as we will show in Section 4.

To specify properties over pCTMCs, we employ the time-bounded fragment of *Continuous Stochastic Logic (CSL)* [1] extended with time-bounded reward operators [27]. The current version of the tool considers only unnested formulae given by the following syntax: $\Phi ::= P_{\sim r}[\phi] \mid R_{\sim r}[C^{\leq t}]$ is a state formula, $\phi ::= \Psi\ U^{I}\ \Psi$ is a path formula, where $\Psi ::= \mathsf{true} \mid a \mid \neg\Psi \mid \Psi \wedge \Psi$, a is an atomic

proposition evaluated over states, $\sim \in \{<, \leq, \geq, >\}$, r is a probability ($r \in [0,1]$) or reward ($r \in \mathbb{R}_{\geq 0}$) threshold, $t \in \mathbb{R}_{\geq 0}$ is a time bound, and I is a time interval of $\mathbb{R}_{\geq 0}$. The future operator, F^I, can be derived as $F^I\ \Psi \equiv \mathsf{true}\ U^I\ \Psi$. Let $\vDash$ denote a satisfaction relation. Intuitively, a state $s \vDash \mathsf{P}_{\sim r}[\phi]$ iff the probability of the set of paths starting in s and satisfying ϕ meets $\sim r$. A path $\omega = s_0 t_0 s_1 t_1 \ldots$ satisfies $\Phi\ \mathsf{U}^I\ \Psi$ iff there exists a time $t \in I.(\omega@t \vDash \Psi \wedge \forall t' \in [0,t).\omega@t' \vDash \Phi)$, where $\omega@t$ denotes the state in ω at time t. A state $s \vDash \mathsf{R}_{\sim p}[\mathsf{C}^{\leq t}]$ iff the expected rewards over the path starting in s cumulated until t time units satisfies $\sim p$. We remark that the synthesis algorithms can be adapted to support the full fragment of time-bounded CSL including nested formulae, as shown in [10].

The *satisfaction function* captures how the satisfaction probability of a given property relates to the parameters and initial state. Let ϕ be a CSL path formula, $\mathcal{C}_\mathcal{P}$ be a pCTMC over a space $\mathcal{P}$ and $s \in S$. We denote with $\Lambda_\phi : \mathcal{P} \to S \to [0,1]$ the satisfaction function such that $\Lambda_\phi(p)(s)$ is the probability of the set of paths (from state s) satisfying ϕ in $\mathcal{C}_p$. The satisfaction function for reward formulae can be defined analogously and is omitted to simplify the presentation.

We consider two parameter synthesis problems: the *threshold synthesis* problem that, given a threshold $\sim r$ and a CSL path formula ϕ, asks for the parameter region where the probability of ϕ meets $\sim r$; and the *max synthesis* problem that determines the parameter region where the probability of the input formula attains its maximum, together with probability bounds approximating that maximum. Solutions to the threshold synthesis problem admit parameter points left undecided, while, in the max synthesis problem, the actual set of maximising parameters is contained in the synthesised region. The min synthesis problem is defined and solved in a symmetric way to the max case.

For $\mathcal{C}_\mathcal{P}$, ϕ, an initial state s_0, a threshold $\sim r$ and a volume tolerance $\varepsilon > 0$, the *threshold synthesis* problem is finding a partition $\{\mathcal{T}, \mathcal{U}, \mathcal{F}\}$ of $\mathcal{P}$, such that: $\forall p \in \mathcal{T} : \Lambda_\phi(p)(s_0) \sim r$; $\forall p \in \mathcal{F} : \Lambda_\phi(p)(s_0) \not\sim r$; and $\mathrm{vol}(\mathcal{U})/\mathrm{vol}(\mathcal{P}) \leq \varepsilon$, where $\mathcal{U}$ is an undecided subspace and $\mathrm{vol}(A) = \int_A 1 d\mu$ is the volume of A.

For $\mathcal{C}_\mathcal{P}$, ϕ, s_0, and a probability tolerance $\epsilon > 0$, the *max synthesis* problem is finding a partition $\{\mathcal{T}, \mathcal{F}\}$ of $\mathcal{P}$ and probability bounds $\Lambda_\phi^\perp$, $\Lambda_\phi^\top$ such that: $\forall p \in \mathcal{T} : \Lambda_\phi^\perp \leq \Lambda_\phi(p)(s_0) \leq \Lambda_\phi^\top$; $\exists p \in \mathcal{T} : \forall p' \in \mathcal{F} : \Lambda_\phi(p)(s_0) > \Lambda_\phi(p')(s_0)$; and $\Lambda_\phi^\top - \Lambda_\phi^\perp \leq \epsilon$.

Figure 1 depicts an example of threshold and max synthesis problems. On the left, the satisfaction function describes the probability of the property (y-axis) depending on the values of parameter k_1 (x-axis). In the centre plot, we highlight the parameter regions for which the threshold $\geq r$ is met ($\mathcal{T}$, green), is not met ($\mathcal{F}$, red) and is undecided ($\mathcal{U}$, yellow). On the right, the solution to the max synthesis problem is the region ($\mathcal{T}$, green) containing all the maximising parameters and whose probability bounds meet the input tolerance ϵ.

2.2 Solution of the synthesis problems

The key ingredient for solving the aforementioned synthesis problems is a procedure that takes a pCTMC $\mathcal{C}_\mathcal{P}$ and CSL path formula ϕ, and provides safe under- and over-approximations of the minimal and maximal probability that

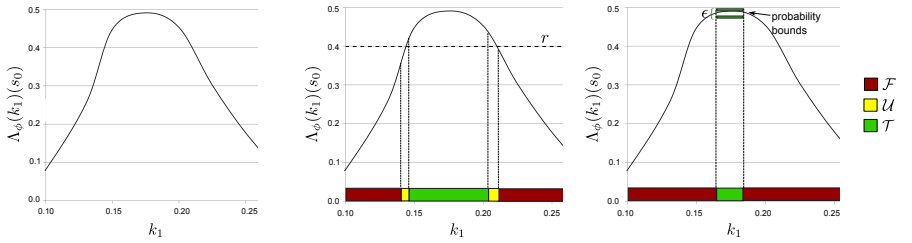


Fig. 1: **Left:** Example of a satisfaction function. **Centre:** Solution of the threshold synthesis problem for $\geq r$. **Right:** Solution of the max synthesis problem.

$\mathcal{C}_\mathcal{P}$ satisfies ϕ: for all $s \in S$, it computes bounds $\Lambda_{\min}(s)$ and $\Lambda_{\max}(s)$ such that $\Lambda_{\min}(s) \leq \inf_{p\in\mathcal{P}} \Lambda_\phi(p)(s)$ and $\Lambda_{\max}(s) \geq \sup_{p\in\mathcal{P}} \Lambda_\phi(p)(s)$. The procedure builds on a parametric transient analysis that computes safe bounds for the parametric transient probabilities in the discrete-time process derived from the CTMC. This discretisation is obtained through standard uniformisation and the Fox and Glynn algorithm [21] that is used to derive the required number of discrete steps to consider (also called uniformisation steps or iterations) for a given time bound[4]. See [10, 27] for more details.

We now summarise the algorithms for threshold and max synthesis based on the partitioning and iterative refinement of the parameter space [12]. Assume a threshold synthesis problem for a path formula ϕ with threshold $\geq r$. At each step, the algorithm refines the undecided parameter subspace $\mathcal{U}$, starting from $\mathcal{U} = \mathcal{P}$: it generates a partition $\mathcal{D}$ of $\mathcal{U}$ and, for each $\mathcal{R} \in \mathcal{D}$, computes the safe probability bounds $\Lambda^\mathcal{R}_{\min}$ and $\Lambda^\mathcal{R}_{\max}$ of the corresponding pCTMC $\mathcal{C}_\mathcal{R}$. If $\Lambda^\mathcal{R}_{\min} \geq r$, then the satisfaction of the threshold is guaranteed for the whole region $\mathcal{R}$, which is hence added to $\mathcal{T}$. Otherwise, the algorithm tests whether $\mathcal{R}$ can be added to $\mathcal{F}$ by checking if $\Lambda^\mathcal{R}_{\max} < r$. If $\mathcal{R}$ is neither in $\mathcal{T}$ nor in $\mathcal{F}$, it forms an undecided subspace that is added to the set $\mathcal{U}$. If the volume tolerance ε is not met, the algorithm proceeds to the next iteration, where $\mathcal{U}$ is further refined. The refinement procedure guarantees termination since the over-approximation $[\Lambda^\mathcal{R}_{\min}, \Lambda^\mathcal{R}_{\max}]$ can be made arbitrarily precise by reducing the volume of $\mathcal{R}$ [13].

In the max synthesis case, the algorithm starts from $\mathcal{T} = \mathcal{P}$ and iteratively refines $\mathcal{T}$ until the tolerance ϵ is met. Let $\mathcal{D}$ be the partition of $\mathcal{T}$ at a generic step. The algorithm rules out from $\mathcal{T}$ subspaces that are guaranteed to be in $\mathcal{F}$, by deriving an under-approximation M of the maximum satisfaction probability. Indeed, for $\mathcal{R} \in \mathcal{D}$, $\Lambda^\mathcal{R}_{\max} < M$ implies that $\mathcal{R}$ is in $\mathcal{F}$. M is derived by sampling a set of parameter values from the region $\mathcal{R}$ with the highest $\Lambda^\mathcal{R}_{\min}$ and taking the highest value of the satisfaction function over these values.

Improvements on the sequential algorithms. The PRISM-PSY tool introduces several improvements on the prototype implementations used in [10, 12]. Here we present those having the most significant impact on performance.

1) Backward computation of probabilistic bounds. In [10, 12], the probability bounds $\Lambda_{\min}$ and $\Lambda_{\max}$ are computed using a forward variant of the parametric

[4] The Fox and Glynn algorithm returns a finite bound on the number of steps needed to approximate transient probabilities up to a specified precision.

transient analysis, which requires a separate computation of the bounds for each initial state. In our tool, we also implemented a more efficient solution that requires only a single computation for all states, based on backward computation.

2) Adaptive Fox-Glynn. While in previous implementations the number of uniformisation steps was fixed and obtained using the maximum exit rate (sum of outgoing rates per a state) of the whole parameter space, the *adaptive Fox-Glynn* technique computes the number of steps for each subregion separately, using the maximum exit rate of the inspected subregion. For large parameter spaces, this technique can significantly decrease the overall number of uniformisation steps, improving the performance by more than a factor of two.

3) Refinement strategies. The tool employs improved refinement algorithms that can decrease the total number of subregions to analyse. Specifically, for threshold synthesis, at each step only the undecided subregions with the largest volume are refined while, for max synthesis, only the regions with either the lowest lower probability bound ($\Lambda_\phi^\perp$) or the highest upper bound ($\Lambda_\phi^\top$).

3 Data-Parallel Algorithms for Parameter Synthesis

In this section we first introduce the basic concepts of the target hardware architecture, i.e. modern general-purpose GPUs. We then formulate the backward variant of the parametric transient analysis using matrix-vector operations, and describe the sparse-matrix representation of pCTMCs. Finally, we present a two-level parallelisation of the synthesis algorithms. A detailed description of the data-parallel algorithms for parameter synthesis can be found in [30].

3.1 Computational model for modern GPUs

Typical GPUs consist of multiple *Streaming Multiprocessors* (SMs), with each SM following a *single instruction multiple threads* (SIMT) model. This approach establishes a hierarchy of threads prior to the actual computation. Within this hierarchy, threads are arranged into blocks that are assigned for parallel execution on SMs. Threads are hardwired into groups of 32 called warps, which form a basic scheduling unit and execute instructions in a lock-step manner. If a sufficient number of threads is dispatched, each SM maintains a set of active warps to hide the memory access latency and maximise utilisation of its functional units.

The SIMT approach supports code divergence within threads of the warp, but this usually causes significant performance degradation due to the serialisation of the execution. Another characteristic of GPUs that significantly affects their performance is the way in which simultaneous memory requests from multiple threads in a warp are handled. Requests exhibiting spatial locality are maximally *coalesced*. Simply stated, accesses to consecutive addresses are served by a single memory fetch as long as they are in the same memory segment.

A typical GPU program consists of a *host* code running on the CPU and a *device* code running on the GPU. The device code is structured into *kernels* that execute the same scalar sequential program in many independent data-parallel threads. The combination of out-of-order CPU and data-parallel processing GPU allows for heterogeneous computation.

3.2 Backward computation of probability bounds

For a pCTMC $\mathcal{C}_{\mathcal{R}}$ over a region $\mathcal{R} = \bigtimes_{k \in K} [k^{\perp}, k^{\top}]$ and a target set $A \subseteq S$, the parametric backward analysis computes a series of vectors $\sigma_i^{\min}$ and $\sigma_i^{\max}$ such that, for all $s \in S$, $\sigma_i^{\min}(s) \leq \inf_{p \in \mathcal{P}} \sigma_{i,p}(s)$ and $\sigma_i^{\max}(s) \geq \sup_{p \in \mathcal{P}} \sigma_{i,p}(s)$, where $\sigma_{i,p}(s)$ is the probability that, starting from the state s, a state in A is reached after i discrete steps in $\mathcal{C}_p$. From these vectors, the probability bounds $\Lambda_{\min}(s)$ and $\Lambda_{\max}(s)$ are computed in a similar way to non-parametric CTMCs [27].

We define a matrix-vector operator $\diamond$ that computes the vector $\sigma_{i+1}^{\max}$ from $\sigma_i^{\max}$ and the parametric rate matrix $\mathbf{R}$ as $\sigma_{i+1}^{\max}(s) = (\mathbf{R} \diamond \sigma_i^{\max})(s)$, where $\sigma_0^{\max}(s) = 1$ if $s \in A$ and 0 otherwise. An analogous operator can be defined for $\sigma_{i+1}^{\min}$. Similarly to standard uniformisation, the definition of $\diamond$ exploits the uniformised matrix, which is, in our case, parametric. For each $s \in S$, $\sigma_{i+1}^{\max}(s)$ is first expressed by maximising the probability in s stepwise, i.e. after the i-th step. Below, we expand the definition of the uniformised matrix using the uniformisation rate q given by the maximal exit rate and the time bound [21, 27]:

$$\sigma_{i+1}^{\max}(s) = \max_{p \in \mathcal{R}} \left(\sum_{s' \in S \setminus \{s\}} \sigma_i^{\max}(s') \frac{\mathbf{R}_p(s, s')}{q} + \sigma_i^{\max}(s) \left(1 - \sum_{s' \in S \setminus \{s\}} \frac{\mathbf{R}_p(s, s')}{q} \right) \right) \quad (1)$$

where $\mathbf{R}_p$ is the rate matrix instantiated with parameter p. The first sum represents the probability of entering a state $s' \neq s$ and, from there, reaching A in i steps. The second sum is the probability of staying in s and, from there, reaching A in i steps. By expanding the parametric rate matrix $\mathbf{R}$ in Equation 1 we get:

$$\sigma_{i+1}^{\max}(s) = \sigma_i^{\max}(s) + \sum_{s' \in S \setminus \{s\}} \frac{\sigma_i^{\max}(s') - \sigma_i^{\max}(s)}{q} \cdot \begin{cases} \sum_{k \in K} k^{\star} \cdot a_{k,s,s'} & (2) \\ b_{s,s'} & (3) \end{cases}$$

where $k^{\star} = k^{\top}$ if $\sigma_i^{\max}(s') > \sigma_i^{\max}(s)$ and $k^{\perp}$ otherwise. These equations allow us to compute the vector $\sigma_{i+1}^{\max}$ using matrix-vector operations, as shown in the implementation of $\diamond$ in Algorithm 1. Note that Equation 2 is used when the transition from s to s' has a parametric rate, while Equation 3 is used when it has a constant rate.

An approximation error is introduced because $\sigma_{i+1}^{\max}$ is computed by optimising $\sigma_{i+1,p}$ locally, i.e. at each step and at each state, and the error accumulates at each uniformisation step. We examine this error and its convergence in [13]. The forward variant of the parametric transient analysis can also be formulated using a vector-matrix operator [10], but the resulting code has more complex control flow and higher branch divergence, which makes parallelisation less efficient.

3.3 Sparse-matrix representation of parametric CTMCs

We introduce a sparse-matrix representation of parametric CTMCs that allows us to implement the operator $\diamond$ in such a way that the resulting program has a similar control flow and memory access pattern as the standard matrix-vector multiplication, for which efficient data-parallel implementations exist [4, 33, 8].

We represent the data in a compact format based on the *compressed sparse row* (CSR) matrix format. The CSR format stores only the non-zero values of the rate matrix $\mathbf{R}$ using three arrays: non-zero values, their column indices, and row beginnings. The CSR format is also used in the PRISM tool as the fastest explicit representation for CTMCs [26].

To handle the non-parametric transitions separately in a more efficient way, we decompose $\mathbf{R}$ into the non-parametric matrix, stored in the CSR format, and the parametric matrix. To enable an efficient data-parallel implementation of the operator $\diamond$, for a region $\mathcal{R} = \bigtimes_{k \in K} [k^{\perp}, k^{\top}]$ and for each parametric transition rate $\mathbf{R}(s, s')$ two quantities, $r^{\perp}_{s,s'} = \sum_{k \in K} k^{\perp} \cdot a_{k,s,s'}$ and $r^{\top}_{s,s'} = \sum_{k \in K} k^{\top} \cdot a_{k,s,s'}$, are stored. From Equation 2, it is enough to test $\sigma_i^{\max}(s') - \sigma_i^{\max}(s) > 0$ to decide whether to use $r^{\top}_{s,s'}$ or $r^{\perp}_{s,s'}$ in the multiplication, as illustrated in Algorithm 1.

In the parallel version, we provide an additional implementation using data structures based on the ELLPACK (ELL) sparse matrix representation [4]. The advantage of ELL over CSR is that it provides a single-stride aligned access to the data arrays, meaning that memory accesses within a single warp are reasonably coalesced. ELL yields better performance than CSR for some problems.

3.4 GPU parallelisation

We implemented PRISM-PSY in *Open Computing Language* (OpenCL) [32]. In contrast to other programming frameworks, OpenCL supports multiple platforms and GPUs, and thus provides better portability. Moreover, its performance is comparable with that of specialised frameworks (e.g. CUDA [20]).

The synthesis algorithms are executed in a heterogeneous way. The sequential refinement procedure is executed on the CPU. For each parameter region $\mathcal{R}$ to analyse, the CPU prepares a kernel that computes the probability bounds $\Lambda^{\mathcal{R}}_{\min}$ and $\Lambda^{\mathcal{R}}_{\max}$ on the GPU, based on the backward parametric transient analysis described above. Following [4, 33, 8], we implement a *state space parallelisation*, i.e. a single row of the rate matrix (corresponding to the processing of a single state) is mapped to a single computational element. Note that the models we consider typically have a balanced distribution of the state successors, and thus yield a balanced distribution of non-zero elements in the rows of the matrix. This ensures a good load balancing within the warps and blocks.

In the case of models with small numbers of states, this parallelisation is not able to efficiently utilise all computational elements, since some of them will be idle during the kernel execution. To overcome this potential performance degradation, we combine state space parallelisation with *parameter space parallelisation* that computes the probability bounds for multiple parameter regions in parallel. As demonstrated in the experimental evaluation (Section 4), this *two-level* parallelisation significantly improves performance on small models. In many cases, this solution can improve the runtime of large models too, because it allows the thread scheduler to better hide memory latency.

Since parallel kernel execution is unsupported by many GPU devices or it may fundamentally decrease performance, we provide a way to perform, in a single compute kernel, multiple matrix-vector operations over multiple parameter

Algorithm 1 Kernel for two-level CSR parallelisation of the $\diamond$ operator

For all $0 \leq n < |S|$ and $0 \leq m <$ number of parallel regions, run in parallel:

1: $e :=$ number of non-zero elements in $\mathbf{R}$
2: **for** $j :=$ `matRowBeg`$[n]$; $j <$ `matRowBeg`$[n+1]$; $j := j+1$ **do**
3: $\quad dMax := \sigma_i^{\max}[m * |S| + \texttt{matCol}[j]] - \sigma_i^{\max}[m * |S| + n]$
4: $\quad$ **if** $dMax > 0$ **then** $\sigma_{i+1}^{\max}[m * |S| + n] \mathrel{+}= dMax * \texttt{matValTop}[m * e + j]$
5: $\quad$ **else** $\sigma_{i+1}^{\max}[m * |S| + n] \mathrel{+}= dMax * \texttt{matValBot}[m * e + j]$

regions. The solution exploits the fact that, in our case, the rate matrices for different regions have the same structure and only differ in the values of $r^{\top}_{s,s'}$ and $r^{\perp}_{s,s'}$. We extend the sparse-matrix representation of the pCTMC and store the values $r^{\perp}_{s,s'}$ and $r^{\top}_{s,s'}$ as well as $\sigma_i^{\max}(s)$ and $\sigma_{i+1}^{\max}(s)$ for all regions. This allows us to utilise $m \cdot |S|$ computational elements for m parallel regions. Algorithm 1 illustrates the kernel for the two-level parallelisation using the CSR format. The vectors `matRowBeg` and `matCol`, the same for all regions, keep the column indices and row beginnings, respectively. The vectors `matValTop` and `matValBot` keep the non-zero values of $r^{\top}_{s,s'}$ and $r^{\perp}_{s,s'}$, respectively. We store only the current $\sigma_i^{\max}$ and $\sigma_{i+1}^{\max}$ using two vectors and the vectors are swapped between the iterations.

Importantly, merging the computation for multiple regions requires modifying the adaptive Fox-Glynn technique to consider the highest uniformisation step among them. This means that the benefits of adaptive Fox-Glynn diminish with the number of subspaces processed in parallel.

4 Experimental Evaluation

In this section we evaluate the performance of the data-parallel synthesis algorithms on case studies of biological and computer systems. We discuss how model features affect parallelisation and show that parameter synthesis can be meaningfully employed to analyse various requirements, ranging from quality of service to the reliability of synthetic biochemical networks.

All the experiments were run on a 4-core Linux workstation with an AMD Phenom$^{\text{TM}}$ II X4 940 Processor @ 3GHz, 8 GB DDR2 @ 1066 MHz RAM and an NVIDIA GeForce GTX 480 GPU with 1.5 GB of GPU memory. The GPU has 14 SMs, each having 32 cores and the capability to maintain up to 48 active warps. Therefore, the GPU can simultaneously maintain and schedule up to 21504 active threads to maximise the utilisation of its computational elements.

In the following, Java denotes the optimised sequential implementation and Csr$_n$ (Ell$_n$) the data-parallel implementation based on CSR (ELL) with n subregions being processed in parallel. We report only an approximate value for the number of final subregions, since it differs slightly in some experiments due to parallel processing. We also report the results for the parameter space parallelisation only up to the best performance is reached.

4.1 Google File System

We consider the performance evaluation case study of the replicated file system used in the Google search engine known as Google File System (GFS). The model

was first introduced as a generalised stochastic Petri net (GSPN) [16] and then translated to a CTMC [2]. Previous work on the model focused on survivability analysis, i.e. the ability of the system to recover from disturbances or disasters, and considers all model parameters to be fixed. Here, we work with a pCTMC model and show how parameter synthesis can be used to examine survivability.

Figure 2 illustrates the GSPN model of the GFS. Default values for stochastic rates can be found in [16, 2]. Files are divided into chunks of equal size. Each chunk exists in several copies, located in different chunk servers. There is one master server that is responsible for keeping the locations of the chunk copies, monitoring the chunk servers and replicating the chunks. The master can be: up and running (token at M_up); or failed (M1), due to a software (M_soft_d) or hardware (M_hard_d) problem. The model reproduces the life-cycle of a single chunk: the numbers of available and lost copies are given by places R_present and R_lost, respectively. Lost chunks are replicated through transition replicate. R is the maximum number of copies. We consider M chunk servers whose behaviour is analogous to that of the master. When a chunk server fails, a chunk copy is lost (destroy) or not (keep) depending on whether the server is storing the single chunk under consideration. We set $M = 60$ and $R = 3$, yielding a model with 21.6K states and 145K transitions.

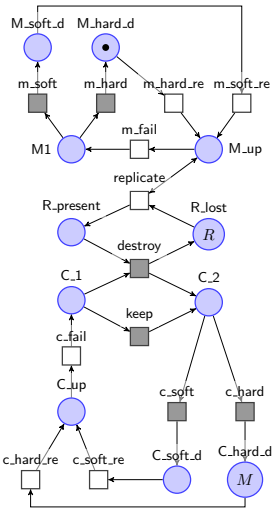
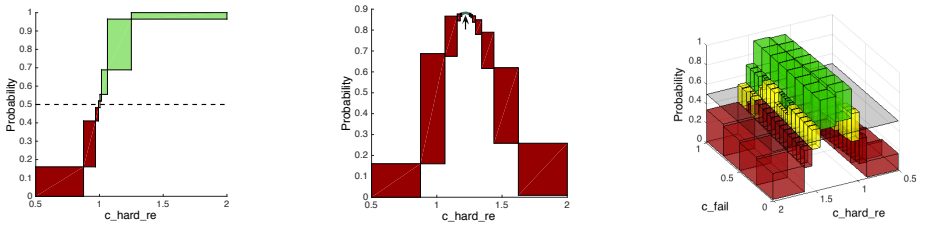


Fig. 2: Google File System from [16, 2]. Transitions can be immediate (grey) or timed (white).

We first formulate a threshold synthesis problem for the CSL formula $\phi_{\mathsf{GFS}_1} = F^{[0,60]}\ \mathsf{SL}_3$, where $\mathsf{SL}_3 = (\mathsf{M\_up} = 1$ and $\mathsf{R\_present} \geq 3)$ is the QoS requirement that the master is running and at least three chunk copies are available (service level 3). The initial state models a severe hardware disaster: all the servers are down due to hardware ($\mathsf{C\_hard\_d} = M$ and $\mathsf{M\_hard\_d} = 1$) and all the chunk copies have been lost ($\mathsf{R\_lost} = R$). We are interested in synthesising the values of parameter c_hard_re, that is, the rate at which chunk servers are repaired from hardware failure. Importantly, c_hard_re can actually be controlled, e.g. by intensifying the frequency of technical interventions. Figure 3(a) illustrates the synthesis results for $\mathsf{c\_hard\_re} \in [0.5, 2]$ and probability threshold ≥ 0.5. The property is met for any c_hard_re above 1, and, in particular, SL_3 is reached with high probability for repair rates above 1.25.

We now evaluate a property requiring that SL_3 is reached strictly within the time interval $[40, 60]$: $\phi_{\mathsf{GFS}_2} = \neg\mathsf{SL}_3\ U^{[40,60]}\ \mathsf{SL}_3$. Although it is generally sought to reach the required QoS as soon as possible, this property can be used in scenarios like planned downtime, where the service does not need to be up before the time scheduled for maintenance. In Figure 3(b), we report the results of max synthesis for parameter c_hard_re. The maximising parameters



(a) (b) (c)

Fig. 3: Synthesis results for the GFS model. Each box denotes a parameter region (width and depth) and its probability bounds (height). Colour code is as in Figure 1. (a) Threshold synthesis, property ϕ_{GFS_1}, threshold ≥ 0.5 (dashed line) and volume tolerance $\varepsilon = 0.01$. (b) Max synthesis, property ϕ_{GFS_2} and probability tolerance $\epsilon = 0.01$. (c) Threshold synthesis, property ϕ_{GFS_2}, threshold ≥ 0.5 (semi-transparent plane) and $\varepsilon = 0.1$. Parameter domains are $\mathsf{c\_hard\_re} \in [0.5, 2]$ (a,b,c) and $\mathsf{c\_fail} \in [0.01, 1]$ (c). Numbers of final regions are 8 (a), 24 (b) and 136 (c).

Threshold synthesis ϕ_{GFS_1}			Max synthesis fϕ_{GFS_2}			Threshold synthesis ϕ_{GFS_2}		
Impl.	Time (s)	Speedup	Impl.	Time (s)	Speedup	Impl.	Time (s)	Speedup
JAVA	842	1.0	JAVA	3279	1.0	JAVA	12221	1.0
CSR$_1$	56	15.0	CSR$_1$	257	12.8	CSR$_1$	764	16.0
CSR$_4$	51	16.5	CSR$_4$	239	13.7	CSR$_8$	660	18.5
CSR$_{16}$	51	16.5	CSR$_8$	211	15.5	CSR$_{64}$	636	19.2
ELL$_{16}$	**41**	**20.5**	ELL$_{32}$	**207**	**15.8**	ELL$_{16}$	**505**	**24.2**

Table 1: Performance of the GFS model: 21.6K states, 145K transitions, and $\leq$47K iterations per subregion. Details of the synthesis problems are reported in Figure 3.

(indicated with a black arrow) are found in the region approximately given by $\mathsf{c\_hard\_re} \in [1.2, 1.23]$, since for high repair rates SL_3 is reached too early.

In the last experiment, we introduce one additional parameter, c_fail, i.e. the rate at which any failure (hardware or software) occurs in a chunk server. Since the GFS is designed to run on cheap commodity hardware, this rate can be controlled indirectly through the reliability of the machines used. We consider a threshold synthesis problem with property ϕ_{GFS_2} and threshold ≥ 0.5. Results in Figure 3(c) evidence that, interestingly, the satisfaction probability is almost independent from the failure rate, except when c_fail approaches 1, and thus slightly higher repair rates are needed.

Table 1 reports the performance of the tool on the above experiments, namely, the speedup achieved by the data-parallel algorithms. Although the state space parallelisation utilises the GPU sufficiently (enough threads are dispatched), the parameter space parallelisation further improves performance, providing up to 24-fold and 16-fold speedup with respect to the sequential algorithm for threshold and max synthesis, respectively. The efficiency of the parameter space parallelisation depends on the effective usage of GPU resources, and thus the speedup does not scale with respect to the number of regions processed in parallel. In this case the adaptive Fox-Glynn technique does not bring any benefit, since the parameters we analyse do not affect the maximal exit rate.

$\phi_{\mathsf{SIR}}(100, 120)$			$\phi_{\mathsf{SIR}}(100, 200)$		
Impl.	Time (s)	Speedup	Impl.	Time (s)	Speedup
JAVA	918	1.0	JAVA	3117	1.0
CSR_1	363	2.5	CSR_1	303	10.3
CSR_4	269	3.4	CSR_4	351	8.8
CSR_{16}	207	4.4	CSR_{16}	315	9.5
CSR_{128}	167	5.5	CSR_{64}	303	10.3
ELL_{128}	**162**	**5.6**	ELL_{128}	**299**	**10.4**

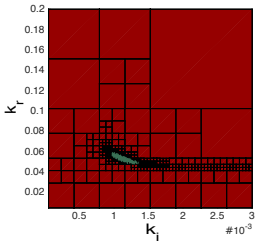


Table 2: Max synthesis for the epidemic model: parameter space $\mathcal{P}_{\mathsf{SIR}}$ and probability tolerance $\epsilon = 1\%$. $\phi_{\mathsf{SIR}}(100, 120)$: 5.1K states, 10K transitions and $\leq$ 3.1K iterations per region and $\sim$3.1K final regions. $\phi_{\mathsf{SIR}}(100, 200)$: 20K states, 40K transitions and $\leq$12K iterations per region and $\sim$826 final regions (depicted on the right).

4.2 Epidemic Model

We further consider the stochastic epidemic model we analysed in [12] using the prototype implementation, in order to evaluate the enhancements of the sequential implementation presented in this paper. It describes the epidemic dynamics of susceptible (S), infected (I) and recovered (R) individuals using the following biochemical reactions network with mass action kinetics: $S + I \xrightarrow{k_i} I + I$ and $I \xrightarrow{k_r} R$. With a total population of 100 individuals and initial state $S = 95, I = 5$ and $R = 0$, the model has 5.1K states and 10K transitions. We consider the same max synthesis problem as in [12]: parameter space $\mathcal{P}_{\mathsf{SIR}} = k_i \times k_r \in [0.005, 0.3] \times [0.005, 0.2]$ and property $\phi_{\mathsf{SIR}}(t_1, t_2) = (I > 0)\ U^{[t_1,t_2]}\ (I = 0)$, expressing that the infection lasts at least t_1 time units but dies out before time t_2. As shown in [12], for $t_1 = 100$ and $t_2 = 120$, the prototype implementation produced around 5K final parameter subspaces and required 3.6 hours.

Table 2 (left) lists the results obtained with PRISM-PSY on the same synthesis problem. We can see that the optimised sequential implementation is about 14-fold faster (918 sec. vs 3.6 h.). This significant acceleration is explained by: more sophisticated refinement strategy for max/min synthesis, which reduces the number of final regions to 3K ($\sim$2-fold speedup); the adaptive Fox-Glynn technique, which reduces the number of iterations ($\sim$ 2.5-fold speedup); and more efficient data structures that accelerate the computation of the probability bounds as well as the refinement procedure ($\sim$3-fold speedup). Note that the actual benefits of these enhancements essentially depend on the structure of the model and the synthesis problem. As the epidemic model is relatively small, the state space parallelisation is not able to sufficiently utilise the GPU, and thus the CSR_1 implementation provides only a 2.5-fold speedup. The parameter space acceleration further improves the speedup to 5.6 (ELL_{128} implementation).

We now consider a more complicated variant of the problem, where we double the population size and extend the time horizon to $t_2 = 200$. Results are presented in Table 2 (middle). In this case, the state space parallelisation sufficiently utilises the GPU and for CSR_1, we obtain a 10.3-fold speedup. On the other hand, the parameter space parallelisation reduces the benefits of the adaptive Fox-Glynn technique, and thus overall performance is improved only slightly. Table 2 (right) depicts the results of max synthesis for the larger variant.

The variant from [14]			The larger variant		
Impl.	Time (s)	Speedup	Impl.	Time (s)	Speedup
JAVA	16482	1.0	JAVA	95466	1.0
CSR$_1$	868	19.0	CSR$_1$	3870	24.7
CSR$_2$	890	18.5	CSR$_2$	3949	24.2
CSR$_4$	866	19.0	CSR$_4$	3946	24.2
ELL$_4$	**666**	**24.7**	ELL$_4$	**3065**	**31.1**

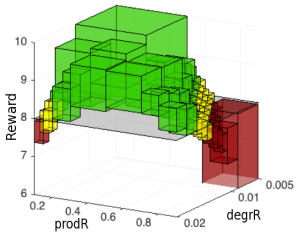


Table 3: Threshold synthesis for the signalling model: property Φ_{SIG}, parameter space $\mathcal{P}_{\mathsf{SIG}}$ and $\varepsilon = 9\%$. The variant from [14]: 116K states, 954K transitions, $\leq 19K$ iterations per region and $\sim$70 final regions (depicted on the right). The larger variant: 424K states, 3.6M transitions, $\leq$ 24K iterations per region, and $\sim$67 final regions.

4.3 Signalling in Prokaryotic Cells

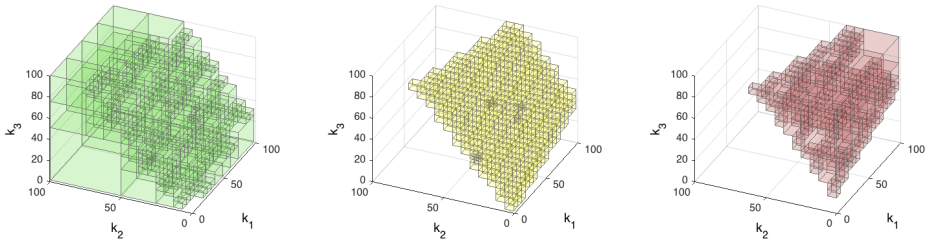
This model was introduced in [31, 14] and describes a two-component signalling pathway in prokaryotic cells with two signalling components both in phosphorylated and dephosphorylated forms: the histidine kinase H and Hp, and the response regulator R and Rp. In this case, parameter synthesis is computationally very demanding, since the model has 116K states and 954K transitions. We consider a threshold synthesis problem that requires a relatively small number of refinements, in order to demonstrate the benefits of the state space parallelisation. We synthesise the production and degradation rates ($prodR$ and $degrR$) of R such that the input noise of response regulators, defined as a quadratic deviation from the average population, is below 9 at least 80% of the time. This can be formalised as the cumulative reward property $\Phi_{\mathsf{SIG}} = R_{\geq 0.8t}[C^{\leq t}]$, where the reward in state is 1 if it satisfies $(R + Rp - avg)^2 < 9$, otherwise the reward is 0. We consider $t = 10$, $avg = 30$ and parameter space $\mathcal{P}_{\mathsf{SIG}} = prodR \times degrR \in [0.1, 0.9] \times [0.005, 0.02]$, which reflects the setting in [14].

As shown in Table 3, a speedup up to 24.7 is obtained using ELL$_4$, which further improves the GPU utilisation and the memory access pattern of the pure state space parallelisation. We also consider a larger variant of the model (about 3.6-times), for which we obtain an even better speedup (up to 31.1-fold), so demonstrating good scalability of the data-parallel algorithm. Table 3 (right) depicts the synthesis results for the small variant, evidencing the non-monotonicity of the satisfaction function for the reward property.

4.4 Approximate Majority

The next model describes a chemical reaction network that computes the *approximate majority* – the asymptotically fastest way to approximate a common decision by all members of a population [11]. We consider the network $AM_{3,3}\#39$: $A{+}B \xrightarrow{k_1=92.9} X{+}X$, $A{+}X \xrightarrow{k_2=26.2} A{+}A$ and $B{+}X \xrightarrow{k_3=23.3} B{+}B$ synthesised in [17] as the best network utilising only 3 species. The structure of the network has been synthesised using an approach based on bounded model checking and the kinetic parameters estimated by Monte Carlo-based optimisation.

As in [17], we consider small numbers of input molecules ($A = 10$, $B = 4$ and $X = 0$), and thus the model has only 120 states and 273 transitions. This



(a) True region (b) Undecided region (c) False region

Fig. 4: Threshold synthesis for the approximate majority model and property Φ_{AM}.

experiment, in contrast to the previous case studies, allows us to demonstrate the performance of our tool on very small models. We synthesised the parameters such that the probability of the correct decision being made after 100 time units is at least 95%. The property is formalised as $\Phi_{\mathsf{AM}} = P_{\geq 0.95}[F^{[100,100]}\ \Psi_{correct}]$ and the parameter space is $\mathcal{P}_{\mathsf{AM}} = k_1 \times k_2 \times k_3 \in [1, 100]^3$.

Due to the low number of states, the state space parallelisation utilises only a small portion of the computational elements of the GPU. Therefore, the GPU parallelisation using a small number of parallel parameter regions slows down the computation, as shown in Table 4. Increasing the number of parallel regions (up to 128) improves the GPU utilisation, and hence performance, yielding up to 5.7-fold speedup.

	Runtime (s)	Speedup
JAVA	1097	1.0
CSR$_1$	11375	0.1
CSR$_4$	3057	0.4
CSR$_{16}$	951	1.2
CSR$_{64}$	319	3.4
CSR$_{128}$	195	5.6
ELL$_{128}$	**193**	**5.7**

Table 4: Threshold synthesis for the approximate majority: property Φ_{AM}, parameter space $\mathcal{P}_{\mathsf{AM}}$ and $\varepsilon = 10\%$. 120 states, 273 transitions, $\leq$ 700K iterations per region and $\sim$ 911 final regions.

This experiment also demonstrates that, in contrast to the Monte Carlo-based optimisation, precise parameter synthesis provides detailed information about the impact of parameters on the probability of correct decision, as shown in Figure 4. Note that the backward transient analysis implemented in our tool computes the probability bounds for all the reachable states, in this case all the inputs satisfying $A + B = 14$.

4.5 Workstation Cluster

Finally, we consider a model describing a cluster of workstations consisting of two sub-clusters with N workstations in each, connected in a star topology [25, 34]. Both sub-clusters have their own switch that connects the workstations in the sub-cluster with a central backbone. The cluster maintains the minimum quality of service if at least 75% of the workstations are operational and connected. We assume that one can control the workstation inspection (ws_check), repair (ws_repair) and failure (ws_fail) rates.

	Runtime (s)	Speedup
JAVA	12074	1.0
CSR$_1$	**637**	**19.0**
CSR$_4$	674	17.9
ELL$_4$	672	18.0

Table 5: Threshold synthesis for the cluster model: property Φ_{CLU}, parameter space $\mathcal{P}_{\mathsf{CLU}}$ and $\varepsilon = 10\%$. 86K states, 415K transitions, $\leq$9K iterations per region and $\sim$ 273 final regions.

We synthesise the parameters such that the minimum quality of service is not maintained at most 0.1% of the time. This is formalised as $\Phi_{\mathsf{CLU}} = R_{\leq 0.1 \cdot t}[C \leq t]$, and associating a reward of 1 to states where the minimum quality of service is not provided. In this experiment, we use $t = 100$ and parameter space $\mathcal{P}_{\mathsf{CLU}} = ws\_check \times ws\_repair \times ws\_fail \in [5, 20] \times [0.5, 5] \times [0.001, 0.02]$.

Table 5 presents the results for $N = 48$ (85K states and 415K transitions). In this model, the parameter space parallelisation considerably reduces the benefits of the adaptive Fox-Glynn technique, and thus the CSR_1 implementation provides the best performance, leading to a 19-fold speedup.

4.6 Result analysis

State space parallelisation improves the scalability of the computation with respect to the model size. The experiments demonstrate that the speedup compared to the sequential baseline tends to improve with the number of states (see Tables 2 and 3). On the other hand, for smaller models, an insufficient number of threads is dispatched, leading to performance degradation (see Table 4). In most cases, the ELL format moderately outperforms the CSR format and it also works better with the parameter space parallelisation due to the more coalesced memory access pattern. Since the refinement procedure (running solely on CPU) is more complicated for max/min synthesis, for these instances the overall speedup is lower than that for threshold synthesis.

Parameter space parallelisation allows us to efficiently utilise the GPU even for small models. It scales well up to reaching the maximal number of active threads that can be dispatched (see Table 4). In practice, performance usually increases even beyond this point, since the parameter space parallelisation can improve the memory access locality. On the other hand, it mitigates the advantage of the adaptive Fox-Glynn technique, which can lead to performance degradation, as reported in Table 5. Importantly, PRISM-PSY can also be configured to perform this parallelisation using multi-core CPUs (not discussed here).

Our experiments clearly indicate that the tool is able to provide good **scalability with respect to the number of computational elements**. Since the two-level parallelisation can tune the GPU utilisation for various synthesis problems, we expect that the execution of the tool on new generations of GPUs with a larger number of cores will lead to a further improvement in acceleration.

5 Conclusion

We have introduced the tool PRISM-PSY that performs precise parameter synthesis for CTMCs and time-bounded specifications. In order to overcome the high computational demands, we have developed data-parallel versions of the algorithms allowing us to significantly accelerate synthesis on many-core GPUs. As a result, the tool provides up to 31-fold speedup with respect to the optimised sequential implementation, and thus considerably extends the applicability of precise parameter synthesis. In future we will extend the tool to support the full fragment of time-bounded CSL and multi-affine rate functions [13].

References

1. Adnan Aziz, Kumud Sanwal, Vigyan Singhal, and Robert Brayton. Verifying Continuous Time Markov Chains. In *Computer Aided Verification (CAV)*, volume 1102 of *LNCS*, pages 269–276. Springer, 1996.
2. Christel Baier, Ernst Moritz Hahn, Boudewijn R Haverkort, Holger Hermanns, and J-P Katoen. Model checking for performability. *Mathematical Structures in Computer Science*, 23(04):751–795, 2013.
3. Jiří Barnat, Petr Bauch, Luboš Brim, and Milan Češka. Designing Fast LTL Model Checking Algorithms for Many-Core GPUs. *Journal of Parallel and Distributed Computing*, 72(9):1083–1097, 2012.
4. Nathan Bell and Michael Garland. Efficient sparse matrix-vector multiplication on CUDA. NVIDIA Technical Report NVR-2008-004, NVIDIA Corporation, 2008.
5. Gunter Bolch, Stefan Greiner, Hermann de Meer, and Kishor S Trivedi. *Queueing networks and Markov chains: modeling and performance evaluation with computer science applications*. John Wiley & Sons, 2006.
6. Luca Bortolussi, Dimitrios Milios, and Guido Sanguinetti. Smoothed model checking for uncertain continuous time markov chains. *CoRR ArXiv*, 1402.1450, 2014.
7. Luca Bortolussi, Dimitrios Milios, and Guido Sanguinetti. U-check: Model checking and parameter synthesis under uncertainty. In *Quantitative Evaluation of Systems*, volume 9259 of *Lecture Notes in Computer Science*, pages 89–104. Springer, 2015.
8. Dragan Bošnački, Stefan Edelkamp, Damian Sulewski, and Anton Wijs. Parallel probabilistic model checking on general purpose graphics processors. *International Journal on Software Tools for Technology Transfer*, 13(1):21–35, 2010.
9. Luboš Brim, Milan Češka, and David Šafránek. Model checking of biological systems. In *Formal Methods for Dynamical Systems (SFM)*, volume 7938 of *LNCS*, pages 63–112. Springer, 2013.
10. Luboš Brim, Milan Češka, Sven Dražan, and David Šafránek. Exploring parameter space of stochastic biochemical systems using quantitative model checking. In *Computer Aided Verification (CAV)*, volume 8044 of *LNCS*, pages 107–123. Springer, 2013.
11. Luca Cardelli and Attila Csikász-Nagy. The cell cycle switch computes approximate majority. *Scientific Reports*, 2, 2012.
12. Milan Češka, Frits Dannenberg, Marta Kwiatkowska, and Nicola Paoletti. Precise parameter synthesis for stochastic biochemical systems. In *Computational Methods in Systems Biology (CMSB)*, volume 8859 of *LNCS*, pages 86–98. Springer, 2014.
13. Milan Češka, Frits Dannenberg, Nicola Paoletti, Marta Kwiatkowska, and Luboš Brim. Precise parameter synthesis for stochastic biochemical systems. *Submitted to Acta Informatica*, 2015.
14. Milan Češka, David Šafránek, Sven Dražan, and Luboš Brim. Robustness analysis of stochastic biochemical systems. *PLoS ONE*, 9(4):e94553, 2014.
15. Taolue Chen, Ernst Moritz Hahn, Tingting Han, Marta Kwiatkowska, Hongyang Qu, and Lijun Zhang. Model repair for Markov decision processes. In *Theoretical Aspects of Software Engineering (TASE)*, pages 85–92. IEEE, 2013.
16. Lucia Cloth and Boudewijn R Haverkort. Model checking for survivability! In *Quantitative Evaluation of Systems (QEST)*, pages 145–154. IEEE, 2005.
17. Neil Dalchau, Niall Murphy, Rasmus Petersen, and Boyan Yordanov. Synthesizing and tuning chemical reaction networks with specified behaviours. In *DNA Computing and Molecular Programming (DNA)*, volume 9211 of *LNCS*, pages 16–33. Springer, 2015.

18. Christian Dehnert, Sebastian Junges, Nils Jansen, Florian Corzilius, Matthias Volk, Harold Bruintjes, Joost-Pieter Katoen, and Erika Abraham. Prophesy: A probabilistic parameter synthesis tool. In *Computer Aided Verification (CAV)*, volume 9206 of *LNCS*, pages 214–231. Springer, 2015.
19. Marie Duflot, Marta Kwiatkowska, Gethin Norman, and David Parker. A formal analysis of Bluetooth device discovery. *International Journal on Software Tools for Technology Transfer*, 8(6):621–632, 2006.
20. Jianbin Fang, Ana Lucia Varbanescu, and Henk Sips. A comprehensive performance comparison of CUDA and OpenCL. In *International Conference on Parallel Processing (ICCP)*. IEEE, 2011.
21. Bennett L. Fox and Peter W. Glynn. Computing Poisson probabilities. *Commun. ACM*, 31(4):440–445, 1988.
22. Daniel T. Gillespie. Exact stochastic simulation of coupled chemical reactions. *J. Phys. Chem.*, 81(25):2340–2361, dec 1977.
23. E. M. Hahn, H. Hermanns, and L. Zhang. Probabilistic reachability for parametric Markov models. *International Journal on Software Tools for Technology Transfer (STTT)*, 13(1):3–19, 2011.
24. Tingting Han, Joost-Pieter Katoen, and Alexandru Mereacre. Approximate parameter synthesis for probabilistic time-bounded reachability. In *Real-Time Systems Symposium (RTSS)*, pages 173–182. IEEE, 2008.
25. Boudewijn R Haverkort, Holger Hermanns, and Joost-Pieter Katoen. On the use of model checking techniques for dependability evaluation. In *Symposium on Reliable Distributed Systems (SRDS)*. IEEE, 2000.
26. M. Kwiatkowska, G. Norman, and D. Parker. PRISM 4.0: Verification of Probabilistic Real-time Systems. In *Computer Aided Verification (CAV)*, volume 6806 of *LNCS*, pages 585–591. Springer, 2011.
27. Marta Kwiatkowska, Gethin Norman, and David Parker. Stochastic model checking. In *Formal methods for performance evaluation*, pages 220–270. Springer, 2007.
28. C. Madsen, C.J. Myers, N. Roehner, C. Winstead, and Zhen Zhang. Utilizing stochastic model checking to analyze genetic circuits. In *Computational Intelligence in Bioinformatics and Computational Biology (CIBCB)*, pages 379–386. IEEE, 2012.
29. Gethin Norman and Vitaly Shmatikov. Analysis of probabilistic contract signing. *Journal of Computer Security*, 14(6):561–589, 2006.
30. Petr Pilař. Accelerating parameter synthesis for stochastic models. Master's thesis, Faculty of Informatics, Masaryk University, Czech Republic, 2015.
31. Ralf Steuer, Steffen Waldherr, Victor Sourjik, and Markus Kollmann. Robust signal processing in living cells. *PLoS Computational Biology*, 7(11):e1002218, 2011.
32. John E. Stone, David Gohara, and Guochun Shi. OpenCL: A parallel programming standard for heterogeneous computing systems. *Computing in Science & Engineering*, 12(3):66–73, 2010.
33. Anton J. Wijs and Dragan Bošnački. Improving GPU sparse matrix-vector multiplication for probabilistic model checking. In *Model Checking Software (SPIN)*, volume 7385 of *LNCS*, pages 98–116. Springer, 2012.
34. PRISM - Case Studies - Workstation Cluster. `http://www.prismmodelchecker.org/casestudies/cluster.php`. Accessed: 2015-09.