A PARALLEL IMPLEMENTATION OF THE FINITE STATE PROJECTION ALGORITHM FOR THE SOLUTION OF THE CHEMICAL MASTER EQUATION

HUY VO* AND BRIAN MUNSKY †

Abstract. Stochastic reaction networks are a popular modeling framework for biochemical processes that treat the molecular copy numbers within a single cell as a continuous time Markov chain, whose forward Chapman-Kolmogorov equation is known in biochemistry literature as the chemical master equation (CME). The solution of the CME contains extremely useful information that can be compared to experimental data in order to improve the quantitative understanding of biochemical reaction networks within the cell. However, this solution is costly to compute as it requires integrating an enormous system of differential equations that grows exponentially with the number of chemical species. To address this issue, we introduce a novel multiple-sinks Finite State Projection algorithm that approximates the CME with an adaptive sequence of reduced-order models with an effecient parallelization based on MPI. The implementation is tested on models of sizable state spaces using a high-performance computing node on Amazon Web Services, showing favorable scalability.

1. Introduction.

The basic building blocks of living things are single cells, in which important biochemical processes occur and are regulated. There is increasing evidence that these chemical processes can lead to diverse outcomes, even in a population of cells with the same genetic makeup [19]. An important factor contributing to this variability is intrinsic noise due to random occurences of biochemical reactions. Such noise is usually modelled using a class of continuous time Markov chain models called stochastic reaction networks (SRNs) which treat molecular copy numbers of chemical species as random integers. These SRNs have been applied in various quantitative studies of real biological systems [45, 43], and can potentially be used for guiding optimal design of new experiments [20].

Many popular methods for the numerical study of SRNs are based on drawing sample paths from the underlying Markov process. The earliest example of the trajectorial approach is Gillespie's algorithm [26], which remains the most popular simulation method for SRNs. There are also works on more effecient variants of Gillespie's algorithm [25], as well as approximate sampling approaches such as tau-leaping algorithms [27, 47, 10, 9, 3]. A major drawback of such Monte Carlo simulations is that convergence rate scales only as $O(n^{-1/2})$, where n is the number of samples. As a consequence, the ease and low cost of simulating single paths may be offset by the need to draw a large number of samples to compute reliable estimates for quantities of interest such as moments and event probabilities. Furthermore, the accuracy of these estimates can only be assessed in terms of confidence intervals. There are goal-oriented approaches that reduce the variance and cost of Monte Carlo estimates such as multilevel sampling approaches [4, 36]. On the other hand, various numerical techniques aim instead at computing summary statistics of the probability distribution function, such as the mean and variance of molecular copy numbers at specific times. When the reaction networks consist of only monomolecular reactions with mass action kinetics, the moments of the copy numbers could be captured exactly by a finite system of ordinary differential equations (ODEs). In general, however,

*Corresponding author. Department of Chemical and Biological Engineering, Colorado State University, Fort Collins, CO, USA. Email: huydvo@colostate.edu.

† Department of Chemical and Biological Engineering, Colorado State University, Fort Collins, CO, USA.

the lower-order moments depend on higher-order moments and an infinite number of moment equations is required to capture just the mean dynamics. There has been much work on moment closure techniques to approximate these inifinite systems by a finite sets of equations corresponding to low-order moments [33, 2, 35, 39, 49, 48]. While these methods can work effeciently and accurately for some SRN models, the error induced by the truncation technique is generally unknown. To circumvent this issue, there are recent works on computing bounds for the transient and stationary moments using semidefinite programming instead of closure [34, 17, 16].

This paper concerns a different approach that seeks to compute directly the time-dependent probability distribution of the process by solving the chemical master equation (CME). The finite state projection (FSP) [42] is a well-known representative of these approaches. In a nutshell, this method consists of truncating the infinite state space of the underlying Markov chain into a finite subset of states, effectively reducing the infinite-dimensional system of ODEs into a finite problem that is amenable to numerical integrators. In contrast to simulation and moment approaches, the FSP provides guarantee of accuracy with a deterministic error bound that can be easily computed from the solution of the truncated system [42]. The major drawback of the FSP is that the cost of solving the truncated system could grow prohibitive when there are many species or when the probability landscape is spread out over large number of states. Much has been done on improving algorithimc aspects of the FSP such as the selection of the state space [41, 7, 56, 8] and alternative tensor formulations [32, 53]. However, most of these algorithms were only implemented in serial, which limited the complexity of the models that could be solved with the FSP approach. There have only been very few works that discuss how to scale up the FSP on modern high performance computing platforms such as multiple-core cluster nodes [58, 52] or graphic processing units [38]. These early studies did not address the problem of exploring and updating the large state space of the CME in parallel, and the resulting implementations were only tested on truncated problems of at most a few million states.

In this paper, we introduce a novel adaptive parallel implementation of the FSP with particular focus on parallel exploration of the state space and the dynamic load-balance of the computations. We extend and parallelize a variant of the FSP described in [44], which makes use of an economical projection of the full CME state space that only expands when needed, using the proven error bound in [42] as the adaptation threshold. Most importantly, we parallelize the time-consuming task of managing the states of the FSP by distributing the storage, exploration, and migration of the states over all processors. This is a major contrast to earlier MPI-based implementations of the FSP, which either computed the state space offline in serial [58], or communicating the states in a all-to-one fashion [52]. We parallelize the numerical linear algebra computations involved in the FSP algorithm based on objects and routines from the library PETSc [5, 6], while intefacing with load-balancing approaches provided by Zoltan and ParMetis [14, 31]. Our solver can tackle CMEs with both time-invariant and time-varying propensity functions. In particular, we provide for the first time a parallel implementation of the Krylov-based Incomplete Orthogonalization Procedure [55, 24] for solving models with time-invariant propensities. We also provide interfaces into the high performance ODEs solution library SUNDIALS [30] for the solution of the general CMEs with time-varying propensities.

The paper is organized as follows. In section 2 we review the stochastic reaction networks modeling approach that gives rise to the CME, and the basic principles of the FSP algorithm. In section 3 we describe a novel adaptive FSP method and review approaches for solving the resulting reduced systems. In section 4, we discuss the parallel implementation of this adaptive FSP. Numerical experiments are discussed in section 5, where with 36 cores of an Amazon Web Services computing node we solve in a few minutes a demanding problem with 23 million states that would have taken hours to solve with a single CPU. Section 6 summarizes the paper's findings and discusses future work.

The default notational convention is as follows. For a vector $\boldsymbol{v}$, we denote its i-th entry by either the subscripted notation $[\boldsymbol{v}]_i$ or the MATLAB-like notation $\boldsymbol{v}(i)$. Similarly, an entry of a matrix $\boldsymbol{A}$ could be denoted as either $[\boldsymbol{A}]_{i,j}$ or $\boldsymbol{A}(i,j)$. We denote by $\operatorname{nnz}(\boldsymbol{v})$ and $\operatorname{nnz}(\boldsymbol{A})$ the numbers of nonzero entries in respectively $\boldsymbol{v}$ and $\boldsymbol{A}$. We use the boldface notation $\boldsymbol{p}$ to denote the probability distribution obtained from solving the CME. The notation n_{proc} is reserved for the number of processors. For a set S, we denote by $|S|$ the number of elements in S.

2. Background.

2.1. Stochastic reaction network models of gene expression.

Consider a chemical reaction network with N molecular species with M reaction channels. The state of the chemical network is the vector $\boldsymbol{x} \in \mathbb{N}^N$ of copy numbers of different molecular species. Each reaction event, such as transcription of a new mRNA molecule, incurs a change in the state that is captured by a stoichiometric vector $\boldsymbol{\nu}_k \in \mathbb{N}^N$. Given the current state $\boldsymbol{x}$, the process can transit only to states of the form $\boldsymbol{x} + \boldsymbol{\nu}_k,\ k = 1, \ldots, M$. The time-dependent state vector is modeled as a continuous-time Markov chain $\{\boldsymbol{X}(t)\}_{t\geq 0}$ with transition rates given by the propensity functions $a_k(t, \boldsymbol{x})$. From a current state $\boldsymbol{x}$, the chain can make jumps to M neighboring states $\boldsymbol{x} + \boldsymbol{\nu}_k$ with the infinitesimal probability $a_k(t, \boldsymbol{x})$.

The chemical master equation (CME) describes the dynamic of the probability distribution of the state $\boldsymbol{X}(t)$. In particular, given the initial state $\boldsymbol{X}(0)$ with known distribution, let $p(t, \boldsymbol{x})$ be the probability of the event $[\boldsymbol{X}(t) = \boldsymbol{x}|\boldsymbol{X}_0]$, then the CME has the form

$$\frac{d}{dt}p(t,\boldsymbol{x}) = \sum_{k=1}^{M} a_k(t, \boldsymbol{x} - \boldsymbol{\nu}_k)p(t, \boldsymbol{x} - \boldsymbol{\nu}_k) - a_k(t, \boldsymbol{x})p(t, \boldsymbol{x}),\ \boldsymbol{x} \in \mathcal{S}, \tag{2.1}$$

where $\mathcal{S}$ is the set of all reachable states.

We restrict our attention to models where the propensity functions are separable in the form

$$a_k(t, \boldsymbol{x}) = c_k(t)\alpha_k(\boldsymbol{x}). \tag{2.2}$$

This assumption is valid for any systems with time-invariant propensities, and for systems with time-varying propensities that follow mass-action kinetics.

Since the state space $\mathcal{S}$ is discrete, we can treat the probability distribution of $\boldsymbol{X}(t)$ as a probability vector $\boldsymbol{p}(t)$. Define the transition rate matrix $\boldsymbol{A}(t)$ with entries enumerated by the states as

$$[\boldsymbol{A}]_{i,j} = \begin{cases} a_r(t, \boldsymbol{x}_j),\ \boldsymbol{x}_i = \boldsymbol{x}_j + \boldsymbol{\nu}_r, \\ -\sum_{r=1}^{M} a_r(t, \boldsymbol{x}_j),\ i = j, \\ 0 \text{ otherwise} \end{cases}. \tag{2.3}$$

Equation (2.1) could then be equivalently formulated as an initial value problem that takes the form

$$\frac{d}{dt}\boldsymbol{p}(t) = \boldsymbol{A}(t)\boldsymbol{p}(t),\ \boldsymbol{p}(0) = \boldsymbol{p}_0. \tag{2.4}$$

Next, we discuss how the CME could be solved numerically using the finite state projection.

2.2. The Finite State Projection algorithm. The finite state projection (FSP) [42] is a systematic approach to approximate the solution of the CME using a vector with finite support. In its simplest formulation, the FSP consists of the truncation of the infinite state space $\mathcal{S}$ into a finite set $\Omega = \left\{\boldsymbol{x}_{i_1}, \ldots, \boldsymbol{x}_{i_{|J|}}\right\}$. Let $J = \left\{i_1, \ldots, i_{|J|}\right\}$ be the corresponding set of indexes. For simplicity and without loss of generality, let us assume that $J = \{1, 2, \ldots, |J|\}$.

The FSP first solves a finite system of the form

$$\frac{d}{dt}\boldsymbol{p}_\Omega(t) = \boldsymbol{A}_\Omega(t)\boldsymbol{p}_\Omega(t), \tag{2.5}$$

where

$$\boldsymbol{A}_\Omega = \boldsymbol{A}(J, J), \tag{2.6}$$

and

$$\boldsymbol{p}_\Omega(0) = \boldsymbol{p}_0(J). \tag{2.7}$$

The solution of the finite system (2.5) provides the approximation to the true CME solution on Ω, that is,

$$\boldsymbol{p}(t) \approx \tilde{\boldsymbol{p}}(t) \equiv \begin{bmatrix} \boldsymbol{p}_\Omega(t) \\ \boldsymbol{0} \end{bmatrix}.$$

The FSP could be viewed as an aggregation method that turns all transitions to outside of Ω into an absorbing state (Fig. 1). The probability of this sink state provides the exact approximation error [42, Theorem 2.2] in terms of the one-norm,

$$\left\|\boldsymbol{p}(t) - \tilde{\boldsymbol{p}}_\Omega(t)\right\|_1 = 1 - \mathbb{1}^T\boldsymbol{p}_\Omega(t). \tag{2.8}$$

For all models encountered in practice, the error bound $g(t)$ decreases monotonically as Ω expands [42]. Under some regularity conditions on the propensity functions, it could be shown that $\left\|\boldsymbol{p}(t) - \tilde{\boldsymbol{p}}_\Omega(t)\right\|_1 \to 0$ as $\Omega \to \mathcal{S}$ [22].

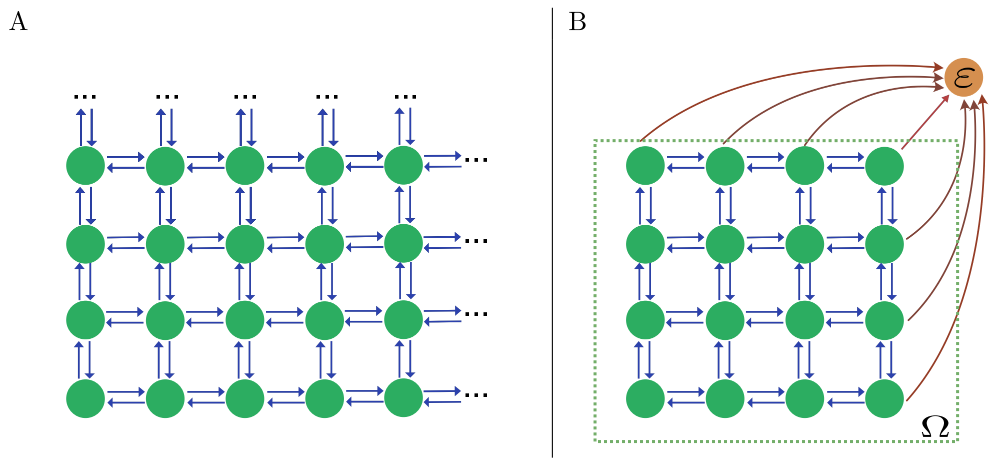

FIG. 1. *Schematic of the Finite State Projection algorithm. (A): The original state space of the CME, which could have infinitely many states. (B): The truncated state space, where the states outside a finite subset (Ω) are lumped into a single absorbing state (ε). The probability of ε is found simply by subtracting to one the sum of probabilities of the remaining states (eq.(2.8)).*

3. Numerical schemes.

3.1. Adaptive Finite State Projection with multiple sink states. We use finite state sets Ω that consist of reachable states that satisfy a set of constraints [44]

$$\mathcal{C} : f_j(\boldsymbol{x}) \leq b_j, j = 1, \ldots, n_{\text{constr}}. \tag{3.1}$$

where b_j are the upper bounds that are changed in an adaptive way as the FSP integration progresses. Let $\mathcal{V}(\boldsymbol{x}; \mathcal{C}, \boldsymbol{b})$ denote the set of constraints determined by $(\mathcal{C}, \boldsymbol{b})$ that is violated by a state $\boldsymbol{x} \in \Omega$, and let $\mathcal{A}_j(\boldsymbol{b})$ denote the set of states that satisfy the j-th constraint. Furthermore, let $\mathcal{R}(\boldsymbol{x}_0)$ denote the set of states that can be reached from the initial state $\boldsymbol{x}_0$. The finite state set for the FSP is thus

$$\Omega = \cap_{j=1}^{n_{\text{constr}}} \mathcal{A}_j(\boldsymbol{b}) \cap \mathcal{R}(\boldsymbol{x}_0).$$

We then approximate the Markov process of the CME with a finite Markov process with the augmented state space $\Omega \cup \{G_1, \ldots, G_{n_{\text{constr}}}\}$, where each G_ℓ ($\ell = 1, \ldots, n_{\text{constr}}$) is a sink state that absorbs transitions from Ω into states that violate the ℓ-th constraint. Fig. 2 presents an example of the truncated state space based on a set of three constraints.

Assuming the augmented state space is enumerated so that the sink state G_ℓ has index $|\Omega| + \ell$, the time-varying transition rate matrix $\bar{\boldsymbol{A}}_\Omega(t)$ of this Markov process could be decomposed into

$$\bar{\boldsymbol{A}}_{\Omega,\mathcal{C}}(t) = \begin{bmatrix} \boldsymbol{A}_\Omega & \boldsymbol{0} \\ \boldsymbol{C}_\Omega & \boldsymbol{0} \end{bmatrix} \tag{3.2}$$

where $\boldsymbol{A}_\Omega$ captures the transitions between states in Ω and $\boldsymbol{C}_\Omega \in \mathbb{R}^{n_{\text{constr}} \times |\Omega|}$ captures the transitions into the absorbing states. Here, if a transition $\boldsymbol{x} \to \boldsymbol{x} + \boldsymbol{\nu}_r$ violates more than one constraints then we divide the rate $a_r(t, \boldsymbol{x})$ equally into the corresponding absorbing states. The nonzero entries of $\boldsymbol{A}_\Omega$ and $\boldsymbol{C}_\Omega$ are given by

$$[\boldsymbol{A}_\Omega(t)]_{i,j} = \begin{cases} a_r(t, \boldsymbol{x}_j), & \text{for } i, j < |\Omega_k| \text{ and } \boldsymbol{x}_i = \boldsymbol{x}_j + \boldsymbol{\nu}_r, \\ -\sum_{r=1}^{M} a_r(t, \boldsymbol{x}_j), & \text{for } i = j < |\Omega_k|, \\ 0 \text{ otherwise} \end{cases}, \tag{3.3}$$

and

$$[\boldsymbol{C}_\Omega(t)]_{\ell,j} = \sum_{r=1}^{M} a_r(\boldsymbol{x}_j) \cdot \frac{\chi_{\mathcal{A}_\ell(\boldsymbol{b})}(\boldsymbol{x}_j + \boldsymbol{\nu}_r)}{|\mathcal{V}(\boldsymbol{x}_j + \boldsymbol{\nu}_r)|} \quad \text{for } \ell = 1, \ldots, n_{\text{constr}}, j = 1, \ldots, |\Omega|. \tag{3.4}$$

The FSP approximation becomes

$$\frac{d}{dt}\bar{\boldsymbol{p}}_\Omega(t) = \bar{\boldsymbol{A}}_\Omega(t)\bar{\boldsymbol{p}}_\Omega(t), \ \bar{\boldsymbol{p}}_\Omega(0) = [\boldsymbol{p}_0(\Omega), 0, \ldots, 0]^T, \tag{3.5}$$

where the probability vector $\bar{p}_\Omega$ is partitioned into

$$\bar{\boldsymbol{p}}_\Omega(t) = \begin{bmatrix} \boldsymbol{p}_\Omega(t) \\ g_1(t) \\ \vdots \\ g_{n_{\text{constr}}}(t) \end{bmatrix} \in \mathbb{R}^{|\Omega| + n_{\text{constr}}}, \tag{3.6}$$

with the last n_{constr} entries of $\bar{\boldsymbol{p}}_\Omega$ tracking the probabilities of the absorbing states.

The advantage of the multi-sink formulaion (3.6) to the original one-sink FSP is that we know which sink state attracts the most probability and we can selectively relax only their corresponding constraints for the next iteration. As a consequent, the finite state set Ω only needs to expand on the directions where the probability mass leaks the most, allowing for a more flexible and economical truncation.

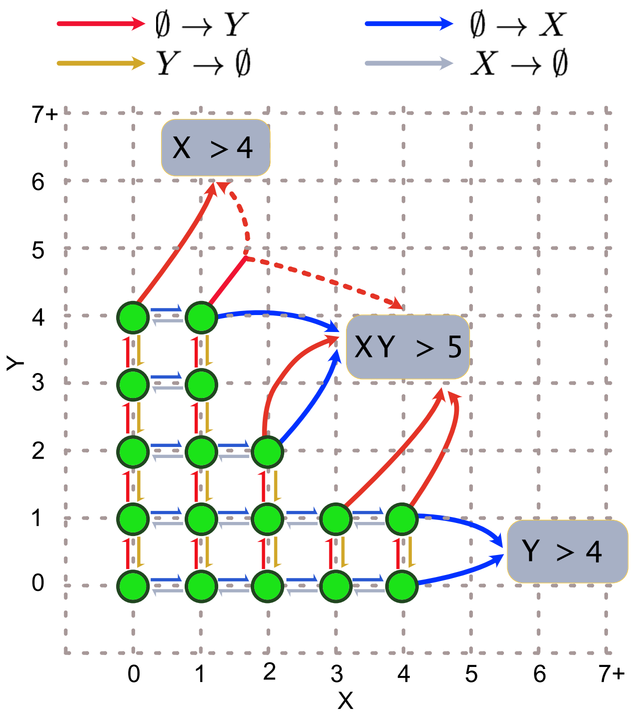

FIG. 2. *Example of the finite state projection with multiple sink states. We consider a simple two-dimensional model with four reactions $\emptyset \to X$, $X \to \emptyset$, $\emptyset \to Y$, $Y \to \emptyset$. The sink states correspond to three constraints: (ε_0): $[X] \leq 4$, (ε_1): $[Y] \leq 4$, (ε_2): $[X][Y] \leq 5$. The arrows represent transitions from each state. Transitions leading to states that violate the j-th constraint will be directed to ε_j. Note that a reaction may bring one state into violating different constraints simultaneously (state $(1, 4)$ in the graphics), in which case the transition rate is divided equally among the receiving absorbing states.*

Taking this one step further, we divide the whole time interval $[0, t_f]$ into subintervals with timesteps $0 := t_0 < t_1 < \ldots < t_K := t_f$, and devise a multiple-time-interval formulation [7, 56]. At each subinterval $[t_k, t_{k+1})$ we solve a finite system of the form

$$\frac{d}{dt}\bar{\boldsymbol{p}}_{\Omega_k}(t) = \bar{\boldsymbol{A}}_{\Omega_k}(t)\bar{\boldsymbol{p}}_{\Omega_k}(t)\; t \in [t_k, t_{k+1}) \tag{3.7}$$

Each solution vector $\bar{\boldsymbol{p}}_{\Omega_k}$ has the partitioning (3.6), with the absorbing state probabilities $g_j(t), j = 1, \ldots, n_{\text{constr}}$ tracking the accumulated probability mass that are loss by various FSP approximations. Let $g(t) = \sum_{j=1}^{n_{\text{constr}}} g_j(t)$, then $g(t)$ is an upper bound on the accumulated approximation error. We require that the maximum error acummulated at the absorbing state satisfies

$$\max_{j=1,\ldots,n_{\text{constr}}} g_j(t) \leq (n_{\text{constr}})^{-1}\varepsilon_{FSP}\frac{t}{t_f}. \tag{3.8}$$

Given the current time step t_k, we let the ODE integrator advance the solution until either reaching final time t_f or an itermediate timepoint t_{k+1} where the inequality(3.8) starts to change direction.

If the integration halts at an intermediate timepoint due to the error control (3.8), we enlarge the FSP constraint bounds $b_j \leftarrow (1+\delta)b_j,\ j = 1, \dots, n_{\text{constr}}$, giving rise to a new set of relaxed constraints $\mathcal{C}_{k+1}$. We then expand from Ω_k into states that satisfy these more relaxed constraints, giving rise to the expanded set Ω_{k+1} that constitutes the FSP approximation of the next time interval. The recommended scaling factor in our code is $\delta := 0.2$, though this could be changed by the user.

The pseudocode in Algorithm 3.1 describes the prototype of the adaptive Finite State Projection algorithm we just discussed. From there, we see that an implementation of the algorithm breaks down to three major groups of tasks: the dynamic management of the finite state subset Ω_k, the generation of the time-dependent matrix $\bar{\boldsymbol{A}}_{\Omega_k}(t)$ and the matrix-vector multplications at each time step, and the advancement of the resulting finite system of ODEs.

Algorithm 3.1 Multiple-sink-state Finite State Projection

Input: Initial vector of FSP bounds $\boldsymbol{b}^0$; Stoichiometry matrix $\boldsymbol{S}$; FSP tolerance ε_{FSP}; Propensity functions stored as time-dependent factors $c_i(t), i = 1, \dots, M$ and state-dependent factors $\alpha_i(\boldsymbol{x}), i = 1, \dots, M$. Initial state $\boldsymbol{x}_0$.

```
 1: $t \leftarrow 0$, $\boldsymbol{b} \leftarrow \boldsymbol{b}^0$;
 2: Generate an initial projection space $\Omega_0$ given by
        $$\Omega_0 = \{\boldsymbol{x}_0\} \cup \left\{\boldsymbol{x} \in \Omega \mid \boldsymbol{f}(\boldsymbol{x}) \le \boldsymbol{b}^0\right\}$$
 3: Set initial condition for the FSP approximation $\boldsymbol{p}_0$.
 4: $k \leftarrow 0$.
 5: while $t < t_f$ do
 6:     Advance the finite ODEs system (3.5) up to the time $t_{tmp}$ where either final
        time is reached or the FSP error condtion (3.8) fails.
 7:     if $t_{tmp} < t_f$ then
 8:         for $j = 1, \dots, n_{\text{constr}}$ do
 9:             if $g_j(t_{tmp}) \ge \frac{1}{n_{\text{constr}}} \frac{t_{tmp}}{t_f} \varepsilon_{FSP}$ then
10:                 $b_j^{(k+1)} := (1+\delta) b_j^{(k)}$
11:             else
12:                 $b_j^{(k+1)} := b_j^{(k)}$.
13:             end if
14:         end for
15:         $\Omega_{k+1} := \text{Expand}(\Omega_k, \mathcal{C}_{k+1})$.
16:         Generate the time-dependent matrix $\bar{\boldsymbol{A}}_{\Omega_{k+1}}(t)$ based on $\Omega_{k+1}$ and $\mathcal{C}_{k+1}$.
17:         Generate $\bar{\boldsymbol{p}}_{k+1}$ and scatter entries of $\bar{\boldsymbol{p}}_k$ into $\bar{\boldsymbol{p}}_{k+1}$.
18:     end if
19:     $k := k + 1$.
20:     $t_k := t_{tmp}$.
21: end while
```

Output: The subset Ω_k and the corresponding approximation $\boldsymbol{p}_k(t_f)$ of the CME solution at final time t_f.

3.2. Adaptive Krylov integrator with event detection for the time-invariant propensities. When all propensities are time-invariant, solving the finite systems (3.7) amounts to computing the action of the matrix exponential,

$$\bar{\boldsymbol{p}}_{\Omega_k}(t_{k+1}) = \exp((t_{k+1} - t_k)\bar{\boldsymbol{A}}_{\Omega_k})\bar{\boldsymbol{p}}_{\Omega_k}(t), . \tag{3.9}$$

A popular approach for evaluating these expressions that are particularly well-suited for large-sparse matrices arising from the FSP are Krylov subspace techniques [50], which have been incorporated in many serial CME solvers [7, 51, 8].

To simplify notations, let $\boldsymbol{v} := \bar{\boldsymbol{p}}_{\Omega_k}$ and $\boldsymbol{B} := \bar{\boldsymbol{A}}_{\Omega_k}$. The Krylov-based approach starts by generating a pair of matrices $\boldsymbol{V}_{m+1} \in \mathbb{R}^{|\Omega_k|\times(m+1)}$, $\boldsymbol{H}_m \in \mathbb{R}^{m\times m}$, such that the columns of $\boldsymbol{V}_{m+1} := [\boldsymbol{v}_1\ \boldsymbol{v}_2 \ldots\ \boldsymbol{v}_{m+1}]$ form a basis for the Krylov subspace

$$\mathcal{K}_{m+1}(\boldsymbol{B}, \boldsymbol{v}) := \operatorname{span}\left\{\boldsymbol{v}, \boldsymbol{B}\boldsymbol{v}, \ldots, \boldsymbol{B}^m\boldsymbol{v}\right\}, \tag{3.10}$$

and that the following relation is satisfied,

$$\boldsymbol{B}\boldsymbol{v} = \boldsymbol{V}_m\boldsymbol{H}_m + h_{m+1,m}\boldsymbol{v}_{m+1}\boldsymbol{e}_m^T, \tag{3.11}$$

where $\boldsymbol{V}_m := \boldsymbol{V}(:, 1 : m)$, $h_{m+1,m} := \|\boldsymbol{B}\boldsymbol{v}_m\|_2$, and $\boldsymbol{e}_m = [0 \ldots 01]^T \in \boldsymbol{R}^m$. Supposing that these conditions are met, the pair $(\boldsymbol{V}_m, \boldsymbol{H}_m)$ can then be used to form an approximation for the matrix exponential operator as

$$\exp(\tau\boldsymbol{B})\boldsymbol{v} \approx \boldsymbol{V}_m \exp\left(\tau\boldsymbol{H}_m\right)(\|\boldsymbol{v}\|_2\boldsymbol{e}_1), \tag{3.12}$$

where $\boldsymbol{e}_1 = [1\ 0 \ldots 0]^T \in \mathbb{R}^m$. The exponential of the small dense matrix $\tau\boldsymbol{H}_m$ in the above expression is usually computed using the diagonal Padé method [50].

There are many ways to generate the matrices $\boldsymbol{V}_{m+1}$ and $\boldsymbol{H}_m$ that satisfy the two conditions described above. The well-known implementation in the package Expokit [50], for example, uses the full orthogonalization method (FOM), which ensures that $\boldsymbol{V}_{m+1}$ is an orthogonal matrix, making the approximation (3.12) equivalent to an orthogonal projection onto the corresponding Krylov subspace. The FOM, however, requires modified Gram-Schmidt sweeps whose cost grows quadratically with the Krylov dimension m. This becomes prohibitive for large input vectors $\boldsymbol{v}$. There are recent works that propose the alternative use of Incomplete Orthogonalization Procedure (IOP) [55, 24] that trade the cost of full orthogonalization for a slight increase in matrix-vector multiplications and larger Krylov dimension. This is the approach that we implement in this paper.

In contrast to the FOM, the IOP requires only that each vector $\boldsymbol{v}_j$ be orthogonal to the preceding q vectors $\boldsymbol{v}_{j-q}, \ldots, \boldsymbol{v}_{j-1}$, where $q \geq 1$ is a user-prescribed orthogonalization length (the case $q = m$ reduces to the classic Arnoldi procedure). When implemented using an adaptive strategy based on the *phipm* algorithm of Niesen and Wright [46], the IOP with $q := 2$ outperform the traditional Arnoldi procedure on several numerical benchmark problems drawn from computer and biochemical systems [55] and shallow water equations [23]. The details of this strategy could be found in [55]. We discuss here only the aspects that are relevant to the parallel implementation of the IOP, and also note that a parallel implementation for the FOM with variable Krylov subspace dimension has been previously proposed [37].

Similar to the serial implementations [23, 55], we break down the matrix exponential (3.9) into a step-by-step integration scheme that advances through timesteps $0 := t_1 < t_2 < t_J := T$. The intermediate solutions are given by the Krylov approximations

$$\boldsymbol{w}_{j+1} := \boldsymbol{V}_{m_j}^{(j)} \exp\left(\tau_j\boldsymbol{H}_{m_j}^{(j)}\right)(\|\boldsymbol{w}_j\|_2\boldsymbol{e}_1), \quad \boldsymbol{w}_0 := \boldsymbol{v}, \tag{3.13}$$

where, similar to (3.12), $\boldsymbol{V}_{m_j}^{(j)}$ and $\boldsymbol{H}_{m_j}^{(j)}$ are generated from the IOP applied to the matrix $\boldsymbol{B}$ and the vector $\boldsymbol{w}_j$. Note that the stepsize $\tau_j := t_{j+1} - t_j$ and the Krylov subspace dimension m_k are not fixed, but determined adaptively during the integration. After a successful step at time t_j, we either keep the Krylov dimension fixed while using a new stepsize τ^{new} for the next step, or keep the current stepsize but use a new value m^{new} for the Krylov basis size. These stepsize and basis size values are determined from a criteria that ensures that a posteriori error estimate of the local truncation error remains below a tolerance (see [55] for details of error estimation and control). The decision to vary which quantity (τ_j or m_j) is based on a guess of the future integration cost associated with each choice.

We estimate the local cost per CPU for advancing with a stepsize τ and basis size m is given by [46, 23, 55]

$$C_{loc}(\tau, m, q) := \lceil \frac{T - t_j}{\tau} \rceil \left(m C^{\text{loc}}_{\text{mat-vec}} + C^{\text{loc}}_{\text{orth}}(m, q) + C^{\text{loc}}_{\text{expm}}(m) \right), \tag{3.14}$$

where $C^{\text{loc}}_{\text{mat-vec}}$, $C^{\text{loc}}_{\text{orth}}(m, q)$, $C^{\text{loc}}_{\text{expm}}(m)$ are the on-processor costs, measured in terms of the number of FLOPs, for matrix-vector multplications with $\boldsymbol{B}$, orthogonalization cost for basis size m and IOP parameter q, and the evaluation of the small matrix exponential of size $m \times m$ with Pade technique.

These costs are estimated in the same way as in [53], but with the global sizes of the matrix and vectors involved replaced by their local dimensions. Once these local costs are computed for each pair (τ_j, m^{new}) and (τ^{new}, m_j), we make a call to `MPI_Reduce` to get the maximum cost $C_{glob}(\tau, m, q)$ over all processors. We then choose to change dimension if $C_{glob}(\tau^{new}, m, q) > C_{glob}(\tau, m^{new}, q)$ and to change the stepsize otherwise. We choose to use the maximum over local costs, rather than the sum as in a previous parallel adaptive Krylov implemention [37], since parallel computational time depends more on the processor with the heaviest workload than on the total workloads over all processors.

4. Parallel implementation.

4.1. Parallel management and dymanic expansion of the state space. The truncated state set Ω is partitioned into $\Omega^{(j)} \subset \Omega$, each separately owned by a processor. A state $\boldsymbol{x} \in \Omega^{(j_{\boldsymbol{x}})}$ held by processor $j_{\boldsymbol{x}}$ has a local index $i_{\text{loc}}(\boldsymbol{x}) \in \{1, \ldots, |\Omega^{(j)}|\}$. To manage and retrieve information about the states in a scalable way, we employ the Distributed Directory (DD) in the Zoltan library [14]. The DD object is essentially an MPI-based hash table that take multi-dimensional vectors of integers as keys. The data entry for each key $\boldsymbol{x} \in \mathcal{S}$ consists of a pair $(j_{\boldsymbol{x}}, i_{\text{loc}}(\boldsymbol{x}))$ that stores the rank of the owning processor and the local index of $\boldsymbol{x}$. Each call to the lookup and insertion routines must be done collectively by all processors.

Let Ω_k be the state set at timestep k of the Finite State Projection. In order to expand Ω_k into a new set Ω_{k+1} determined by the constraints $\mathcal{C}_{k+1}$, we use an iterative breadth-first search algorithm. Let $\partial\Omega$ denote the set of states on the "boundary" of $\Omega \subset \mathcal{S}$, that is,

$$\partial\Omega = \{\boldsymbol{x} \in \mathcal{S} | \boldsymbol{x} + \boldsymbol{\nu}_r \in \mathcal{S} \setminus \Omega \text{ for some } r = 1, \ldots, M\}. \tag{4.1}$$

The serial state space expansion scheme starts at the current state set $\Omega_{now} := \Omega_k$. We then compute the set $\bigcup_{r=1}^{M} (\partial\Omega_{now} + \boldsymbol{\nu}_r)$ and thin it down to a subset B that consists only states satisfying $\mathcal{C}_{k+1}$. We then annex B to Ω_{now}, and the process repeats until all reachable states that satisfy $\mathcal{C}_{k+1}$ have been explored.

Our parallelization of this scheme is described in algorithm 4.1. On line 3 of the pseudocode, we partition $\partial\Omega_{now}$ into roughly equal parts that are distributed to individual processors. This helps improve the load-balance of the exploration step on line 5-7. The distribution step employs a fast and simple partitioning scheme (called `BLOCK` in Zoltan) that distributes roughly equal numbers of states to the processors. Then, the exploration of new states on line 5-7 is done concurrently on all processors without the need of inter-processor communication. We then annex the states discovered thus far to the global state set, while making sure to not add redundant states by checking with the Distributed Directory.

Algorithm 4.1 Parallel expansion of the finite state subset.

Input: Current state set Ω_k.

Output: New state set Ω_{k+1} consisting of states reachable from Ω_k that satisfies the new constraints $\mathcal{C}_{k+1}$.

```
1: $\Omega_{\text{now}} := \Omega_{\text{old}}$.
2: repeat
3:     Distribute $\partial\Omega_{\text{now}}$ to all processors into partitions $A^{(j)}$ such that $\partial\Omega_{\text{now}} = \cup_{j=1}^{n_{\text{proc}}} A_j$ and $|A_j| \approx |A_k|$ for $j, k = 1, \ldots, n_{\text{proc}}$.
4:     for all processors $j = 1, \ldots, n_{\text{proc}}$ in parallel do
5:         $B^{(j)} := \bigcup_{r=1}^{M} \{\boldsymbol{x} \in A^{(j)} + \boldsymbol{\nu}_r | \boldsymbol{x}$ satisfies all constraints $\mathcal{C}_{k+1}\}$.
6:     end for
7:     Parallel union $\Omega_{now} := \Omega_{now} \bigcup_{j=1}^{n_{\text{proc}}} B^{(j)}$.
8: until $B^{(j)} = \emptyset$ for all $j = 1, \ldots, n_{\text{proc}}$.
9: $\Omega_{k+1} := \Omega_{\text{now}}$.
```

4.2. Parallel matrix object. Let $\bar{\boldsymbol{A}}_{\Omega,\mathcal{C}}$ be the time-dependent transition rate matrix associated with a state set Ω and the constraint set $\mathcal{C}$. Recall that we can partition $\bar{\boldsymbol{A}}_{\Omega,\mathcal{C}}$ into $\boldsymbol{A}_\Omega$ and $\boldsymbol{C}_{\Omega,\mathcal{C}}$, defined respectively in eq. (3.3) and eq. (3.4). Recall separability assumption (2.2) on the propensities, and let $\mathcal{V}$ denote the set of reactions r for which $c_r(t)$ varies with the time variable t. These matrices have the decompositions

$$\boldsymbol{A}_\Omega(t) = \sum_{r\in\mathcal{V}} c_r(t)\boldsymbol{A}_{r,\Omega} + \boldsymbol{A}_{0,\Omega}, \tag{4.2}$$

$$\boldsymbol{C}_{\Omega,\mathcal{C}}(t) = \sum_{r\in\mathcal{V}} c_r(t)\boldsymbol{C}_{r,\Omega,\mathcal{C}} + \boldsymbol{C}_{0,\Omega}, \tag{4.3}$$

where $\boldsymbol{A}_{0,\Omega} := \sum_{r\notin\mathcal{V}} c_r\boldsymbol{A}_{r,\Omega}$ and $\boldsymbol{C}_{0,\Omega} := \sum_{r\notin\mathcal{V}} c_r\boldsymbol{C}_{r,\Omega}$ captures the time-invariant part of the matrices $\boldsymbol{A}_\Omega$ and $\boldsymbol{C}_{\Omega,\mathcal{C}}$.

As illustrated in fig. 3, each $\boldsymbol{A}_r$ is distributed row-wise into the processors as done in PETSc's parallel `Mat` object [5], while the (very thin) $\boldsymbol{C}_r$ matrices are distributed column-wise. The column-wise partitioning of $\boldsymbol{C}_r$ means that no off-processor elements of the input vector is needed for the parallel matrix-vector multiplication with $\boldsymbol{C}_{\Omega,\mathcal{C}}$, while the update of the global output vector requires only $O(n_{\text{constr}})$ messages. PETSc offers a handful of different storage schemes for the on-processor portions of the sparse matrix. We found in our tests that storing the FSP matrix with the `MATMPISELL` format [57] yields better performance than the default compressed sparse row (CSR) format. This is also the matrix format that will be used in the numerical tests below.

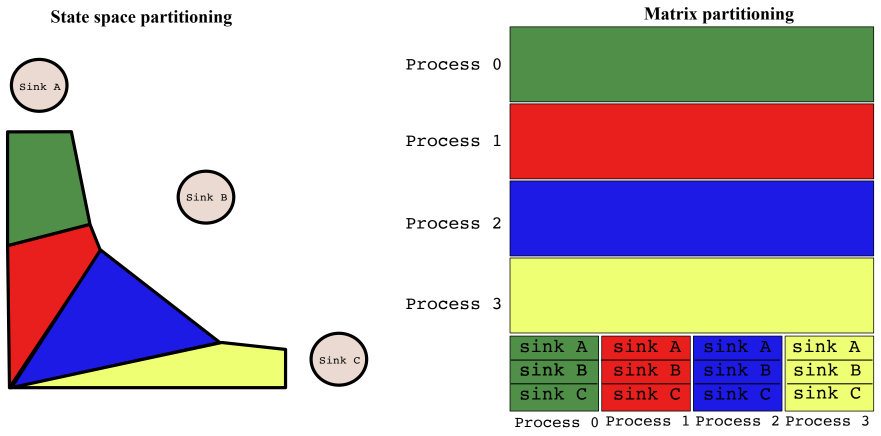

FIG. 3. *Relationship between state partitioning and the partitioning of the FSP-reduced transit rate matrix $\bar{\boldsymbol{A}}_{\Omega,\mathcal{C}}(t)$ (eq. (3.5) in the main text). The submatrix corresponding to transitions between states is distributed row-wise to the processes, while the submatrix corresponding to transitions into sink states is distributed column-wise.*

4.3. Dynamic load-balancing approaches. Since the adaptive FSP adds states to the projection space after every time step, it is important to redistribute the states and their associated data after expansion, so that the subsequent computational work loads are evenly distributed among the processors, while keeping inter-processor communication minimal. In addition, the redistriution scheme must also take into account the cost of data migration. While the true computation and communication cost is generally hard to quantify, several approximations and heuristics based on graphs and hypergraphs have been shown to yield good performance in practice [11, 31, 12]. We specifically interface our state space routines with the Graph-based adaptive repartitioning algorithm of ParMETIS [31] and the hypergraph-based PHG algorithm of Zoltan [14, 12], and also to a simple strategy that only seeks to distribute CME states equally among processors.

4.4. Code availability and Python interface. All the numerical techniques and algorithms discussed have been implemented as a C++ library [1] that can also be used in Python using a Cython-based wrapper [2]. The features of the C++ programming language allow us to write our codes in an object-oriented way, with an eye on reusability and extensibility. Specifically, the basic building blocks of the presently discussed FSP variant (as well as future developments) are the three object classes `StateSetBase`, `FspMatrixBase`, and `OdeSolverBase`. These objects correspond to the basic data structures and methods that are common in all FSP methods. In particular:

1. The `StateSetBase` class manages the subset of CME states. It provides basic methods for state lookup and insertion, as well as parallel state partitioning and redistribution (via interface to Zoltan [14]). It has a virtual method `Expand` to be specified by more specialized classes to implement the specific state space expansion technique. In particular, we have a child class `StateSetConstrained` that implement the nonlinear constraints-based state expansion technique mentioned in section 4.1. Other methods for expanding

[1]C++ library available at: https://github.com/voduchuy/pacmensl
[2]Python wraper available at: https://github.com/voduchuy/pypacmensl

the state space [51, 56] can also be similarly developed by specifying only the `Expand` method.

2. The `FspMatrixBase` class manages the time-dependent truncated transition rate matrix of the CME. This class provides the basic method to generate the state transition rate matrix resulted from FSP truncation as defined in eq. (3.3) (but not the multi-sink components (3.4), which we leave to a child class named `FspMatrixConstrained`, which works with `StateSetConstrained` and which generates the additional absorbing state components needed for the multi-sink FSP solver).
3. The `OdeSolverBase` class provides basic methods to solve the truncated linear ODE systems (3.7). The main class data is the pointer to an instance of `FspMatrixBase`, and a pointer to a function that perform the check for the FSP criteria (2.8). This means that the ODE solution can halt midway if it finds the FSP error to exceed the prescribed tolerance, and return to the FSP solver to request for an expanded state space. The specific ODE solver used is provided in child classes that provide interfaces to PETSc's TS module [1] and SUNDIALS [30], as well as implementing the Krylov-based IOP technique discussed in section 4.

The multi-sink FSP approach we introduce in this paper is implemented in a class called `FspSolverMultiSinks`, which is based on the composition of the objects discussed above. We note that parallel implementations of other FSP-based algorithms [7, 51, 56, 8, 29] can be similarly benefited from reusing the base classes we mentioned above.

5. Results.

In this section, we present numerical tests of our parallel FSP implementation on three different stochastic chemical kinetics models. The aim of these tests is to demonstrate how different algorithmic choices affect the runtime and scalability of the solver on specific problems. All tests are run on a `c5.18x` instance of Amazon Web Serivces. This node consists of two sockets, each of which houses 18 Intel Xeon Platinum 8124M CPU cores that operate at 3.00GHz per core. We use PETSc 3.11.3 and Zoltan library that were compiled against OpenMPI 4.0 with the GNU C/C++ compilers. In addition to default build settings, the PETSc library is compiled with the option to enable AVX-512 kernels [57], which Intel Xeon processors support. The execution of the parallel programs are done via a call to `mpirun` with the options `--bind-to-core` and `--map-by-socket` which binds each process to a physical core and which spreads the processes as evenly as possible across the NUMA nodes. Our codes are written in an object-oriented style in C++ and are compiled using OpenMPI compiler wrapper of GNU.

5.1. Repressilator.

We first consider a three-species model inspired by the well-known repressilator gene circuit [18]. This model consists of three species, TetR, λcI and LacI, which constitute a negative feedback network (Table 1). We solve the problem up to time $t = 10$ (abitrary time unit) using the FSP tolerance of 10^{-4}, starting from a point mass measure concentrated at $\boldsymbol{x}_0 = (\mathrm{TetR}, \lambda\mathrm{cI}, \mathrm{LacI}) = (20, 0, 0)$. We let the state space grow adaptively and use the Adaptive Repartitioning algorithm of ParMetis [31] to dynamically re-balance the state space across processors.

	reaction	propensity	parameters (arbitrary unit)
1.	$\emptyset \to \text{TetR}$	$k/(1 + a[\text{LacI}]^b)$	$k = 100,\ a = 20,\ b = 6$
2.	$\text{TetR} \to \emptyset$	$\gamma[\text{TetR}]$	$\gamma = 1$
3.	$\emptyset \to \lambda\text{cI}$	$k/(1 + a[\text{TetR}]^b)$	
4.	$\lambda\text{cI} \to \emptyset$	$\gamma[\lambda\text{cI}]$	
5.	$\emptyset \to \text{LacI}$	$k/(1 + a[\lambda\text{cI}]^b)$	
6.	$\text{LacI} \to \emptyset$	$\gamma[\text{LacI}]$	

TABLE 1
Reactions and propensities in the repressilator model. ([X] is the number of copies of the species X.)

We compare the performance of four algorithmic variants based on the choice of the FSP shape and on whether the FSP grows dynamically or is fixed at the beginning of the integration. For the choice of FSP shape, we can define the FSP state space is the customary way of choosing a hyper-rectangle defined by the constraints $x_i \leq b_i, i = 1, 2, 3$. However, the structure of the repressilator network suggests that we can exploit the negative correlation between species to truncate more states. In particular, we can add more nonlinear constraints of the form $x_i x_{i+1} \leq b_{i,i+1}, x_3 x_1 \leq b_{3,1} (i = 1, 2)$ which stems from the intuition that more copies of a repressing species lead to fewer copies of the repressed species. This adds up to a total of six constraints as detailed in Table 2.

	constraint function	initial bound
1.	$[\text{TetR}]$	22
2.	$[\lambda\text{cI}]$	2
3.	$[\text{LacI}]$	2
4.	$[\text{TetR}] \cdot [\lambda\text{cI}]$	44
5.	$[\lambda\text{cI}] \cdot [\text{LacI}]$	4
6.	$[\text{TetR}] \cdot [\text{LacI}]$	44

TABLE 2
Constraints on the FSP for the repressilator model.

For the option the explore the FSP states, the adaptive variants choose a small set of states at the beginning of the integration and explore more states dynamically as explained in section 3. The static FSP variants solve the truncated FSP systems with a large set of states fixed for the whole integration time. The constraint bounds for the static variants is chosen as the final constraints ouput from the respective adaptive variants.

The marginal distributions computed from the FSP solution at the final time is given in Fig. 4, with no visible difference in computational outputs between different FSP options. On the performance side, within the shape choice of shape constraints, there is no difference between the adaptive and the static variants 6. However, since an appropriate state space size is generally difficult to know beforehand, it is more convenient to use adaptive FSP. For the FSP shape constraints, we see that using the extended set of constraints results in a smaller state space (Fig. 5A), which means that the truncated ODEs systems to be solved in these variants are smaller. The FSP variants that use these nonlinear constraints consistently reduce the average workload per processor (in terms of FLOPs) across the numbers of processors (Fig. 5B), and consequently the computational time (Fig. 6). In addition, we also record the ratio of the maximum vs minimum workload per processor (Fig. 5C). These ratios stay well below 1.1, indicating that the dynamic load-balancing methods were able to distribute the computation almost equally among processes. On the other hand, parallel effeciency degrades as the number of parallel processes increases, with all variants drop below 50 percent effeciency when executed on 32 processors. This is because the gain from dividing the problem across the fast Intel Xeon processors has been offset by the overhead of managing inter-process communication. Nevertheless, there is additional speedup in all major components of the code (Fig. 7) as we increase the number of processes. This is a relatively small problem that could be solved on one core in an order of minutes, so the gains from parallelization is expectedly moderate.

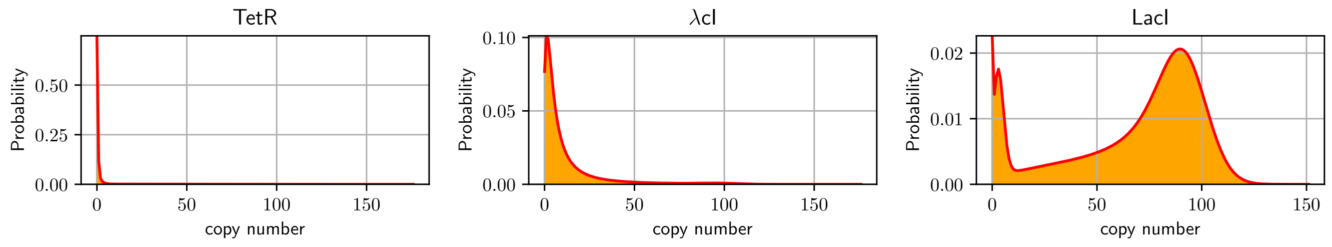

FIG. 4. *Repressilator example. Marginal distribution of* TetR, λcI *and* LacI *at time* $t_f = 10$ *minutes.*

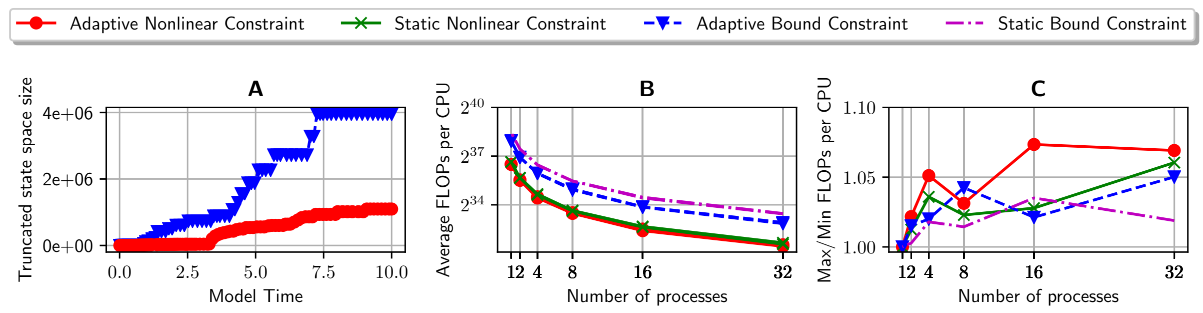

FIG. 5. *Workload in the numerical integration of the repressilator example. (A): Size of the FSP staet space over time for the adaptive variants, using the hyperectangular shape vs the nonlinear shape with additional constraints (cf. Table. 2). (B): average number of floating point operations (FLOPs) per process for four different choices of FSP shape and adaptivity. (C): The load-imbalance ratio, defined as the ratio between the maximum and minimum numbers of FLOPs per process, in the four algorithmic variants.*

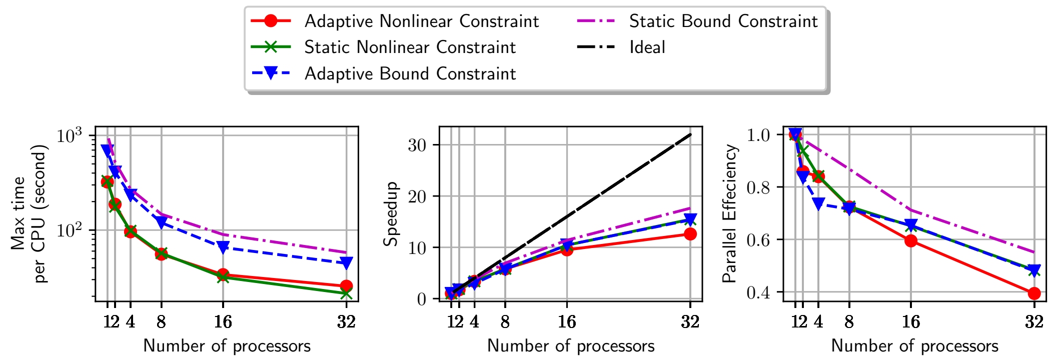

FIG. 6. *Computational time, speedup, and parallel effeciency in the parallel integration of the repressilator example. The four curves in each plot correspond to different choices for the FSP, using either the hyperectangular shape or the nonlinear shape with additional constraints (cf. Table. 2), with either adaptive or static FSP.*

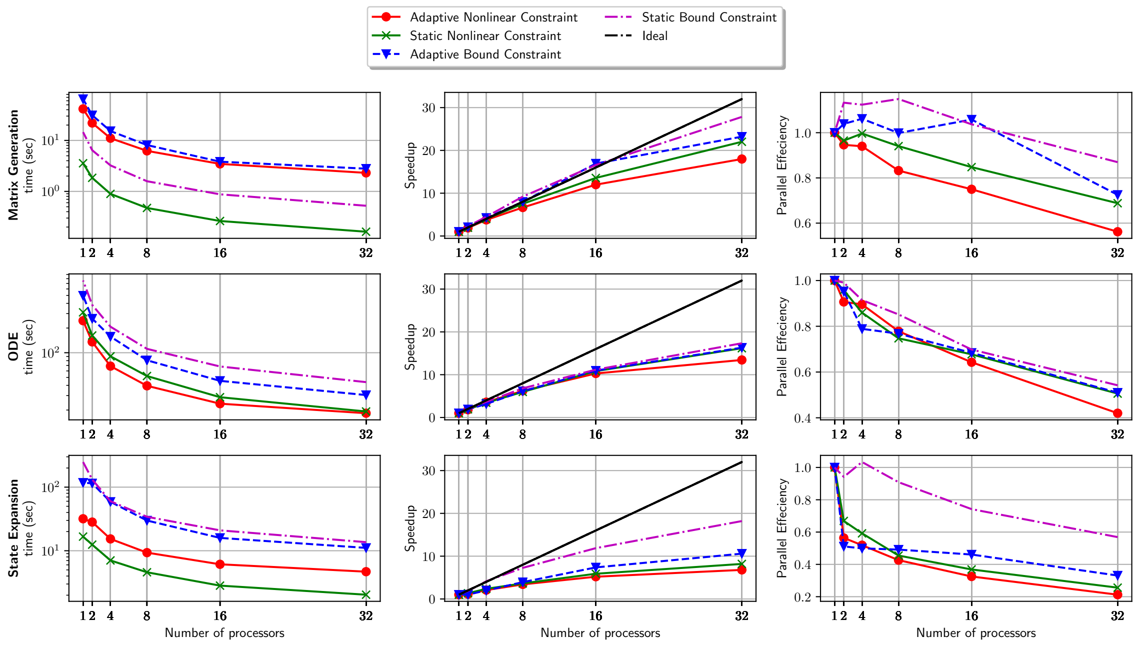

FIG. 7. *Time spent in the critical components in the parallel solution of repressilator example. These critical components include matrix generation (first row), solution of the truncated ODEs systems (second row), and state space expansion (third row). We compare the computational time, speedup, and parallel effeciency of four algorithmic variants, based on either using an adaptive of static FSP, and on using the nonlinear or hyper-rectangular constraints.*

5.2. Six-species transcription regulation. We next consider a transcription regulation model introduced in [28]. In this model, the transcription-translation process of a protein M (monomer) is activated when a transcription factor D (dimer) binds to the DNA at a single site, but is repressed when D binds to both sites of the

DNA's promoter region. The propensities associated with second-order reactions are time-varying, due to the change in the cell's volume (Table 3). This model results in a time-dependent distribution that spreads out over an increasing number of states over time, and is usually used as a benchmark problem for direct CME methods [7, 56, 51]. Most of the previous works only attempt at the time-invariant version of this model. We are only aware of the paper of Dinh and Sidje [15] that makes an attempt at directly solving the CME with time-varying propensities in serial. Here we revisit the time-varying version of this model with our novel parallel FSP implementation.

	reaction	propensity	rate constant (s^{-1})
1.	$RNA \longrightarrow RNA + M$	$c_1[RNA]$	$c_1 = 0.043$
2.	$M \longrightarrow \emptyset$	$c_2[M]$	$c_2 = 0.0007$
3.	$DNA.D \longrightarrow RNA + DNA.D$	$c_3[DNA.D]$	$c_3 = 0.0715$
4.	$RNA \longrightarrow \emptyset$	$c_4[RNA]$	$c_4 = 0.0039$
5.	$DNA + D \longrightarrow DNA.D$	$c_5(t)[DNA][D]$	$c_5(t) = 0.012 \times 10^9/A \cdot V(t)$
6.	$DNA.D \longrightarrow DNA + D$	$c_6[DNA.D]$	$c_6 = 0.4791$
7.	$DNA.D + D \longrightarrow DNA.2D$	$c_7(t)[DNA.D][D]$	$c_7(t) = 0.00012 \times 10^9/A \cdot V(t)$
8.	$DNA.2D \longrightarrow DNA.D + D$	$c_8[DNA.2D]$	$c_8 = 0.8765 \times 10^{-11}$
9.	$M + M \longrightarrow D$	$\frac{c_9(t)}{2}[M]([M]-1)$	$c_9(t) = 0.05 \times 10^9/A \cdot V(t)$
10.	$D \longrightarrow M + M$	$c_{10}[D]$	$c_{10} = 0.5$

Table 3

Reaction channels in the six-species transcription regulation model. The parameter $A = 6.0221415 \times 10^{23}$ *is Avogadro's number, and* $V = 10^{-15}2^{t/\tau}L$ *is the system volume chosen for the numerial test. Here,* τ *is the average cell cycle time, which we set to* $35min$ *based on [28] ([X] is the number of copies of the species X.)*

The FSP error tolerance is set to 10^{-4}. We start at the initial state

$$\boldsymbol{x}_0 = ([M], [D], [DNA], [DNA.D], [DNA.2D], [RNA]) = (2, 6, 0, 2, 0, 0)$$

and set the final time to $t_f = 5$ minutes. The FSP tolerance was set to 10^{-4}. For this problem, we use the default hyper-rectangular shape for the FSP-truncated state space. Similar to the previous example, we compare the adaptive and static FSP variants, with the latter using the final bounds output by the former. In addition, we compare the effects of using a simple load-balancing apporach based on distributing states equally, and a graph-based approach. All variants use the BDF method implemented in SUNDIALS [30] for solving the truncated ODEs systems (c.f., eq. (3.7)).

Fig. 8 shows the marginal distributions at the final time, and Fig. 9A shows the size of the dynamic state space. Increasing the number of processes consistently reduce the workload per process (Fig. 9B), and the total workload is distributed almost equally among processors in all variants (Fig. 9C).

We plot the solver runtime across different number of processors in Fig. 10. Interestingly, using the more advanced graph-based techinique did not improve runtime significantly, with the graph-based partitioning incurring a slight increase in time spent in state expansion (Fig. 11) due to additional overhead for generating the graph. On the other hand, using the adaptive FSP is significantly faster than the static approach (Fig. 11), despite the fact that the adaptive variant has some extra overhead for dynamically re-distributing the states at the intermediate timesteps.

The single-core performance of our adaptive FSP implementation is comparable, if not faster, than the previous approach reported in [15]. In particular, the experiment reported in [15] solve the same problem in the order of 4×10^4 second, running on a single Intel Xeon E5-4640 CPU at 2.4GHz, while our single-core variants finish the problem in the order of 10^3 second. This suggests that our FSP implementation could be as effecient as comtemporary methods when running on a single core, and the speedup from using more cores, a unique advantage of our parallel implementation, can significantly reduce the time spent in the forward solution of the CME on this example.

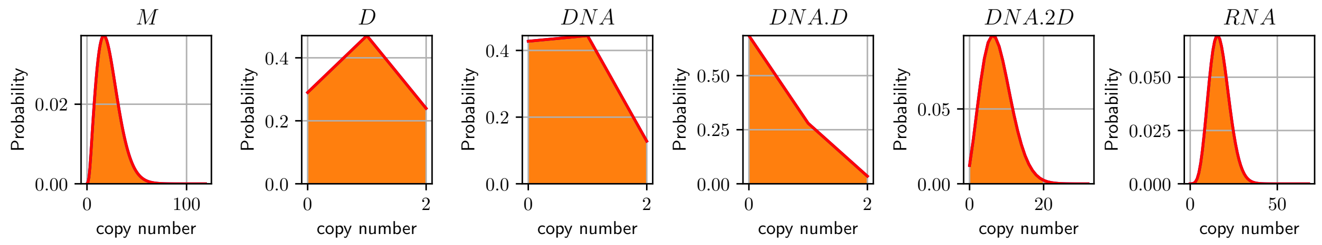

FIG. 8. *Transcription regulation example. Marginal distributions of monomer, dimer and DNA at time $t_f = 5$ minutes.*

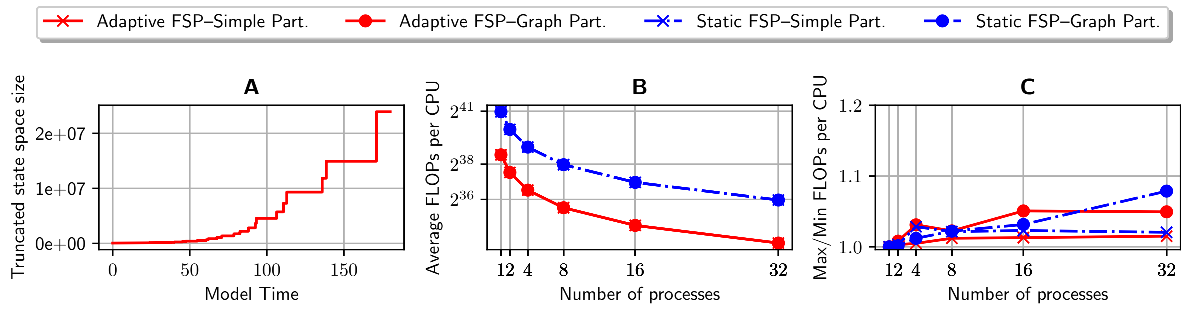

FIG. 9. *Workload in the numerical integration of the six-species transcription regulation example. (A): Size of the FSP state space over time for the adaptive FSP variants. (B): average number of floating point operations (FLOPs) per process for four different choices of FSP adaptivity and load-balancing. (C): The load-imbalance ratio, defined as the ratio between the maximum and minimum numbers of FLOPs per process, in the four algorithmic variants.*

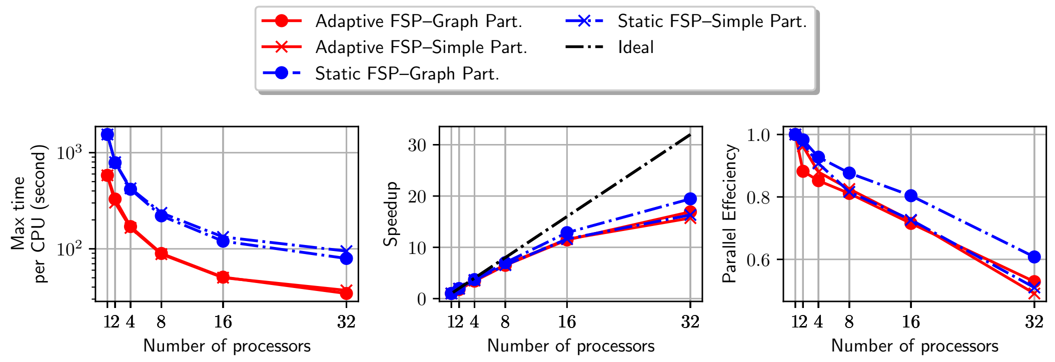

FIG. 10. *Computational time, speedup, and parallel effeciency in the parallel integration of the transcription regulation example. The four curves correspond to different choices of the FSP adaptivity (either adaptive or fixed), and the load-balancing model (graph-based or simple). The straight diagonal line in the middle plot represents the ideal linear speedup.*

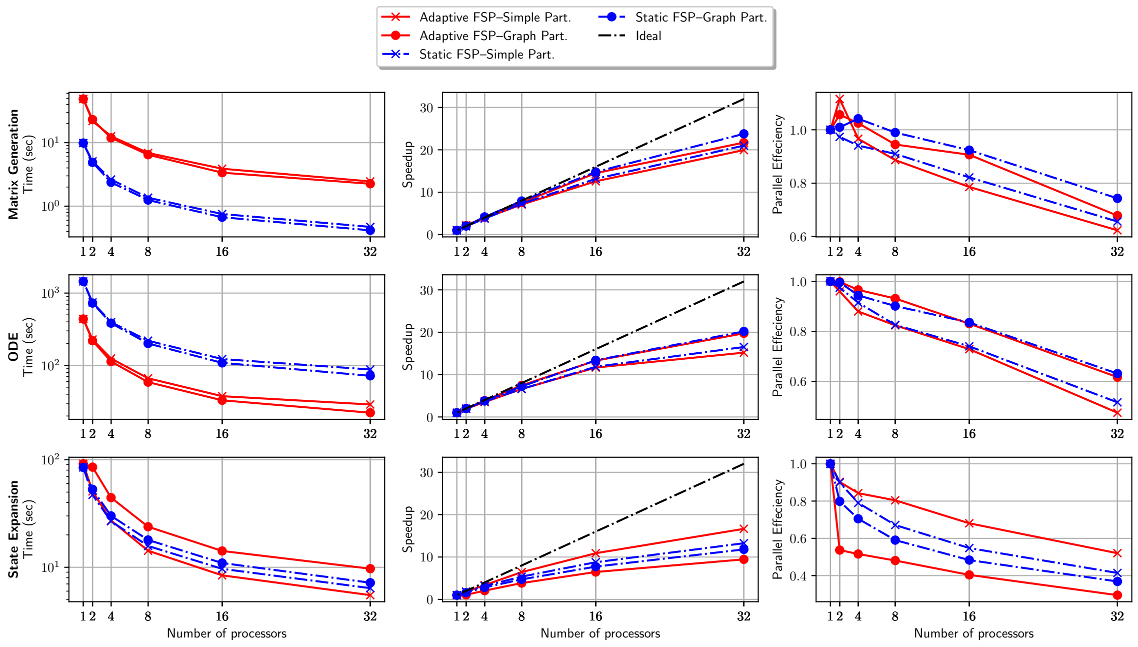

FIG. 11. *Detail timing for critical components in the parallel solution of transcription regulation example. These critical components include matrix generation (first row), solution of the truncated ODEs systems (second row), and state space expansion (third row). We compare the computational time, speedup, and parallel effeciency of four algorithmic variants based on the choice of FSP algorithm, either adaptive or static, and the choice of load-balancing method, either simple or graph-based.*

5.3. A five-species signal-activated gene expression model. The final model that we consider is an extension of the spatial signal-activated gene expression model that was previously fit to MAPK-activated gene expression data in yeast [43, 45]. We consider a single gene with four states that can transcribe two different RNA species. This could occur, for example, due to overlapping or competing promoter sites. Each RNA species is transcribed in the nucleus, then transported to the cytoplasm where they can be degraded (table 4). Genes only transcribe RNA when in active states (indexed by $2, 3, 4$ in the model). The rate of gene switching to inactive state is modulated by the time-varying Hog1p signal, modeled by

$$\text{hog1p}(t) = A_{hog}\left(\frac{h_1(t)}{1 + h_1(t)/M_{hog}}\right)^{\eta}, \tag{5.1}$$

where $h_1(t) = (1.0 - \exp(-r_1 * t)) * \exp(-r_2 * t)$ and the remaining parameters are $r_1 = 6.9e - 5$, $r_2 = 7.1e - 3$, $\eta = 3.1$, $A_{hog} = 9.3 \times 10^9$, $M_{hog} 6.4 \times 10^{-4}$.

We start at

$$\boldsymbol{x}_0 := (\text{Gene}, [RNA_{1,nuc}], [RNA_{2,nuc}], [RNA_{1,cyt}], [RNA_{2,cyt}]) = (OFF, 0, 0, 0, 0),$$

and integrate the CME up to time $t_f = 180$ second. We use the FSP tolerance of 10^{-4} and an adaptive hyper-rectangular FSP. All variants use the BDF method implemented in SUNDIALS [30] for solving the truncated ODEs systems (c.f., eq. (3.7)).

We compare the runtime of two variants: one using the simple partitioning scheme and the other the Graph-based partitioning scheme. Interestingly, we found that using simple partitioning results in shorter overall runtime across different number of processors, with nearly ideal parallel speedup (Fig. 14). In order to gain more insights into the performance difference between the two variants, we compare the time each variant spends in the two major components of the solver: the ODE integrator time (which includes the cost of matrix-vector multplications), and the state set computation (including state exploration and partitioning). We see that distributing the states using the Graph partitioner indeed result in shorter time for the ODE integrator, as the Graph partitioner seeks to reduce the communication cost of the matrix-vector multplications (Fig. 15). However, this saving in the ODE component is outweighted by the higher cost of the more elaborate partitioning method, which inflates the time spent in the state set component (Fig. 15).

Another interesting observation is that the state set component achieves superlinear speedup when using the simple partitioning scheme. This could be explained by the fact that the number of states needed for the FSP in this example is very large, eventually reaching $23,908,221$ states (Fig. 14), with the marginal distributions of the RNA having long tails (Fig. 12). This means that the speed of exploring these states with a single core will be limited by memory bandwidth, whereas distributing the exploration among many cores allow for better use of computer memory.

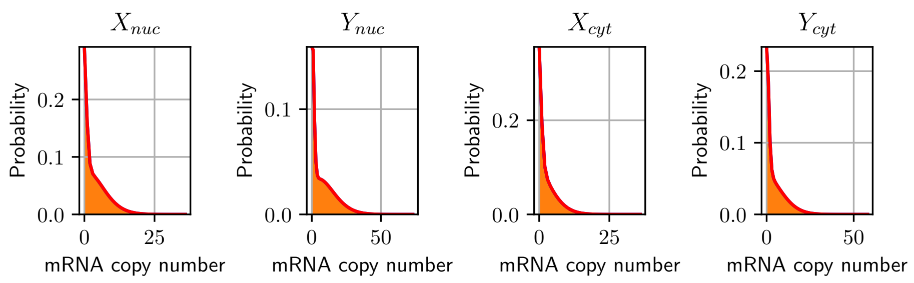

FIG. 12. *Signal-activated gene expression example. Marginal distribution of the RNA species at 3 minutes after signal activation.*

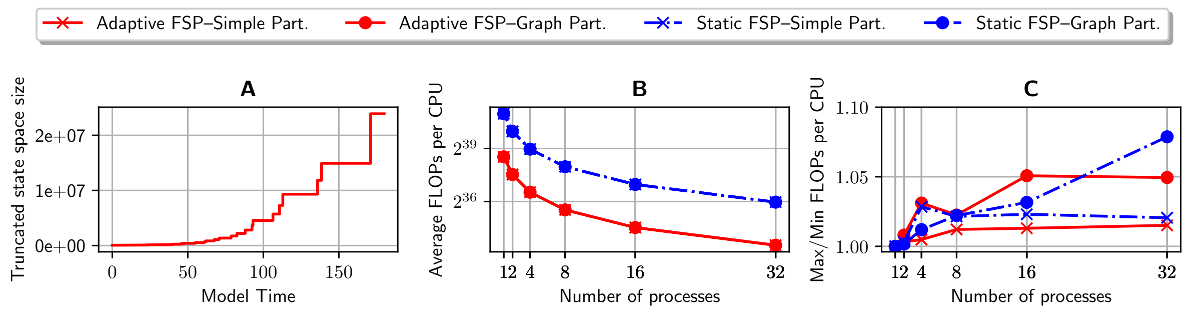

FIG. 13. *Computational time, speedup, and parallel effeciency in the parallel integration of the signal-activated gene expression example. The four curves correspond to different choices of the FSP adaptivity (either adaptive or fixed), and the load-balancing model (graph-based or simple). The straight diagonal line in the middle plot represents the ideal linear speedup.*

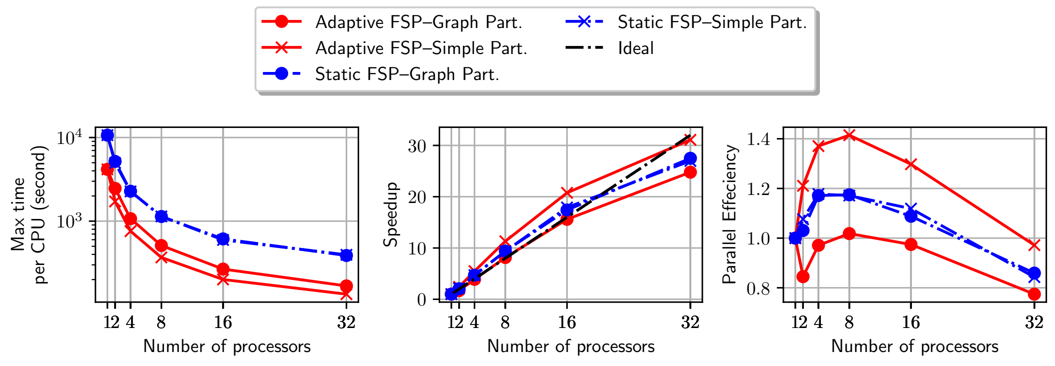

FIG. 14. *Computational time, speedup, and parallel effeciency in the parallel integration of the signal-activated gene expression example. The four curves correspond to different choices of the FSP adaptivity (either adaptive or fixed), and the load-balancing model (graph-based or simple). The straight diagonal line in the middle plot represents the ideal linear speedup.*

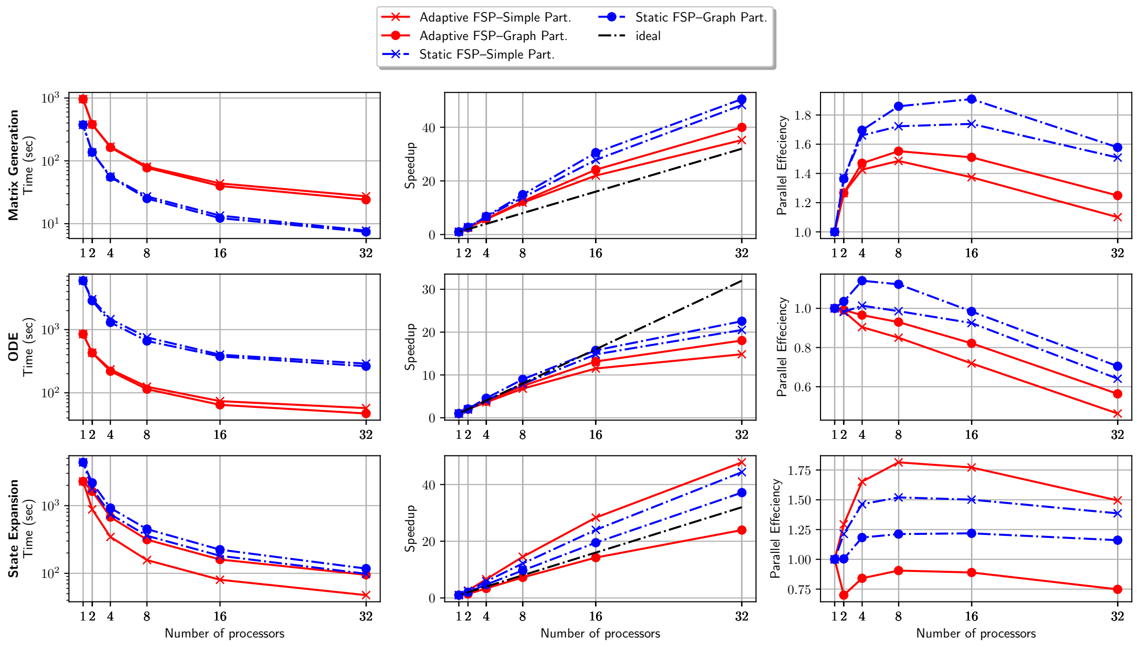

FIG. 15. *Detail timing for critical components in the parallel solution of signal-activated gene expression example. These critical components include matrix generation (first row), solution of the truncated ODEs systems (second row), and state space expansion (third row). We compare the computational time, speedup, and parallel effeciency of four algorithmic variants based on the choice of FSP algorithm, either adaptive or static, and the choice of load-balancing method, either simple or graph-based.*

	reaction	propensity
1.	$G_i \xrightarrow{k_{i,(i+1)}} G_{i+1}$	$\alpha_1 = k_{i,i+1}, \quad i = 1,2,3$
2.	$G_2 \xrightarrow{k_{2,1}(t)} G_1$	$\alpha_2(t) = \max(0, 3200.0 - 7710.0 * (hog1p(t)))$
3.	$G_i \xrightarrow{k_{i,(i-1)}} G_{i-1}$	$\alpha_3 = k_{i,(i-1)}$
4.	$G_i \xrightarrow{r_i} G_i + RNA_{1,nuc}$	$\alpha_4 = r_i[G_i]$
5.	$G_i \xrightarrow{r_i} G_i + RNA_{2,nuc}$	$\alpha_5 = r_i[G_i]$
6.	$RNA_{1,nuc} \xrightarrow{k_{trans}} RNA_{1,cyt}$	$\alpha_8 = k_{trans}[RNA_{1,nuc}]$
7.	$RNA_{2,nuc} \xrightarrow{k_{trans}} RNA_{2,cyt}$	$\alpha_9 = k_{trans}[RNA_{2,nuc}]$
8.	$RNA_{1,cyt} \xrightarrow{\gamma_{cyt}} \emptyset$	$\alpha_{10} = \gamma_1[RNA_{1,cyt}]$
9.	$RNA_{2,cyt} \xrightarrow{\gamma_{cyt}} \emptyset$	$\alpha_{11} = \gamma_2[RNA_{2,cyt}]$

TABLE 4

Five-species signal-activated gene expression reactions and propensities. The gene is considered as one species with 4 different states G_i, $i = 0, \ldots, 3$.

Parameter	Value
k_{12}	1.29
k_{23}	0.0067
k_{34}	0.133
k_{32}	0.027
k_{43}	0.0381
r_1	0
r_2	0.005
r_3	0.45
r_4	0.025
k_{trans}	0.01
γ_1	0.001
γ_2	0.0049

TABLE 5

Parameters in the five-species signal-activated gene expression example. We assume that the time unit is seconds (sec)*. Hence, the parameters' units are* sec^{-1}.

6. Conclusion. In this paper, we presented a novel parallel implementation for an adaptive finite state projection for solving the chemical master equation with time-varying propensities. We test several combinations of FSP state space expansion and load-balancing techniques on sizable problems that require up to 23 million states. Our numerical tests suggest that the largest performance gain comes from the ability of the solver to expand the state space adaptively in parallel, which is a novel feature of our solver in comparison to previous attempts at parallelizing the FSP [52, 58].

While our present implementation has shown significant speedup on realistically large CMEs, there is still room for improvement. First, our object-oriented implementation provides a convenient starting point to parallelize and test alternative strategies for updating the finite state projection, such as sliding window [56] and simulation-driven FSP [51, 54], which have been shown to be competitive with the original FSP in serial settings. Second, recent advances in hybrid MPI-CUDA support in PETSc [40] could provide additional opportunities for the acceleration of the matrix-vector multiplication. Likewise, the task of exploring the state space could utilize the massively parallel GPU cores.

We are exploring the application of our solver to build high performance computational pipelines for the analysis and design of single-cell experiments. For example, we can use our parallel solver to compute more quickly the bounds on the likelihood of single-cell data [21], which facilitates model comparison and selection. It is also possible to make more effecient use of high-performance computing resources for performing Bayesian inference, as we recently explored in [13].

Acknowledgement. HV and BM were supported by National Institutes of Health (R35 GM124747).

References.

[1] S. Abhyankar et al. *PETSc/TS: A Modern Scalable DAE/ODE Solver Library.* Preprint ANL/MCS-P5061-0114. ANL, 2014.

[2] A. Ale, P. Kirk, and M. P. Stumpf. "A general moment expansion method for stochastic kinetic models". In: *J. Chem. Phys.* 141.8 (2014), pp. 174101–185101. ISSN: 00219606. DOI: 10.1063/1.4892838.

[3] D. F. Anderson, A. Ganguly, and T. G. Kurtz. "Error analysis of tau-leap simulation methods". In: *Ann. Appl. Probab.* 21.6 (2011), pp. 2226–2262. DOI: 10.1214/10-AAP756. arXiv: arXiv:0909.4790v3.

[4] D. F. Anderson and D. J. Higham. "Multilevel Monte Carlo for continuous time Markov Chains, with applications in biochemical kinetics". In: *Multiscale Model. Simul.* 10.1 (2012), pp. 146–179. ISSN: 15403459. DOI: 10.1137/110840546. arXiv: 1107.2181.

[5] S. Balay et al. "Efficient Management of Parallelism in Object-Oriented Numerical Software Libraries". In: *Mod. Softw. Tools Sci. Comput.* Boston, MA: Birkhäuser Boston, 1997, pp. 163–202. DOI: 10.1007/978-1-4612-1986-6_8.

[6] S. Balay et al. *PETSc Users Manual.* Tech. rep. Argonne National Laboratory, 2019.

[7] K. Burrage et al. "A Krylov-based finite state projection algorithm for solving the chemical master equation arising in the discrete modelling of biological systems". In: *Proc. 150th Markov Anniv. Meet.* Ed. by A. Langeville and W. J. Stewart. Charleston, SC, USA: Boson books, 2006, pp. 1–18. ISBN: 1932482342.

[8] Y. Cao, A. Terebus, and J. Liang. "Accurate Chemical Master Equation Solution Using Multi-Finite Buffers". In: *Multiscale Model. Simul.* 14.2 (2016), pp. 923–963. ISSN: 1540-3459. DOI: 10.1137/15M1034180.

[9] Y. Cao, D. T. Gillespie, and L. R. Petzold. "Adaptive explicit-implicit tau-leaping method with automatic tau selection." In: *J. Chem. Phys.* 126.22 (2007), p. 224101. ISSN: 0021-9606. DOI: 10.1063/1.2745299.

[10] Y. Cao, D. T. Gillespie, and L. R. Petzold. "Efficient step size selection for the tau-leaping simulation method." In: *J. Chem. Phys.* 124.4 (2006), p. 044109. ISSN: 0021-9606. DOI: 10.1063/1.2159468.

[11] U. V. Catalyurek and C. Aykanat. "Hypergraph-Partitioning-Based Decomposition for Parallel Sparse-Matrix Vector Multiplication". In: *IEEE Trans. Parallel Distrib. Syst.* 10.7 (1999), pp. 673–693.

[12] U. V. Catalyurek et al. "Hypergraph-based dynamic load balancing for adaptive scientific computations". In: *Proc. - 21st Int. Parallel Distrib. Process. Symp. IPDPS 2007; Abstr. CD-ROM.* 2007. ISBN: 1424409101. DOI: 10.1109/IPDPS.2007.370258.

[13] T. A. Catanach, H. D. Vo, and B. Munsky. *Bayesian inference of Stochastic reaction networks using Multifidelity Sequential Tempered Markov Chain Monte Carlo.* Tech. rep. 2020. arXiv: 2001.01373.

[14] K. Devine et al. "Zoltan data management services for parallel dynamic applications". In: *Comput. Sci. Eng.* 4.2 (2002), p. 90. ISSN: 15219615. DOI: 10.1109/5992.988653.

[15] K. N. Dinh and R. B. Sidje. "An adaptive Magnus expansion method for solving the chemical master equation with time-dependent propensities". In: *Journal of Coupled Systems and Multiscale Dynamics* 5.2 (2017), pp. 119–131. ISSN: 2330-152X. DOI: doi:10.1166/jcsmd.2017.1124.

[16] G. R. Dowdy and P. I. Barton. "Dynamic Bounds on Stochastic Chemical Kinetic Systems through Semidefinite Programming". In: *Comput. Aided Chem. Eng.* Vol. 44. 2018, pp. 2425–2430. DOI: 10.1016/B978-0-444-64241-7.50399-2. arXiv: 1802.04409.

[17] G. R. Dowdy and P. I. Barton. "Using Semidefinite Programming to Calculate Bounds on Stochastic Chemical Kinetic Systems at Steady State". In: *Comput. Aided Chem. Eng.* Vol. 40. Elsevier B.V., 2017, pp. 2239–2244. DOI: 10.1016/B978-0-444-63965-3.50375-5.

[18] M. B. Elowitz and S. Leibler. "A synthetic oscillatory network of transcriptional regulators." In: *Nature* 403.6767 (2000), pp. 335–338. ISSN: 0028-0836. DOI: 10.1038/35002125.

[19] M. B. Elowitz et al. "Stochastic gene expression in a single cell". In: *Science (80-.).* 297.5584 (2002), pp. 1183–1186. ISSN: 1095-9203. DOI: 10.1126/science.1070919.

[20] Z. R. Fox and B. Munsky. "The finite state projection based Fisher information matrix approach to estimate information and optimize single-cell experiments". In: *PLOS Comput. Biol.* 15.1 (2019). Ed. by O. A. Igoshin, e1006365. ISSN: 1553-7358. DOI: 10.1371/journal.pcbi.1006365.

[21] Z. Fox, G. Neuert, and B. Munsky. "Finite state projection based bounds to compare chemical master equation models using single-cell data". In: *J. Chem. Phys.* 145.7 (2016). ISSN: 00219606. DOI: 10.1063/1.4960505.

[22] L. Gauckler and H. Yserentant. "Regularity and approximability of the solutions to the chemical master equation". In: *ESAIM. Math. Model. ...* 48 (2014), pp. 1757–1775. DOI: 10.1051/m2an/2014018.

[23] S. Gaudreault and J. A. Pudykiewicz. "An efficient exponential time integration method for the numerical solution of the shallow water equations on the sphere". In: *J. Comput. Phys.* 322 (2016), pp. 827 –848. ISSN: 0021-9991. DOI: https://doi.org/10.1016/j.jcp.2016.07.012.

[24] S. Gaudreault, G. Rainwater, and M. Tokman. "KIOPS: A fast adaptive Krylov subspace solver for exponential integrators". In: *J. Comput. Phys.* 372 (2018), pp. 236 –255. ISSN: 0021-9991. DOI: https://doi.org/10.1016/j.jcp.2018.06.026.

[25] M. a. Gibson and J. Bruck. "Efficient exact stochastic simulation of chemical systems with many species and many channels". In: *J. Phys. Chem. A* 104.9 (2000), pp. 1876–1889. ISSN: 1089-5639. DOI: 10.1021/jp993732q.

[26] D. T. Gillespie. *A general method for numerically simulation the stochastic time evolution of coupled chemical reactions.* 1976. DOI: 10.1016/0021-9991(76)90041-3.

[27] D. T. Gillespie. "Approximate accelerated stochastic simulation of chemically reacting systems". In: *J. Chem. Phys.* 115.4 (2001), pp. 1716–1733. ISSN: 00219606. DOI: 10.1063/1.1378322.

[28] J. Goutsias. "Quasiequilibrium approximation of fast reaction kinetics in stochastic biochemical systems." In: *J. Chem. Phys.* 122.18 (2005), p. 184102.

[29] A. Gupta, J. Mikelson, and M. Khammash. "The finite state projection algorithm for the stationary solution of the chemical master equation". In: (2017). ISSN: 00219606. DOI: doi:10.1063/1.2145882.

[30] A. C. Hindmarsh, R. Serban, and D. R. Reynolds. *User Documentation for cvode v4.1.0 (sundials v4.1.0).* Tech. rep. 2019.

[31] G. Karypis, K. Schloegel, and V. Kumar. *Parmetis: Parallel graph partitioning and sparse matrix ordering library.* Minneapolis, MN, 2013.

[32] V. Kazeev et al. "Direct Solution of the Chemical Master Equation using Quantized Tensor Trains". In: *PLoS Comput Biol* 10.3 (2014).

[33] I. Krishnarajah et al. "Novel moment closure approximations in stochastic epidemics". In: *Bull. Math. Biol.* 67.4 (2005), pp. 855–873. ISSN: 00928240. DOI: 10.1016/j.bulm.2004.11.002.

[34] N. J. Kuntz et al. "Bounding the stationary distributions of the chemical master equation via mathematical programming". In: *Journal of Chemical Physics* 151 (2019). DOI: 10.1063/1.5100670.

[35] C. H. Lee, K.-H. Kim, and P. Kim. "A moment closure method for stochastic reaction networks." In: *J. Chem. Phys.* 130.13 (2009), p. 134107. ISSN: 1089-7690. DOI: 10.1063/1.3103264.

[36] C Lester et al. "An adaptive multi-level simulation algorithm for stochastic biological systems". In: *J. Chem. Phys.* 142.2 (2015). ISSN: 00219606. DOI: 10.1063/1.4904980. arXiv: 1409.1838.

[37] J. Loffeld and M. Tokman. "Implementation of Parallel Adaptive-Krylov Exponential Solvers for Stiff Problems". In: *SIAM Journal on Scientific Computing* 36.5 (2014), pp. C591–C616. DOI: 10.1137/13094462X. eprint: https://doi.org/10.1137/13094462X.

[38] M. Maggioni. "GPU-based Linear Algebra for Calculating Steady-State Probability and Dynamics of Molecular Networks". In: (2012). ISSN: 0925-4005. DOI: 10.1016/j.snb.2015.06.051.

[39] P. Milner, C. S. Gillespie, and D. J. Wilkinson. "Moment closure approximations for stochastic kinetic models with rational rate laws". In: *Math. Biosci.* 231.2 (2011), pp. 99–104. ISSN: 00255564. DOI: 10.1016/j.mbs.2011.02.006.

[40] V. Minden, B. Smith, and M. G. Knepley. "Preliminary Implementation of PETSc Using GPUs". In: *GPU Solutions to Multi-scale Problems in Science and Engineering.* Ed. by D. A. Yuen et al. Berlin, Heidelberg: Springer Berlin Heidelberg, 2013, pp. 131–140. ISBN: 978-3-642-16405-7. DOI: 10.1007/978-3-642-16405-7_7.

[41] B. Munsky. "The finite state projection approach for the solution of the master equation and its applications to stochastic gene regulatory networks". In: June (2008).

[42] B. Munsky and M. Khammash. "The finite state projection algorithm for the solution of the chemical master equation". In: *J. Chem. Phys.* 124.4 (2006), p. 44104. ISSN: 00219606. DOI: doi:10.1063/1.2145882.

[43] B. Munsky et al. "Distribution shapes govern the discovery of predictive models for gene regulation". In: *Proc. Natl. Acad. Sci.* 115.29 (2018), pp. 7533–7538. ISSN: 0027-8424. DOI: 10.1073/pnas.1804060115.

[44] B. Munsky. "Modeling Cellular Variability". In: *Quant. Biol. from Mol. to Cell. Syst.* Ed. by M. E. Wall. Taylor & F. 2012. Chap. 11, pp. 234–266.

[45] G. Neuert et al. "Systematic Identification of Signal-Activated Stochastic Gene Regulation". In: *Science (80-.).* 339.6119 (2013), pp. 584–587. ISSN: 0036-8075. DOI: 10.1126/science.1231456.

[46] J. Niesen and W. M. Wright. "Algorithm 919: A Krylov Subspace Algorithm for Evaluating the φ-Functions Appearing in Exponential Integrators". In: *ACM Trans. Math. Softw.* 38.3 (Apr. 2012). ISSN: 0098-3500. DOI: 10.1145/2168773.2168781.

[47] M. Rathinam et al. "Consistency and stability of tau-leaping schemes for chemical reaction systems". In: *Multiscale Model. Simul.* 4.3 (2005), pp. 867–895. ISSN: 15403459. DOI: 10.1137/040603206.

[48] J. Ruess and J. Lygeros. *Moment-Based Methods for Parameter Inference and Experiment Design for Stochastic Biochemical Reaction Networks.* 2015. DOI: 10.1145/2688906.

[49] D. Schnoerr, G. Sanguinetti, and R. Grima. "Approximation and inference methods for stochastic biochemical kinetics - a tutorial review". In: *J. Phys. A* 50 (2017). ISSN: 1751-8113. DOI: 10.1088/1751-8121/aa54d9. arXiv: 1608.06582.

[50] R. B. Sidje. "Expokit: A Software Package for Computing Matrix Exponentials". In: *ACM Trans. Math. Softw.* 24.1 (Mar. 1998), 130–156. ISSN: 0098-3500. DOI: 10.1145/285861.285868.

[51] R. B. Sidje and H. D. Vo. "Solving the chemical master equation by a fast adaptive finite state projection based on the stochastic simulation algorithm". In: *Math. Biosci.* 269 (2015), pp. 10–16. ISSN: 18793134. DOI: 10.1016/j.mbs.2015.08.010.

[52] V. Sunkara. "Analysis and Numerics of the Chemical Master Equation". PhD thesis. 2013, pp. 1–134.

[53] H. D. Vo and R. B. Sidje. "An adaptive solution to the chemical master equation using tensors". In: *J. Chem. Phys.* 147.147 (2017). ISSN: 00219606. DOI: 10.1063/1.4994917.

[54] H. Vo and R. Sidje. "Improved krylov-FSP method for solving the chemical master equation". In: *Lect. Notes Eng. Comput. Sci.* Vol. 2226. 2016. ISBN: 9789881404824.

[55] H. D. Vo and R. B. Sidje. "Approximating the large sparse matrix exponential using incomplete orthogonalization and Krylov subspaces of variable dimension". In: *Numer. Linear Alg. Appl.* 24.3 (2017). e2090 nla.2090, e2090. DOI: 10.1002/nla.2090. eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1002/nla.2090.

[56] V. Wolf et al. "Solving the chemical master equation using sliding windows." In: *BMC Syst. Biol.* 4 (2010), p. 42. ISSN: 1752-0509. DOI: 10.1186/1752-0509-4-42.

[57] H. Zhang et al. "Vectorized parallel sparse matrix-vector multiplication in PETSc using AVX-512". In: *ACM Int. Conf. Proceeding Ser.* 2018. ISBN: 9781450365109. DOI: 10.1145/3225058.3225100.

[58] J. Zhang et al. "Parallel solution of the chemical master equation". In: *Proc. 2009 Spring Simul. Multiconference* 1 (2009), 112:1–112:5.