SUPPLEMENTARY INFORMATION

GPU-powered model analysis with PySB/cupSODA

Leonard A. Harris[1,2,*], Marco S. Nobile[3,4,*], James C. Pino[2,5,*], Alexander L. R. Lubbock[1,2], Daniela Besozzi[3,4], Giancarlo Mauri[3,4], Paolo Cazzaniga[4,6,†], and Carlos F. Lopez[1,2,†]

[*]These authors contributed equally.
[†]To whom correspondence should be addressed: c.lopez@vanderbilt.edu, paolo.cazzaniga@unibg.it

[1]Department of Cancer Biology, Vanderbilt University School of Medicine, Nashville, TN, USA
[2]Quantitative Systems Biology Center, Vanderbilt University School of Medicine, Nashville, TN, USA
[3]Department of Informatics, Systems and Communication, University of Milano-Bicocca, Milan, Italy
[4]SYSBIO.IT Centre of Systems Biology, Milan, Italy
[5]Chemical and Physical Biology Graduate Program, Vanderbilt University, Nashville, TN, USA
[6]Department of Human and Social Sciences, University of Bergamo, Bergamo, Italy

Contents

1 Installation instructions

1.1 PySB

The PySB/cupSODA interface and sensitivity analysis tool used here are available as of PySB version 1.4.0. PySB can be downloaded from the Python Package Index (PyPI) using the Python package manager `pip` with the command `pip install pysb`. PySB requires BioNetGen (bionetgen.org/index.php/Download) and the Python packages `numpy`, `scipy`, and `sympy`. Complete installation instructions are available at pysb.readthedocs.io/en/latest/installation.html.

1.2 cupSODA

Precompiled binaries for Linux, OS/X, and Windows are available at github.com/aresio/cupSODA/releases. On Linux and OS/X, create a new directory `/usr/local/share/cupSODA`; on Windows, create the directory `C:\Program Files\cupSODA`. Place within the new directory the cupSODA binary or a symbolic link to it (note that the binary or link must be named `cupSODA`). Alternatively, the path to the binary or link can be manually set using the `set_cupsoda_path` function from `pysb/simulator/cupsoda.py`, e.g.,

```
set_cupsoda_path('/path/to/directory/')
```

Note that since cupSODA is written in CUDA, it runs exclusively on Nvidia GPUs (developer.nvidia.com/cuda-gpus).

2 Example usage

In Supplementary Figure **S1**, we show the control flow schematic of the PySB/cupSODA interface. The "User" column represents user-defined code elements/blocks that perform specific tasks. The remaining columns illustrate operations performed by PySB, external tools (such as cupSODA), and the resulting code/data structures, respectively. After the user constructs the model and calls `run_cupsoda`, PySB outputs the model to BioNetGen (BNG) language (BNGL) [1, 2] and then calls BNG to construct the reaction network (i.e., the full set of reactions and species generated by applying the model rule set to the seed species) [3]. PySB then reads in the network produced by BNG, constructs the set of cupSODA input files [4, 5], and passes them to cupSODA for simulation. The output trajectories from cupSODA are read in by PySB into a 3D matrix that can be processed and plotted to show the range of outputs.

Use of the PySB/cupSODA interface is exemplified in Supplementary Figure **S2**, where we present source code for running and plotting simulations of the Michaelis-Menten (MM) reaction set with a variety of initial conditions. The MM reaction set is comprised of three reactions,

$$E + S \underset{k_r}{\overset{k_f}{\rightleftharpoons}} ES \xrightarrow{k_{\text{cat}}} E + P,$$

where E is the enzyme, S is the substrate, ES is the enzyme-substrate complex, and P is the product. The rate constants k_f, k_r, and k_{cat} control the rates of enzyme-substrate formation, dissociation, and conversion to product, respectively. Since all initial substrate is eventually converted to product (product formation is modeled as an irreversible step), the final concentration of product obviously depends on the initial concentration of substrate. Furthermore, assuming fixed values of the rate constants, the rate at which substrate is converted to product strongly depends on the initial amount of enzyme. Therefore, we vary the initial concentrations of enzyme and substrate and plot the range of time courses for the product.

The code in Supplementary Figure **S2** is divided into three parts: (1) PySB declarations (model construction; line 1); (2) model simulation (lines 7–25); and (3) plotting of the results (lines 26–35). These code blocks correspond to the numbered tasks shown in the "User" column of Supplementary Figure **S1**. Note that in (1) the model is constructed in a separate file (`pysb/examples/michment.py`; available at github.com/LoLab-VU/PySB_cupSODA_Bioinfo2017) and imported. Also, in (2), performing the simulations requires defining the initial concentrations for all "seed" species (the species initially present in the system [1]) for all simulation runs and passing these to the `run_cupsoda` function as a dictionary. In this

case, we have two seed species (enzyme and substrate) and 11 initial concentrations for each (lines 8 and 10). Taking all possible combinations (i.e., the Cartesian product; line 12) gives 121 sets of initial concentrations. The mean, minimum, and maximum product concentrations produced by running the code in Supplementary Figure **S2** are shown in the lower-right panel of Supplementary Figure **S1**.

3 Models

3.1 Cell cycle

The Tyson cell cycle model [6] is a simplified description of the regulatory network that controls formation of the maturation-promoting factor (MPF), an enzyme necessary for entry into mitosis in eukaryotic cells. MPF is a complex comprised of two protein subunits, cdc2 (cell division cycle protein 2, also known as cyclin-dependent kinase 1, CDK1 [7]) and cyclin B1. Transcription of cyclin B1 begins at the end of the S (synthesis) phase of the cell cycle and continues throughout the G2 (Gap 2) phase [8]. The model assumes that synthesis of cyclin B1 (referred to simply as "cyclin" in [6]) occurs at a constant rate, that newly synthesized cyclin B1 is stable (degradation rate ≈ 0), and that it binds to cdc2 to form an inactive complex, known as "preMPF". wee1, a nuclear kinase, is known to inhibit MPF activity by phosphorylating cdc2 [9]. The phosphatase cdc25 activates MPF by removing the phosphate group from cdc2 [10]. Active MPF, in turn, activates cdc25 and inhibits wee1, thus amplifying its own activation through positive feedback [11]. In [6], wee1 and cdc25 are not explicitly modeled: phosphorylation of cdc2 and activation of MPF are modeled as mass-action processes with rate constants assumed to be proportional to the (implicit) concentrations of wee1 and cdc25, respectively. Autocatalytic activation of MPF is modeled as a third-order mass-action process (i.e., quadratic in active MPF) with a rate constant inversely proportional to the square of the total cdc2 concentration. Active MPF, together with polo-like kinase 1 (PLK-1), also phosphorylates and induces activation of the anaphase-promoting complex/cyclosome (APC/C), which polyubiquitinates cyclin B1, marking it for degradation and leading to disassembly of the MPF [12]. This negative feedback loop, whereby active MPF drives its own breakdown, completes the cycle and allows the cell to exit mitosis. In [6], MPF breakdown is modeled as a two-step process of cdc2/cyclin B1 dissociation followed by rapid cyclin B1 degradation (both first-order mass-action processes, i.e., APC/C and PLK-1 are not explicitly modeled). This requires distinguishing newly synthesized cyclin B1 (assumed to be stable; see above) from cyclin B1 released during MPF disassembly, which is done by assuming that the former is unphosphorylated and the latter is phosphorylated.

In this work, we have simplified the model slightly by combining MPF disassembly and cyclin B1 degradation into a single, first-order mass-action step, i.e., assuming that cyclin B1 is degraded *in situ*. This was suggested in [6] as an alternative mechanism that does not change the fundamental characteristics of the model. The cell cycle model was constructed in PySB [3] and is comprised of five species (two with initial populations > 0, i.e., `cdc2-P`$(0)=1$ a.u., `cyclin`$(0) = 0.25$ a.u. [6]; a.u., *arbitrary units*) and seven reactions. Values of all rate constants were taken from [6] and correspond to the region of parameter space where spontaneous limit cycle oscillations are observed. The model (`tyson_oscillator_in_situ.py`) is available at github.com/LoLab-VU/PySB_cupSODA_Bioinfo2017.

3.2 Ras/cAMP/PKA

The Ras/cAMP/PKA signaling pathway in yeast plays a major role in regulating cell growth and proliferation in response to nutritional sensing [13] and stress conditions [14]. In *Saccharomyces cerevisiae*, normal activity of the protein kinase A (PKA) is necessary for cellular growth and cell cycle progression. Reduction of PKA activity, which is related to a decrease in intracellular cyclic adenosine monophosphate (cAMP), leads to the accumulation of storage carbohydrates (glycogen and trehalose), high stress tolerance, and growth and cell cycle arrest [13]. The Ras/cAMP/PKA pathway is a complex signaling cascade in which cAMP is synthesized by the adenylate cyclase Cyr1 and induces the activation of the cAMP-dependent PKA protein. The adenylate cyclase activity is controlled by Ras (Ras1 and Ras2) and Gpa2 proteins [15]. Ras proteins cycle between an inactive state (bound to guanosine diphosphate (GDP)) and an active state (bound to guanosine triphosphate (GTP)) and are positively regulated by Cdc25, a Guanine Nucleotide Exchange Factor (GEF), that stimulates the GDP–GTP exchange [16], and negatively regulated by Ira1 and Ira2, two

GTPase-Activating Proteins (GAPs), that stimulate the GTPase activity of Ras. Deactivation of cAMP is governed by two phosphodiesterases, Pde1 and Pde2, which constitutes a major feedback mechanism in the pathway [17]. In general, the complex interplay between the components of the Ras/cAMP/PKA pathway poses a challenge to both computational and experimental investigations.

The Ras/cAMP/PKA model [18] was downloaded from the BioModels Database [19] in Systems Biology Markup Language (SBML) format [20] (`BIOMD0000000478.xml`). It was converted to BNGL [1] using the SBML-to-BNGL translator ("flat" translation) in BioNetGen 2.2.6 [2] and then into PySB format [3] using an in-house script. The model is comprised of 33 species (11 with initial populations > 0; Supplementary Table **S3**) and 39 reactions with mass-action rate laws (one rate constant per reaction). The PySB-encoded version of the model (`ras_camp_pka.py`) is available at github.com/LoLab-VU/PySB_cupSODA_Bioinfo2017.

3.3 EARM

The extrinsic apoptosis reaction model (EARM) [3, 21] describes the biochemical events that lead from binding of an external ligand to induction of apoptosis in mammalian cells. Tumor necrosis factor (TNF)-related apoptosis-inducing ligand (TRAIL) binds to a death receptor on the cell membrane, leading to the formation of death-inducing signaling complex (DISC), which if not inhibited by FLIP can cleave and activate caspase-8. Active caspase-8 truncates Bid (into tBid), targeting it to the mitochondrial outer membrane. When at the membrane, tBid recruits cytoplasmic Bax to be inserted into the membrane, where tBid can further activate Bax and the closely related membrane-bound protein Bak. Active Bax/Bak then homo-oligomerize, causing pore formation in the mitochondria. This process, known as mitochondrial outer membrane permeabilization (MOMP), is the "point of no return" in apoptosis [22, 23]. MOMP releases mitochondrial proteins, in particular cytochrome c and Smac. Cytochrome-c binds to the cytosolic protein Apaf-1 and to caspase-9, forming the "apoptosome," which activates caspase-3. Smac binds to XIAP, preventing XIAP from binding and inhibiting active capsase-3. Caspase-3 activation marks the beginning of the end: DNA and structural proteins degrade and the cell eventually collapses in a controlled manner [24].

The version of EARM used in this study is a slightly updated version of the "embedded" model reported in Lopez et al. [3], which itself is an expanded version of the model reported in Albeck et al. [21].[1] The model is encoded in PySB [3] and is comprised of 77 species (21 with initial populations > 0; Supplementary Table **S4**) and 105 reactions with mass-action rate laws. Values of the kinetic parameters were obtained by calibrating the model to the average time courses for three proteins (initiator caspase, mitochondrial intermembrane space reporter protein (IMS-RP), and effector caspase) in HeLa cells following TRAIL exposure [3] using a particle swarm optimization (PSO) algorithm [27, 28]. The PSO calibration was performed twice and the best fit parameter set from each run was saved for further analysis. The model (`earm_lopez_embedded_flat.py`), and the two accompanying parameter sets, are available at github.com/LoLab-VU/PySB_cupSODA_Bioinfo2017.

4 Performance analysis

4.1 Computational overhead

In Fig. **1**A–C of the main text, we compare CPU run times for PySB/cupSODA vs. SciPy/LSODA on a GeForce GTX 980 Ti GPU (*Diablo*; Supplementary Table **S1**). The PySB/cupSODA run times include computational overhead associated with construction of the cupSODA input files in the PySB frontend, reading of the input files by cupSODA, post-simulation output of the species time courses to file by cupSODA, and reading of the results from the output files back into PySB. To quantify this overhead, in Supplementary Fig. **S3** we include raw cupSODA run times alongside the PySB/cupSODA and SciPy/LSODA run times. These results show that, for large numbers of simulations, the overhead can constitute as much as 44% of the total run time for large models (EARM) and as much as 94% for small models (cell cycle). While

[1] The primary difference between the Lopez et al. [3] and Albeck et al. [21] versions of EARM is in the detail used to describe the BCL-2 family of proteins. In [3], the pro-apoptotic proteins Bax and Bak, the anti-apoptotic proteins Bcl-XL, Bcl-2, and MCL1, and two sensitizer proteins Bad and Noxa are all included. The model in [21] only includes a single effector BCL-2 family protein (Bax), a single anti-apoptotic protein (Bcl-2), and omits the sensitizer proteins. The inclusion of additional detail in [3] was used to better decipher the complex interplay that is the root of much controversy in the field [25, 26].

not insignificant, we see in Supplementary Fig. **S3**D that for large numbers of simulations ($\gtrsim$1000) this additional time cost is small compared to the computational savings achieved relative to SciPy/LSODA. Streamlining the implementation of the PySB/cupSODA interface to minimize this overhead is an area of active investigation.

4.2 Memory usage modes

CUDA threads can access a variety of different memory spaces during their execution, each differing in terms of visibility, efficiency, and data persistence [29, 30]. All threads can read and write from *registers* (i.e., local variables) and *global memory*. The latter is persistent across multiple kernel launches and represents the main communication interface with the CPU. Threads can also communicate by means of a very limited amount of *shared memory* (a few KBs, even on recent GPU architectures), which is visible to all threads belonging to the same block and persists for the lifetime of the block. Importantly, although shared memory was designed for intra-block communication, threads can exploit it as a high-performance memory thanks to the extremely low latency to data (about two orders of magnitude smaller than global memory). Threads also have access to a cached, read-only high-performance *constant memory*. This special type of memory persists for the lifetime of the application, is loaded from the CPU, and is very scarce (a few KBs). Additional memory spaces available on CUDA devices include *texture* and *pinned* memories. Since cupSODA does not leverage either of these advanced memories they will not be discussed further; we refer the interested reader to [30] for additional information.

cupSODA 1.0.0 (github.com/aresio/cupSODA/releases/tag/v1.0.0) is equipped with three memory usage modes that exploit global, shared, and constant memory spaces on CUDA devices differently. These modes can be invoked in PySB using the following keywords:

- `global`: All data structures required by cupSODA are stored in global memory. These include: (i) the stoichiometric matrices defining the reactions ($2 * $ `n_species` $\times$ `n_reactions`, i.e., both reactant and product sides); (ii) the compressed representation of the ordinary differential equations (ODEs) [5] and the corresponding Jacobian matrix [31]; (iii) the 2D array of initial species amounts (`n_species` $\times$ `n_sims`); (iv) the 2D array of kinetic parameters (`n_reactions` $\times$ `n_sims`); (v) the array of sampling times (`n_samples`); (vi) the array of species to be sampled ($\texttt{n\_species}' \leq$ `n_species`); (vii) the array of current simulation times (`n_sims`); (viii) all LSODA working data structures and settings (e.g., tolerances, `max_steps`) [32]; (ix) the 3D array of species time courses ($\texttt{n\_species}' \times$ `n_samples` $\times$ `n_sims`).

- `shared`: Same as `global` except that the current species concentrations and current simulation times are stored in shared memory. This improves the performance of the simulator because these data are frequently accessed during the course of a simulation to evaluate the right-hand sides of the ODEs, calculate elements of the Jacobian matrix, and update the species concentrations and simulation times.

- `sharedconstant` (*default*): Same as `shared` except that the compressed representation of the ODEs [5] and the Jacobian matrix are stored in constant memory. These non-mutable arrays are frequently accessed by the LSODA integrator within cupSODA during the course of performing the numerical integration of the ODEs. In particular, if the system is detected as "stiff," LSODA switches to an implicit integration algorithm that relies on the Jacobian matrix [32].

In principle, the `sharedconstant` mode is the best option since it leverages both of the low-latency memory spaces (shared and constant) to store the most frequently accessed data during the course of a simulation. We have thus made `sharedconstant` the default memory usage mode in PySB/cupSODA. However, as mentioned above, both shared and constant memory spaces are very limited resources. Thus, they may not be able to accommodate the data structures for models with very large numbers of reactions and species (which is common in rule-based modeling frameworks such as PySB [1, 3]). It is for this reason that we have provided users with the ability to change the memory usage mode using the keywords listed above.

All results presented in Fig. **1** of the main text and in Supplementary Fig. **S3** were obtained on a GeForce GTX 980 Ti GPU (*Diablo*; Supplementary Table **S1**) using the `sharedconstant` memory mode, which we found to be most efficient in almost all cases. To quantify the performance gains of this mode,

in Supplementary Fig. **S4** we compare raw cupSODA run times on this GPU for each model using each memory mode. In all cases, the shared and constant memories are sufficiently large to store the associated data structures. We see that for small models (cell cycle) there is a significant advantage to using the shared memory when running large numbers of simulations (∼19% run time reduction for `shared` relative to `global` for 100 000 simulations). Using the constant memory has minimal impact on run times in this case (run time reductions $\lesssim$5% for `sharedconstant` relative to `shared` in all cases). However, for large models (Ras/cAMP/PKA and EARM), both shared and constant memory usages provide significant run time reductions in all cases (`shared` decreases run times relative to `global` by ∼10–20% and `sharedconstant` further decreases run times relative to `shared` by an additional ∼5–15%). These results verify that as long as the capacities of the shared and constant memory spaces are not exceeded, the `sharedconstant` memory mode is the most efficient option in most cases.

4.3 GPU architecture

To investigate the effects of GPU architecture on run times, in Supplementary Fig. **S5** we compare raw cupSODA run times (`sharedconstant` memory mode) for each model on the GeForce GTX 980 Ti (*Diablo*) and on three additional GPUs (Supplementary Table **S1**). For all models, we see that for ≤1000 simulations the GeForce GTX 980 Ti and GeForce GTX 970 (*Mule*) are the fastest and perform essentially equally. However, for larger numbers of simulations the GeForce GTX 980 Ti outperforms the GeForce GTX 970. These observations can be explained in terms of the number of CUDA cores and streaming multiprocessors on each GPU. Both the GeForce GTX 980 Ti and GeForce GTX 970 have >1000 CUDA cores (Supplementary Table **S1**). Therefore, for ≤1000 simulations, both are able to perform all of the simulations in parallel (one simulation per core). And since they have comparable clock rates, they perform essentially identically. However, once the number of simulations exceeds the number of available cores, the simulations can no longer be run entirely in parallel. Briefly, GPUs are comprised of a set of "streaming multiprocessors" (SMs), each of which can accommodate at least one "block"[2] of multiple "threads" [33, 34] (one simulation per thread; here we use 16 threads/block). SMs run in parallel and blocks are scheduled over the available SMs. When the number of SMs is less than the number of blocks, the blocks are queued. Therefore, devices with more SMs run faster than devices with fewer SMs because they can run more simulations in parallel. In this case, the GeForce GTX 980 Ti has 22 SMs and the GeForce GTX 970 has 13 SMs. Therefore, for ≥10 000 simulations, assuming one block per SM, we would expect the GeForce GTX 980 Ti to outperform the GeForce GTX 970 by a factor of $22/13 \approx 1.7$. This ratio corresponds very closely to the ratios of run times for ≥10 000 simulations for all models (Supplementary Fig. **S5**D). For example, for 100 000 simulations of the cell cycle model, the ratio of run times for the GeForce GTX 970 to the GeForce GTX 980 Ti is $6.22/3.87 \approx 1.61$. Similar ratios are seen for 10 000 simulations of the cell cycle, Ras/cAMP/PKA, and EARM models.

Interestingly, the GeForce GTX 760 (*Lolab-760*) also has >1000 CUDA cores and has a clock rate comparable to those of the GeForce GTX 980 Ti and GeForce GTX 970 (Supplementary Table **S1**). Therefore, for ≤1000 simulations it should also be capable of running all simulations in parallel. Yet, we see in Supplementary Fig. **S5**D that it is significantly slower than both, in most cases by a factor of ∼2–3. We can explain this difference in terms of the maximum number of registers per thread on each GPU. Registers are high-performance memory locations used to store basic variable types, such as characters, integers, and floats. When the number of basic variables in a CUDA program exceeds the maximum number of registers in a thread, the compiler will resort to using global memory to store these variables, a phenomenon known as "register spilling" [34]. Since access to global memory is much slower than to registers, performance is significantly reduced. In this case, the GeForce GTX 760 has only 63 maximum registers/thread, while the other GeForce GPUs have 255 (Supplementary Table **S1**). We believe that this difference explains most of the reduction in performance seen for the GeForce GTX 760, i.e., cupSODA uses more than 63 registers/thread, causing register spilling on the GeForce GTX 760. However, a notable exception is the run time for 1000 simulations of EARM, the largest model that we consider (Sec. 3.3 and Table 1 of the main text), where the GeForce GTX 760 is ∼6× slower than the other GeForce GPUs (Supplementary Fig. **S5**D), a factor larger than we would expect based on register spilling alone. We cannot pinpoint the exact cause of this additional performance reduction, but we believe it to be due to a variety of additional factors that become

[2] SMs can sometimes accommodate more than one block, depending on available resources, e.g., shared memory.

important for models of this size, including the lower number of SMs and the reduced memory bandwidth on the GeForce GTX 760 (Supplementary Table **S1**).

Finally, we see in Supplementary Fig. **S5** that for all models the Tesla K20c is significantly slower than the GeForce GTX 980 Ti and GeForce GTX 970, but performs comparably to the GeForce GTX 760 for $\leq$1000 simulations and outperforms it by a factor of $\sim$2$\times$ for $\geq$10 000 simulations. The fact that the Tesla K20c does not outperform the GeForce GTX 760 for $\leq$1000 simulations is somewhat surprising given that, like the other GeForce GPUs, the Tesla K20c has 255 maximum registers/thread (Supplementary Table **S1**). It also has a slightly larger memory bandwidth relative to the GeForce GTX 760. However, some of this advantage is offset by its slower maximum clock rate (0.71 GHz). Furthermore, Tesla GPUs are equipped with error-correcting code (ECC) memory [34], a feature introduced to protect against rare random memory errors due to electrical or magnetic interference [35]. ECC improves the reliability of data stored in memory but comes at a cost: it reduces the amount of available memory (including shared memory) and introduces significant performance overheads [33, 35]. We believe that the combined effects of reduced maximum clock rate and costs associated with the ECC memory explain why the Tesla K20c does not outperform the GeForce GTX 760 for $\leq$1000 simulations. However, for $\geq$10 000 simulations, the larger number of CUDA cores, SMs, maximum registers/thread, and memory bandwidth (Supplementary Table **S1**) all together outweigh these costs, leading to the observed performance gains of the Tesla K20c (Supplementary Fig. **S5**).

5 Sensitivity analysis

In order to demonstrate the utility of PySB/cupSODA as a tool for model analysis, we performed sensitivity analyses with respect to initial species populations[3] for all three models (Sec. 3). Let $\mathbf{\Theta} = [\theta_0 \dots \theta_{N-1}]^{\mathrm{T}} \in \mathbb{R}^N_{\geq 0}$ be a vector of N initial populations of chemical species, such that $\theta_k \neq 0$ for at least one value of $k \in \{0, \dots, N-1\}$. Let also $F : \mathbf{\Theta} \to \mathbb{R}$ be a model output given $\mathbf{\Theta}$ (e.g., the population of a species at a given time point, the time required for a species population to reach a predefined threshold). We denote by $\mathbf{\Theta}^0 = [\theta^0_0 \dots \theta^0_{N-1}]^{\mathrm{T}}$ a vector of *reference* initial population values obtained, e.g., by calibrating a model to experimental data. The model *sensitivity* to changes in the initial species populations is then defined as

$$\begin{aligned} S: \ & \mathbf{\Theta} \to \mathbb{R} \\ & \mathbf{\Theta} \mapsto \frac{F(\mathbf{\Theta}) - F(\mathbf{\Theta}^0)}{F(\mathbf{\Theta}^0)} \cdot 100. \end{aligned} \tag{S1}$$

Let $\mathbf{J} = [0 \dots N-1]^{\mathrm{T}} \in \mathbb{N}^N$ be a vector of N species indices and $\mathbf{\Delta} = [\delta_0 \dots \delta_{M-1}]^{\mathrm{T}} \in \mathbb{R}^M_{\geq 0}$, $M \geq 1$, be a vector of M multiplicative factors representing perturbations to each initial species population. We generate an NM-length vector of 2-tuples $\mathbf{A} \equiv \mathrm{vec}(\mathbf{\Delta} \times \mathbf{J})$, where $\times$ denotes the Cartesian product and *vec* is the matrix vectorization. We then generate the $NM \times NM$ matrix $\mathbf{B} \equiv \mathbf{A} \times \mathbf{A}$ by taking the Cartesian product of $\mathbf{A}$ with itself. Element B_{ij} of this matrix is a 2-tuple of 2-tuples, where the first element of each inner 2-tuple is a perturbation (an element of $\mathbf{\Delta}$) and the second element is a species index (an element of $\mathbf{J}$). The perturbations and species indices relate to the row and column indices of $\mathbf{B}$ via

$$B_{ij} = \left((\delta_{i\%M}, J_{\lfloor i/M \rfloor}), (\delta_{j\%M}, J_{\lfloor j/M \rfloor})\right), \quad i, j \in \{0, \dots, NM-1\}, \tag{S2}$$

where % is the modulo operator and $\lfloor \cdot \rfloor$ is the floor operator. For convenience, we also define the vector $\mathbf{A'} \equiv \mathrm{vec}(\mathbf{1}_M \times \mathbf{J})$, where $\mathbf{1}_M = [1 \dots 1]^{\mathrm{T}}$ is an all-ones vector of length M. This allows us to define the matrix $\mathbf{B'} \equiv \mathbf{A'} \times \mathbf{A}$, the ij-th element of which is

$$B'_{ij} = \left((1, J_{\lfloor i/M \rfloor}), (\delta_{j\%M}, J_{\lfloor j/M \rfloor})\right), \quad i, j \in \{0, \dots, NM-1\}, \tag{S3}$$

i.e., the initial population of species $J_{\lfloor i/M \rfloor}$ is unperturbed.

[3] The approach described here is similar to that used by Gaudet et al. [36].

We define the function

$$G:\ B_{ij} \to \mathbb{R}^N_{\geq 0}$$
$$\begin{cases} \text{if } J_{\lfloor i/M \rfloor} \neq J_{\lfloor j/M \rfloor}: & \\ \quad \theta^0_{J_{\lfloor i/M \rfloor}} & \mapsto \quad \delta_{i\%M} \cdot \theta^0_{J_{\lfloor i/M \rfloor}} \\ \quad \theta^0_{J_{\lfloor j/M \rfloor}} & \mapsto \quad \delta_{j\%M} \cdot \theta^0_{J_{\lfloor j/M \rfloor}} \\ \quad \theta^0_k & \mapsto \quad \theta^0_k, \quad k \in \mathbf{J} \setminus \{J_{\lfloor i/M \rfloor}, J_{\lfloor j/M \rfloor}\} \\ \text{else}: & \\ \quad \theta^0_k & \mapsto \quad \theta^0_k, \quad k \in \mathbf{J} \end{cases}, \tag{S4}$$

which takes an element B_{ij} and returns a vector of initial species populations that differs from the reference vector $\mathbf{\Theta}^0$ at indices $J_{\lfloor i/M \rfloor}$ and $J_{\lfloor j/M \rfloor}$ if and only if $J_{\lfloor i/M \rfloor} \neq J_{\lfloor j/M \rfloor}$. We then define the function

$$H:\ \mathbf{B} \to \mathbb{R}^{NM \times NM}_{\geq 0}$$
$$B_{ij} \mapsto S(G(B_{ij})),\ \forall\ i,j \in \{0,\ldots,NM-1\}, \tag{S5}$$

which sequentially applies Eqs. S4 and S1 to each element of $\mathbf{B}$ to produce an $NM \times NM$ matrix of sensitivity values. Let $\mathbf{P} \equiv H(\mathbf{B})$ be the *pairwise sensitivity matrix*. Note that $\mathbf{P}$ is symmetric and that the elements of the block diagonal are all zero. Therefore, $N\cdot(N-1)\cdot M^2/2 < (NM)^2$ evaluations of F (i.e., simulations) are needed to construct it. Further, let $\mathbf{P'} \equiv H(\mathbf{B}) - H(\mathbf{B'})$ be the *normalized pairwise sensitivity matrix*, i.e., each sensitivity value is normalized by subtracting the corresponding sensitivity value in which the initial population of species $J_{\lfloor i/M \rfloor}$ is unperturbed (see Eq. S3). In principle, to construct $\mathbf{P'}$ an additional $N\cdot(N-1)\cdot M$ evaluations of F are required.[4] However, these can be avoided if 1 is contained within $\mathbf{\Delta}$.

Finally, we define the *single-parameter sensitivity multiset*[5]

$$\mathbf{Q}_k = \bigcup_{\substack{kM \leq i < (k+1)M \\ kM > j \geq (k+1)M}} P'_{ij}, \quad k \in \mathbf{J}, \tag{S6}$$

which can be plotted as a boxplot to visualize the range of model outputs due to changes in the initial population of species k.

Example: *Two species, three perturbations* ($N = 2$, $M = 3$)

$$\mathbf{\Theta}^0 = \begin{bmatrix} 100 & 100 \end{bmatrix}^{\mathrm{T}}$$
$$\mathbf{J} = \begin{bmatrix} 0 & 1 \end{bmatrix}^{\mathrm{T}}$$
$$\mathbf{\Delta} = \begin{bmatrix} 0.8 & 1.0 & 1.2 \end{bmatrix}^{\mathrm{T}}$$
$$\mathbf{A} = \mathrm{vec}\left(\begin{bmatrix} (0.8,0) & (0.8,1) \\ (1.0,0) & (1.0,1) \\ (1.2,0) & (1.2,1) \end{bmatrix}\right)$$
$$= \begin{bmatrix} (0.8,0) & (1.0,0) & (1.2,0) & (0.8,1) & (1.0,1) & (1.2,1) \end{bmatrix}^{\mathrm{T}}$$
$$\mathbf{B} = \left[\begin{array}{ccc|ccc} ((0.8,0),(0.8,0)) & ((0.8,0),(1.0,0)) & ((0.8,0),(1.2,0)) & ((0.8,0),(0.8,1)) & ((0.8,0),(1.0,1)) & ((0.8,0),(1.2,1)) \\ ((1.0,0),(0.8,0)) & ((1.0,0),(1.0,0)) & ((1.0,0),(1.2,0)) & ((1.0,0),(0.8,1)) & ((1.0,0),(1.0,1)) & ((1.0,0),(1.2,1)) \\ ((1.2,0),(0.8,0)) & ((1.2,0),(1.0,0)) & ((1.2,0),(1.2,0)) & ((1.2,0),(0.8,1)) & ((1.2,0),(1.0,1)) & ((1.2,0),(1.2,1)) \\ \hline ((0.8,1),(0.8,0)) & ((0.8,1),(1.0,0)) & ((0.8,1),(1.2,0)) & ((0.8,1),(0.8,1)) & ((0.8,1),(1.0,1)) & ((0.8,1),(1.2,1)) \\ ((1.0,1),(0.8,0)) & ((1.0,1),(1.0,0)) & ((1.0,1),(1.2,0)) & ((1.0,1),(0.8,1)) & ((1.0,1),(1.0,1)) & ((1.0,1),(1.2,1)) \\ ((1.2,1),(0.8,0)) & ((1.2,1),(1.0,0)) & ((1.2,1),(1.2,0)) & ((1.2,1),(0.8,1)) & ((1.2,1),(1.0,1)) & ((1.2,1),(1.2,1)) \end{array}\right]$$

[4] Note that $\mathbf{P'}$ is not symmetric, hence no factor of 1/2. Further, the vector $\mathbf{A'}$ has N, not NM unique elements. Therefore, the matrix $\mathbf{B'}$ has $N\cdot NM$ unique elements. Eliminating the NM block diagonal elements in $\mathbf{P'}$ (which are all zero), gives $N^2M - NM = N\cdot(N-1)\cdot M$ evaluations of F.

[5] A multiset is a generalization of a set that allows for duplicate elements [37]. It is formally defined as a 2-tuple (A, m), where A is a set and $m\colon A \to \mathbb{N}_{\geq 1}$ is the *multiplicity*, i.e., $m(a)$ is the number of occurrences of $a \in A$.

$$
\mathbf{B}' = \left[\begin{array}{ccc|ccc}
((1.0,0),(0.8,0)) & ((1.0,0),(1.0,0)) & ((1.0,0),(1.2,0)) & ((1.0,0),(0.8,1)) & ((1.0,0),(1.0,1)) & ((1.0,0),(1.2,1)) \\
((1.0,0),(0.8,0)) & ((1.0,0),(1.0,0)) & ((1.0,0),(1.2,0)) & ((1.0,0),(0.8,1)) & ((1.0,0),(1.0,1)) & ((1.0,0),(1.2,1)) \\
((1.0,0),(0.8,0)) & ((1.0,0),(1.0,0)) & ((1.0,0),(1.2,0)) & ((1.0,0),(0.8,1)) & ((1.0,0),(1.0,1)) & ((1.0,0),(1.2,1)) \\
\hdashline
((1.0,1),(0.8,0)) & ((1.0,1),(1.0,0)) & ((1.0,1),(1.2,0)) & ((1.0,1),(0.8,1)) & ((1.0,1),(1.0,1)) & ((1.0,1),(1.2,1)) \\
((1.0,1),(0.8,0)) & ((1.0,1),(1.0,0)) & ((1.0,1),(1.2,0)) & ((1.0,1),(0.8,1)) & ((1.0,1),(1.0,1)) & ((1.0,1),(1.2,1)) \\
((1.0,1),(0.8,0)) & ((1.0,1),(1.0,0)) & ((1.0,1),(1.2,0)) & ((1.0,1),(0.8,1)) & ((1.0,1),(1.0,1)) & ((1.0,1),(1.2,1))
\end{array}\right]
$$

$$
\begin{aligned}
G(B_{00}) &= G(\,((0.8,0),(0.8,0))\,) = \begin{bmatrix}100 & 100\end{bmatrix}^{\mathrm{T}} = \mathbf{\Theta}^0 \\
G(B_{05}) &= G(\,((0.8,0),(1.2,1))\,) = \begin{bmatrix}80 & 120\end{bmatrix}^{\mathrm{T}} \\
G(B_{50}) &= G(\,((1.2,1),(0.8,0))\,) = \begin{bmatrix}80 & 120\end{bmatrix}^{\mathrm{T}} = G(B_{05}) \\
G(B_{55}) &= G(\,((1.2,1),(1.2,1))\,) = \begin{bmatrix}100 & 100\end{bmatrix}^{\mathrm{T}} = \mathbf{\Theta}^0
\end{aligned}
$$

$$
S(G(B_{00})) = S(\mathbf{\Theta}^0) = \frac{F(\mathbf{\Theta}^0) - F(\mathbf{\Theta}^0)}{F(\mathbf{\Theta}^0)} \cdot 100 = 0
$$

$$
S(G(B_{05})) = S([\,80 \;\; 120\,]^{\mathrm{T}}) = \frac{F\left([\,80 \;\; 120\,]^{\mathrm{T}}\right) - F(\mathbf{\Theta}^0)}{F(\mathbf{\Theta}^0)} \cdot 100
$$

$$
\begin{aligned}
S(G(B_{50})) &= S(G(B_{05})) \\
S(G(B_{55})) &= S(\mathbf{\Theta}^0) = 0
\end{aligned}
$$

$$
\mathbf{P} \equiv H(\mathbf{B})
$$

$$
= \left[\begin{array}{ccc|ccc}
0 & 0 & 0 & S([\,80 \;\; 80\,]^{\mathrm{T}}) & S([\,80 \;\; 100\,]^{\mathrm{T}}) & S([\,80 \;\; 120\,]^{\mathrm{T}}) \\
0 & 0 & 0 & S([\,100 \; 80\,]^{\mathrm{T}}) & 0 & S([\,100 \; 120\,]^{\mathrm{T}}) \\
0 & 0 & 0 & S([\,120 \; 80\,]^{\mathrm{T}}) & S([\,120 \; 100\,]^{\mathrm{T}}) & S([\,120 \; 120\,]^{\mathrm{T}}) \\
\hdashline
S([\,80 \;\; 80\,]^{\mathrm{T}}) & S([\,100 \; 80\,]^{\mathrm{T}}) & S([\,120 \; 80\,]^{\mathrm{T}}) & 0 & 0 & 0 \\
S([\,80 \;\; 100\,]^{\mathrm{T}}) & 0 & S([\,120 \; 100\,]^{\mathrm{T}}) & 0 & 0 & 0 \\
S([\,80 \;\; 120\,]^{\mathrm{T}}) & S([\,100 \; 120\,]^{\mathrm{T}}) & S([\,120 \; 120\,]^{\mathrm{T}}) & 0 & 0 & 0
\end{array}\right]
$$

$$
\mathbf{P}' \equiv H(\mathbf{B}) - H(\mathbf{B}')
$$

$$
= \left[\begin{array}{ccc|ccc}
0 & 0 & 0 & \begin{array}{c} S([\,80 \;\; 80\,]^{\mathrm{T}}) - \\ S([\,100 \; 80\,]^{\mathrm{T}}) \end{array} & S([\,80 \;\; 100\,]^{\mathrm{T}}) & \begin{array}{c} S([\,80 \;\; 120\,]^{\mathrm{T}}) - \\ S([\,100 \; 120\,]^{\mathrm{T}}) \end{array} \\
0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & \begin{array}{c} S([\,120 \; 80\,]^{\mathrm{T}}) - \\ S([\,100 \; 80\,]^{\mathrm{T}}) \end{array} & S([\,120 \; 100\,]^{\mathrm{T}}) & \begin{array}{c} S([\,120 \; 120\,]^{\mathrm{T}}) - \\ S([\,100 \; 120\,]^{\mathrm{T}}) \end{array} \\
\hdashline
\begin{array}{c} S([\,80 \;\; 80\,]^{\mathrm{T}}) - \\ S([\,80 \;\; 100\,]^{\mathrm{T}}) \end{array} & S([\,100 \; 80\,]^{\mathrm{T}}) & \begin{array}{c} S([\,120 \; 80\,]^{\mathrm{T}}) - \\ S([\,120 \; 100\,]^{\mathrm{T}}) \end{array} & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 \\
\begin{array}{c} S([\,80 \;\; 120\,]^{\mathrm{T}}) - \\ S([\,80 \;\; 100\,]^{\mathrm{T}}) \end{array} & S([\,100 \; 120\,]^{\mathrm{T}}) & \begin{array}{c} S([\,120 \; 120\,]^{\mathrm{T}}) - \\ S([\,120 \; 100\,]^{\mathrm{T}}) \end{array} & 0 & 0 & 0
\end{array}\right]
$$

$$
\begin{aligned}
\mathbf{Q}_0 = \Big\{ & 0,\; 0,\; 0,\; S([\,80 \;\; 100\,]^{\mathrm{T}}),\; S([\,120 \; 100\,]^{\mathrm{T}}), \\
& S([\,80 \;\; 80\,]^{\mathrm{T}}) - S([\,100 \; 80\,]^{\mathrm{T}}),\;\; S([\,80 \;\; 120\,]^{\mathrm{T}}) - S([\,100 \; 120\,]^{\mathrm{T}}), \\
& S([\,120 \; 80\,]^{\mathrm{T}}) - S([\,100 \; 80\,]^{\mathrm{T}}),\;\; S([\,120 \; 120\,]^{\mathrm{T}}) - S([\,100 \; 120\,]^{\mathrm{T}}) \Big\}
\end{aligned}
$$

$$\mathbf{Q}_1 = \Big\{0,\ 0,\ 0,\ S([100\ 80\ ]^{\mathrm{T}}),\ S([100\ 120]^{\mathrm{T}}),$$
$$S([80\ \ 80\ ]^{\mathrm{T}}) - S([80\ \ 100]^{\mathrm{T}}),\ \ S([120\ 80\ ]^{\mathrm{T}}) - S([120\ 100]^{\mathrm{T}}),$$
$$S([80\ \ 120]^{\mathrm{T}}) - S([80\ \ 100]^{\mathrm{T}}),\ \ S([120\ 120]^{\mathrm{T}}) - S([120\ 100]^{\mathrm{T}})\Big\}$$

5.1 Cell cycle

The sensitivity of the cell cycle model [6] was assessed in terms of the period of oscillation of active MPF (plotted as a ratio with respect to total cdc2; Supplementary Fig. **S6**). The model has two species that have non-zero initial populations, cyclin and cdc2 (Supplementary Table **S2**). We considered 21 perturbations in the range ±20% around the reference initial populations.[6] This amounts to $2{\cdot}(2-1){\cdot}21^2/2 = 441$ total simulations. The simulations (all run on *Diablo*; Supplementary Table **S1**) took ∼4.2 s using PySB/cupSODA (`sharedconstant` memory mode; 16 threads/block) and ∼9.4 s using SciPy/LSODA. In Supplementary Fig. **S7**A, we plot the pairwise sensitivity matrix $\mathbf{P}$ (Eq. S5). In Supplementary Fig. **S7**B, we plot as boxplots the single-parameter sensitivity multisets $\mathbf{Q}_k$ for both species (Eq. S6). This plot shows that the period of oscillation of active MPF is completely *insensitive* to changes in the initial population of cdc2. The period varies, however, by as much as ±5% with changes in the initial population of cyclin.

5.2 Ras/cAMP/PKA

For the Ras/cAMP/PKA model [18], we use as the sensitivity metric the amplitude of the first peak in the oscillatory time course of cAMP protein (Supplementary Fig. **S8**). The model has 11 species that have non-zero initial populations (Supplementary Table **S3**) and we consider 21 perturbations in the range ±20% around the reference initial populations,[7] amounting to a total of $11{\cdot}(11-1){\cdot}21^2/2 = 24\,255$ simulations (all run on *Diablo*; Supplementary Table **S1**). These took ∼8 min to run using PySB/cupSODA (`sharedconstant` memory mode; 16 threads/block) and ∼65 min using SciPy/LSODA. In Supplementary Fig. **S9**, we plot the pairwise sensitivity matrix $\mathbf{P}$ (Eq. S5) and the single-parameter sensitivity multisets $\mathbf{Q}_k$ for all 11 species (Eq. S6). The latter plot shows that the cAMP first-peak amplitude is most sensitive to changes in the initial populations of ATP, CYR1, and Pde2. Intuitively, these results make sense since ATP is directly involved in cAMP production via the reaction

$$\text{Ras2\_GTP\_CYR1} + \text{ATP} \longrightarrow \text{Ras2\_GTP\_CYR1} + \text{ATP} + \text{cAMP},$$

CYR1 acts one step upstream

$$\text{Ras2\_GTP} + \text{CYR1} \longrightarrow \text{Ras2\_GTP\_CYR1},$$

and Pde2 acts downstream to promote cAMP degradation

$$\text{cAMP} + \text{Pde2} \longleftrightarrow \text{cAMP\_Pde2} \longrightarrow \text{AMP} + \text{Pde2}.$$

Smaller contributions from Cdc25 and Ira2 (Supplementary Fig. **S9**B) also make intuitive sense since both inhibit cAMP production by acting either upstream of CYR1 or on the Ras2_GTP_CYR1 and/or Ras2_GTP complexes (see `ras_camp_pka.py` at github.com/LoLab-VU/PySB_cupSODA_Bioinfo2017). It is interesting to note that the reference initial population values for these species (Supplementary Table **S3**) range from a few hundred (Cdc25, CYR1, Ira2) to a few thousand (Pde2) to tens of millions (ATP).

5.3 EARM

EARM sensitivity was quantified in terms of the "time-to-death," defined as the time at which Smac cleavage reaches 50% (Supplementary Fig. **S10**). The model has 21 species with non-zero initial populations (Supplementary Table **S4**) and we consider 11 perturbations in the range ±20% around the reference initial

[6] $\mathbf{\Delta} = [0.80, 0.82, 0.84, 0.86, 0.88, 0.90, 0.92, 0.94, 0.96, 0.98, 1.00, 1.02, 1.04, 1.06, 1.08, 1.10, 1.12, 1.14, 1.16, 1.18, 1.20]^{\mathrm{T}}$

[7] See footnote 6.

populations,[8] amounting to a total of $21 \cdot (21-1) \cdot 11^2/2 = 25\,410$ simulations (all run on *Diablo*; Supplementary Table **S1**). We considered two different sets of rate parameters, obtained by calibrating the model to experimental data using PSO (see Sec. 3.3), that have comparable goodness-of-fit values. In both cases, the simulations took ∼11 min to run using PySB/cupSODA (`sharedconstant` memory mode; 16 threads/block) and ∼35 min using SciPy/LSODA. Results of the sensitivity analyses for both parameter sets are shown in Supplementary Figs. **S11** and **S12**. We see in both cases large variations in time-to-death with changes in the initial numbers of receptors (R), caspase 8 (C8), Bid, and BAR. However, in Supplementary Fig. **S11** we also see sensitivities to Bak and MCL1. Interestingly, in Supplementary Fig. **S12** we see little sensitivity to Bak and MCL1. Instead, we see sensitivities to Bax, BCLxl, caspase 3 (C3), caspase 6 (C6), ligand (L), and XIAP. These results suggest, therefore, that the model harbors two parallel modes of apoptosis induction, one that is Bax-independent and mediated by Bak, and the other that is Bak-independent and mediated by Bax. This demonstrates the importance of performing sensitivity analyses for different parameter sets when analyzing complex kinetic models such as EARM. Typical calibrations can require millions of simulations and can produce tens- to hundreds-of-thousands of unique parameter sets [38, 39]. Large-scale analyses such as this will clearly require highly-optimized simulation tools that can run on high-performance computing platforms, such as PySB/cupSODA.

6 Bibliography

[1] J. R. Faeder, M. L. Blinov, and W. S. Hlavacek, "Rule-based modeling of biochemical systems with BioNetGen," *Methods Mol. Biol.*, vol. 500, pp. 113–167, 2009.

[2] L. A. Harris, J. S. Hogg, J.-J. Tapia, J. A. P. Sekar, S. Gupta, I. Korsunsky, A. Arora, D. Barua, R. P. Sheehan, and J. R. Faeder, "BioNetGen 2.2: advances in rule-based modeling," *Bioinformatics*, vol. 32, pp. 3366–3368, 2016.

[3] C. F. Lopez, J. L. Muhlich, J. A. Bachman, and P. K. Sorger, "Programming biological models in Python using PySB," *Mol. Syst. Biol.*, vol. 9, p. 646, 2013.

[4] M. S. Nobile, D. Besozzi, P. Cazzaniga, G. Mauri, and D. Pescini, "cupSODA: a CUDA-powered simulator of mass-action kinetics," *Lect. Notes Comput. Sci.*, vol. 7979, pp. 344–357, 2013.

[5] M. S. Nobile, P. Cazzaniga, D. Besozzi, and G. Mauri, "GPU-accelerated simulations of mass-action kinetics models with cupSODA," *J. Supercomput.*, vol. 69, pp. 17–24, 2014.

[6] J. J. Tyson, "Modeling the cell division cycle: cdc2 and cyclin interactions," *Proc. Natl. Acad. Sci. U.S.A.*, vol. 88, pp. 7328–7332, 1991.

[7] M. Dorée and T. Hunt, "From Cdc2 to Cdk1: when did the cell cycle kinase join its cyclin partner?," *J. Cell Sci.*, vol. 115, pp. 2461–2464, 2002.

[8] L. A. Porter and D. J. Donoghue, "Cyclin B1 and CDK1: nuclear localization and upstream regulators," *Prog. Cell Cycle Res.*, vol. 5, pp. 335–347, 2003.

[9] G. J. Den Haese, N. Walworth, A. M. Carr, and K. L. Gould, "The Wee1 protein kinase regulates T14 phosphorylation of fission yeast Cdc2," *Mol. Biol. Cell*, vol. 6, pp. 371–385, 1995.

[10] U. Strausfeld, J. C. Labbé, D. Fesquet, J. C. Cavadore, A. Picard, K. Sadhu, P. Russell, and M. Dorée, "Dephosphorylation and activation of a $p34^{cdc2}$/cyclin B complex *in vitro* by human CDC25 protein," *Nature*, vol. 351, pp. 242–245, 1991.

[11] A. L. Jeong and Y. Yang, "PP2A function toward mitotic kinases and substrates during the cell cycle," *BMB Rep.*, vol. 46, pp. 289–294, 2013.

[12] A. Castro, C. Bernis, S. Vigneron, J.-C. Labbé, and T. Lorca, "The anaphase-promoting complex: a key factor in the regulation of cell cycle," *Oncogene*, vol. 24, pp. 314–325, 2005.

[8] $\mathbf{\Delta} = [\,0.80, 0.84, 0.88, 0.92, 0.96, 1.00, 1.04, 1.08, 1.12, 1.16, 1.20\,]^{\mathrm{T}}$

[13] J. M. Thevelein and J. H. de Winde, "Novel sensing mechanisms and targets for the cAMP-protein kinase A pathway in the yeast *Saccharomyces cerevisiae*," *Mol. Microbiol.*, vol. 33, pp. 904–918, 1999.

[14] L. Wang, G. Renault, H. Garreau, and M. Jacquet, "Stress induces depletion of Cdc25p and decreases the cAMP producing capability in *Saccharomyces cerevisiae*," *Microbiology*, vol. 150, pp. 3383–3391, 2004.

[15] S. Colombo, P. Ma, L. Cauwenberg, J. Winderickx, M. Crauwels, A. Teunissen, D. Nauwelaers, J. H. de Winde, M.-F. Gorwa, D. Colavizza, and J. M. Thevelein, "Involvement of distinct G-proteins, Gpa2 and Ras, in glucose- and intracellular acidification-induced cAMP signalling in the yeast *Saccharomyces cerevisiae*," *EMBO J.*, vol. 17, pp. 3326–3341, 1998.

[16] S. A. Haney and J. R. Broach, "Cdc25p, the guanine nucleotide exchange factor for the Ras proteins of *Saccharomyces cerevisiae*, promotes exchange by stabilizing Ras in a nucleotide-free state," *J. Biol. Chem.*, vol. 269, pp. 16541–16548, 1994.

[17] J. Nikawa, S. Cameron, T. Toda, K. M. Ferguson, and M. Wigler, "Rigorous feedback control of cAMP levels in *Saccharomyces cerevisiae*," *Gene Dev.*, vol. 1, pp. 931–937, 1987.

[18] D. Besozzi, P. Cazzaniga, D. Pescini, G. Mauri, S. Colombo, and E. Martegani, "The role of feedback control mechanisms on the establishment of oscillatory regimes in the Ras/cAMP/PKA pathway in *S. cerevisiae*," *EURASIP J. Bioinform. Syst. Biol.*, vol. 2012, p. 10, 2012.

[19] V. Chelliah, C. Laibe, and N. Le Novère, "BioModels Database: a repository of mathematical models of biological processes," *Methods Mol. Biol.*, vol. 1021, pp. 189–199, 2013.

[20] M. Hucka, A. Finney, H. M. Sauro, H. Bolouri, J. C. Doyle, H. Kitano, A. P. Arkin, B. J. Bornstein, D. Bray, A. Cornish-Bowden, A. A. Cuellar, S. Dronov, E. D. Gilles, M. Ginkel, V. Gor, I. I. Goryanin, W. J. Hedley, T. C. Hodgman, J.-H. Hofmeyr, P. J. Hunter, N. S. Juty, J. L. Kasberger, A. Kremling, U. Kummer, N. Le Novère, L. M. Loew, D. Lucio, P. Mendes, E. Minch, E. D. Mjolsness, Y. Nakayama, M. R. Nelson, P. F. Nielsen, T. Sakurada, J. C. Schaff, B. E. Shapiro, T. S. Shimizu, H. D. Spence, J. Stelling, K. Takahashi, M. Tomita, J. Wagner, and J. Wang, "The systems biology markup language (SBML): a medium for representation and exchange of biochemical network models," *Bioinformatics*, vol. 19, pp. 524–531, 2003.

[21] J. G. Albeck, J. M. Burke, S. L. Spencer, D. A. Lauffenburger, and P. K. Sorger, "Modeling a snap-action, variable-delay switch controlling extrinsic cell death," *PLoS Biol.*, vol. 6, p. e299, 2008.

[22] T. Kuwana, M. R. Mackey, G. Perkins, M. H. Ellisman, M. Latterich, R. Schneiter, D. R. Green, and D. D. Newmeyer, "Bid, Bax, and lipids cooperate to form supramolecular openings in the outer mitochondrial membrane," *Cell*, vol. 111, pp. 331–342, 2002.

[23] M. Certo, V. Del Gaizo Moore, M. Nishino, G. Wei, S. Korsmeyer, S. A. Armstrong, and A. Letai, "Mitochondria primed by death signals determine cellular addiction to antiapoptotic BCL-2 family members," *Cancer Cell*, vol. 9, pp. 351–365, 2006.

[24] R. C. Taylor, S. P. Cullen, and S. J. Martin, "Apoptosis: controlled demolition at the cellular level," *Nat. Rev. Mol. Cell Biol.*, vol. 9, pp. 231–241, 2008.

[25] C. Borner and D. Andrews, "The apoptotic pore on mitochondria: are we breaking through or still stuck?," *Cell Death Differ.*, vol. 21, pp. 187–191, 2014.

[26] A. Shamas-Din, D. Satsoura, O. Khan, W. Zhu, B. Leber, C. Fradin, and D. Andrews, "Multiple partners can kiss-and-run: Bax transfers between multiple membranes and permeabilizes those primed by tBid," *Cell Death Dis.*, vol. 5, p. e1277, 2014.

[27] J. Kennedy, "Particle swarm optimization," in *Encyclopedia of Machine Learning* (C. Sammut and G. I. Webb, eds.), pp. 760–766, Boston, MA: Springer US, 2010.

[28] F.-A. Fortin, F.-M. D. Rainville, M.-A. Gardner, M. Parizeau, and C. Gagné, "DEAP: Evolutionary algorithms made easy," *J. Mach. Learn. Res.*, vol. 13, pp. 2171–2175, 2012.

[29] J. Nickolls, I. Buck, M. Garland, and K. Skadron, "Scalable parallel programming with CUDA," *ACM Queue*, vol. 6, pp. 40–53, 2008.

[30] M. S. Nobile, P. Cazzaniga, A. Tangherloni, and D. Besozzi, "Graphics processing units in bioinformatics, computational biology and systems biology," *Brief. Bioinform. (advance access)*, 2016. doi:10.1093/bib/bbw058.

[31] J. C. Butcher, *Numerical Methods for Ordinary Differential Equations*. John Wiley & Sons, 2003.

[32] L. Petzold, "Automatic selection of methods for solving stiff and nonstiff systems of ordinary differential equations," *SIAM J. Sci. Stat. Comput.*, vol. 4, pp. 136–148, 1983.

[33] N. Wilt, *The CUDA handbook: A comprehensive guide to GPU programming*. Pearson Education, 2013.

[34] *CUDA C Programming Guide*. v7.5 (Sec. 5.3.2); http://docs.nvidia.com/cuda/cuda-c-programming-guide/#axzz4H4I33oqo.

[35] S. Mittal and J. S. Vetter, "A survey of techniques for modeling and improving reliability of computing systems," *IEEE T. Parall. Distr.*, vol. 27, pp. 1226–1238, 2016.

[36] S. Gaudet, S. L. Spencer, W. W. Chen, and P. K. Sorger, "Exploring the contextual sensitivity of factors that determine cell-to-cell variability in receptor-mediated apoptosis," *PLoS Comput. Biol.*, vol. 8, p. e1002482, 2012.

[37] D. E. Knuth, *The Art of Computer Programming, Volume 2 (3rd Ed.): Seminumerical Algorithms*. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc., 1997.

[38] R. N. Gutenkunst, J. J. Waterfall, F. P. Casey, K. S. Brown, C. R. Myers, and J. P. Sethna, "Universally sloppy parameter sensitivities in systems biology models," *PLoS Comput. Biol.*, vol. 3, p. e189, 2007.

[39] H. Eydgahi, W. W. Chen, J. L. Muhlich, D. Vitkup, J. N. Tsitsiklis, and P. K. Sorger, "Properties of cell death models calibrated and compared using Bayesian approaches," *Mol. Syst. Biol.*, vol. 9, p. 644, 2013.

7 Supplementary Tables

Supplementary Table S1: Machines used in this study.

Machine name	**Diablo***	**Mule**†	**Lolab-760**†	**Puma**†
CPU	Intel Xeon E5-2667 v3	Intel Core i7-5930K	Intel Core i7-4820K	Intel Xeon E5-2687W v2
Processor speed (GHz)	3.20	3.50	3.70	3.40
GPU	GeForce GTX 980 Ti	GeForce GTX 970	GeForce GTX 760	Tesla K20c
CUDA cores	2816	1664	1152	2496
Streaming multiprocessors	22	13	6	13
Global memory (MB)	6143	4095	2047	5120
Max clock rate (GHz)	1.08	1.18	1.07	0.71
Max registers per thread	255	255	63	255
Memory bandwidth (GB/s)	336	224	192	208

*Used for all results in Fig. 1 of the main text and Supplementary Figs. **S3** and **S4**.
†Compared against *Diablo* in Supplementary Fig. **S5**.

Supplementary Table S2: Reference initial species populations for the cell cycle model [6]. The PySB version of the model, `tyson_oscillator_in_situ.py`, is available at github.com/LoLab-VU/PySB_cupSODA_Bioinfo2017. `cdc0`: initial population of cdc2; `cyc0`: initial population of cyclin.

Parameter	Reference value
`cdc0`	6022
`cyc0`	1505

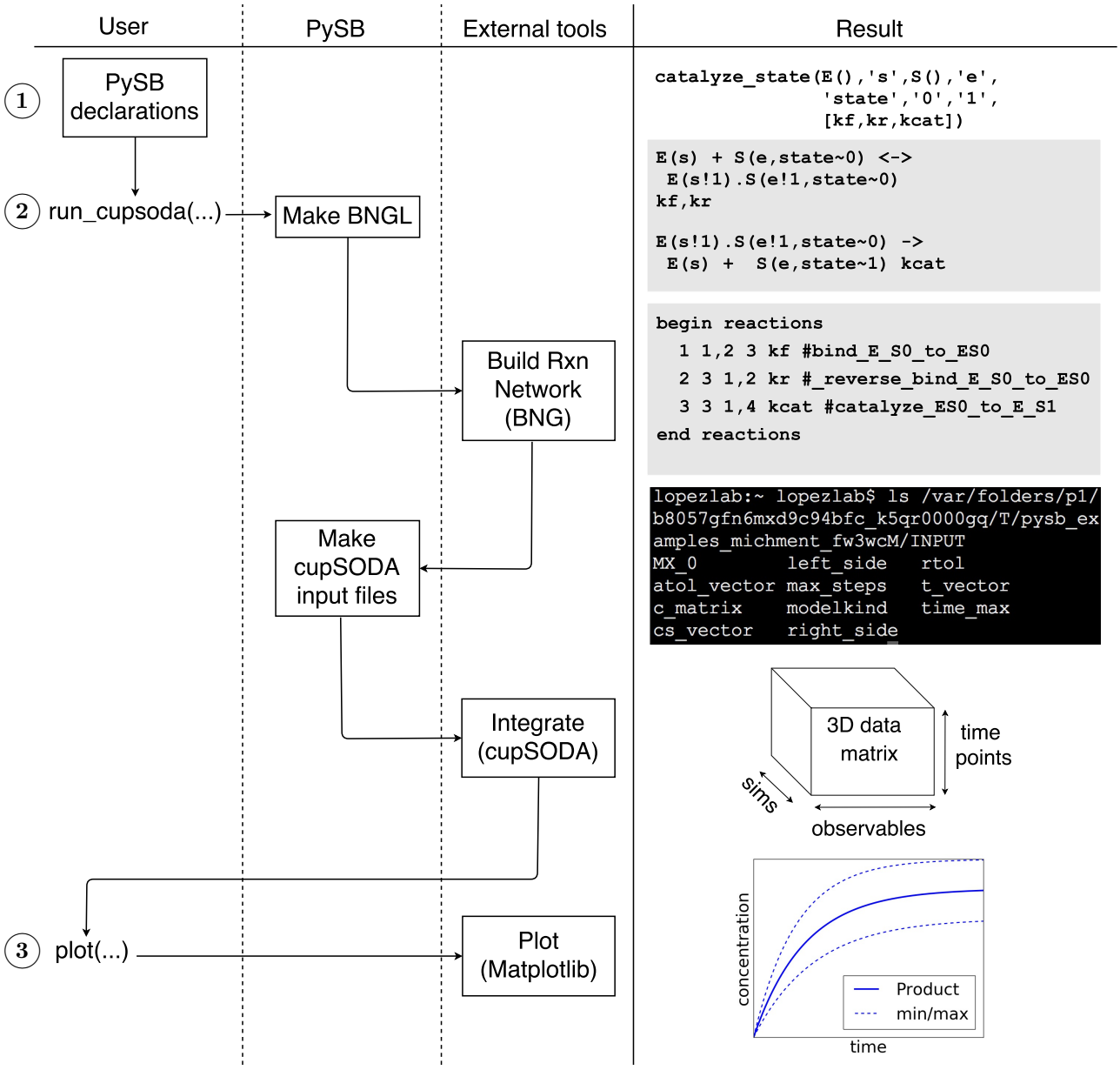
Supplementary Table S3: Reference initial species populations for the Ras/cAMP/PKA model [18]. The PySB version of the model, `ras_camp_pka.py`, is available at github.com/LoLab-VU/PySB_cupSODA_Bioinfo2017.

Parameter	Reference value
ATP_0	24 000 000
Cdc25_0	300
CYR1_0	200
GDP_0	1 500 000
GTP_0	5 000 000
Ira2_0	200
Pde1_0	1400
Pde2_0	6500
PKA_0	2500
PPA2_0	4000
Ras2_GDP_0	20 000

Supplementary Table S4: Reference initial species populations for EARM [3]. The PySB version of the model, `earm_lopez_embedded_flat.py`, is available at github.com/LoLab-VU/PySB_cupSODA_Bioinfo2017.

Parameter	Reference value
Apaf_0	100 000
Bad_0	1000
Bak_0	20 000
BAR_0	1000
Bax_0	80 000
Bcl2_0	20 000
BclxL_0	20 000
Bid_0	40 000
C3_0	10 000
C6_0	10 000
C8_0	20 000
C9_0	100 000
CytoC_0	500 000
flip_0	100
L_0	3000
Mcl1_0	20 000
Noxa_0	1000
PARP_0	1 000 000
R_0	200
Smac_0	100 000
XIAP_0	100 000

8 Supplementary Figures

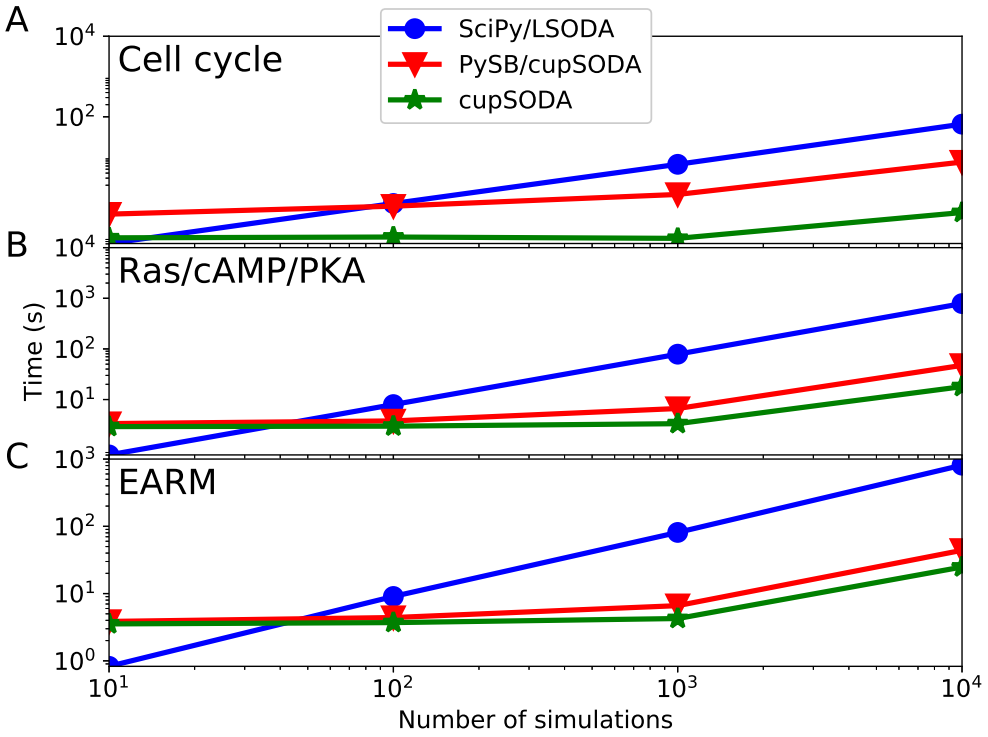


Supplementary Fig. S1: Basic workflow of the PySB/cupSODA interface. The user constructs a model and calls the `run_cupsoda` function with appropriate arguments. PySB translates the model into BNGL and passes the file to BNG to generate the network of reactions and species. PySB reads these in and generates the 11 input files that cupSODA requires. After running the simulations, cupSODA outputs species trajectories into a series of files that PySB reads in and processes into a three-dimensional NumPy record array of "observables" (user-defined model outputs) [1]. Mean, minimum, and maximum concentrations at each time point can then be plotted using the Matplotlib `plot` function. Circled numbers correspond to the associated code blocks in Supplementary Figure **S2**.

```
from pysb.examples.michment import model                                    (1)
from pysb.simulator.cupsoda import run_cupsoda
import numpy as np
import matplotlib.pyplot as plt
import itertools

# factors to multiply the values of the initial conditions                  }
multipliers = np.linspace(0.8, 1.2, 11)                                     }
# 2D array of initial concentrations                                        }
initial_concentrations = [multipliers*ic[1].value for ic in model.initial_conditions]
# Cartesian product of initial concentrations                               }
cartesian_product = itertools.product(*initial_concentrations)              }
# the Cartesian product object must be cast to a list, then to a numpy array }
# and transposed to give a (n_species x n_vals) matrix of initial concentrations
initials_matrix = np.array(list(cartesian_product)).T                       }
# we can now construct the initials dictionary                              }
initials = { ic[0] : initials_matrix[i] for i,ic in enumerate(model.initial_conditions) }  (2)
# simulation time span and output points                                    }
tspan = np.linspace(0, 50, 501)                                             }
# run_cupsoda returns a 3D array of species and observables trajectories    }
trajectories = run_cupsoda(model, tspan, initials=initials,                 }
                           atol=1e-10, rtol=1e-4, verbose=True)             }
# extract the trajectories for the 'Product' into a numpy array and         }
# transpose to aid in plotting                                              }
x = np.array([tr['Product'] for tr in trajectories]).T                      }
# plot the mean, minimum, and maximum concentrations at each time point     }
plt.plot(tspan, x.mean(axis=1), 'b', lw=3, label="Product")                 }
plt.plot(tspan, x.max(axis=1), 'b--', lw=2, label="min/max")                }
plt.plot(tspan, x.min(axis=1), 'b--', lw=2)                                 }
# define the axis labels and legend                                         }
plt.xlabel('time')                                                          }  (3)
plt.ylabel('concentration')                                                 }
plt.legend(loc='upper left')                                                }
# show the plot                                                             }
plt.show()                                                                  }
```

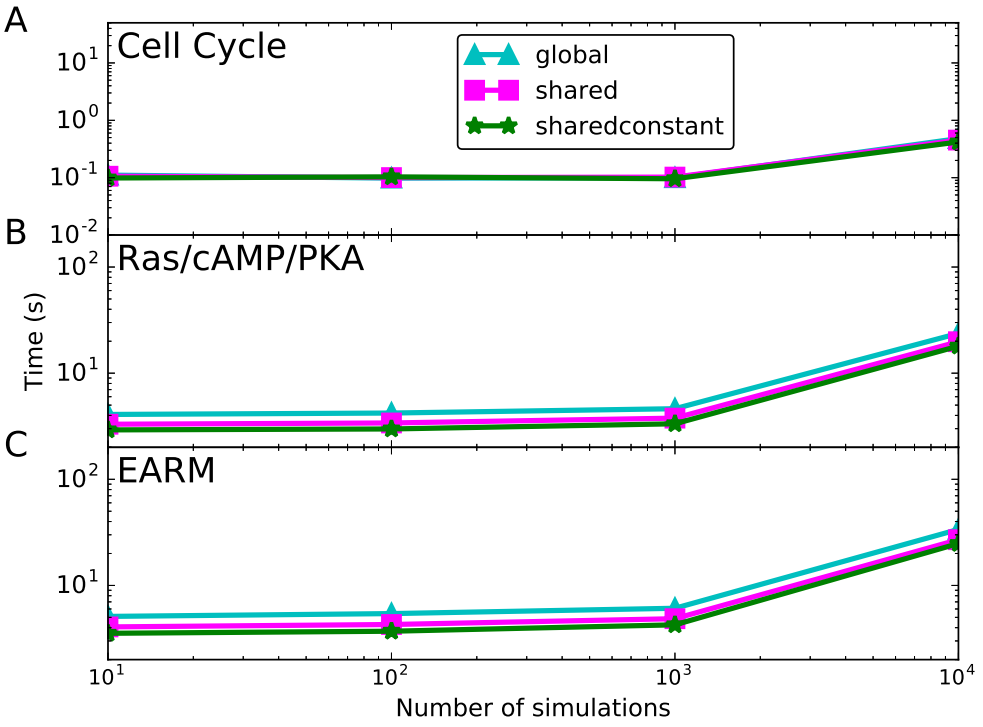
Supplementary Fig. S2: The source code for `run_michment_cupsoda.py` (available at github.com/LoLab-VU/PySB_cupSODA_Bioinfo2017) is composed of three main parts: (1) the model, constructed in `pysb/examples/michment.py`, is imported; (2) a set of initial concentrations is constructed and passed to the `run_cupsoda` function for simulation; (3) mean, minimum, and maximum concentrations at each time point are plotted. Circled numbers correspond to the associated control points in Supplementary Figure **S1**.



D

	# sims	SciPy/ LSODA (s)	PySB/ cupSODA (s)		cupSODA (s)	
Cell cycle	10	0.07	0.38	(0.19×)	0.10	(74.2%)
	100	0.70	0.60	(1.2×)	0.10	(82.9%)
	1000	6.61	1.18	(5.6×)	0.10	(91.2%)
	10 000	65.64	7.52	(8.7×)	0.42	(94.2%)
	100 000	671.78	71.79	(9.4×)	3.87	(94.2%)
Ras/cAMP/ PKA	10	0.81	3.35	(0.2×)	2.92	(12.9%)
	100	7.96	3.78	(2.1×)	2.98	(21.2%)
	1000	78.71	6.65	(11.8×)	3.34	(49.9%)
	10 000	786.10	47.32	(16.6×)	17.98	(62.0%)
	100 000	7804.03	459.75	(17.0×)	169.11	(63.2%)
EARM	10	0.83	3.85	(0.2×)	3.54	(8.1%)
	100	9.06	4.40	(2.0×)	3.70	(15.9%)
	1000	81.15	6.61	(12.3×)	4.25	(35.7%)
	10 000	811.16	43.98	(18.4×)	24.82	(43.6%)

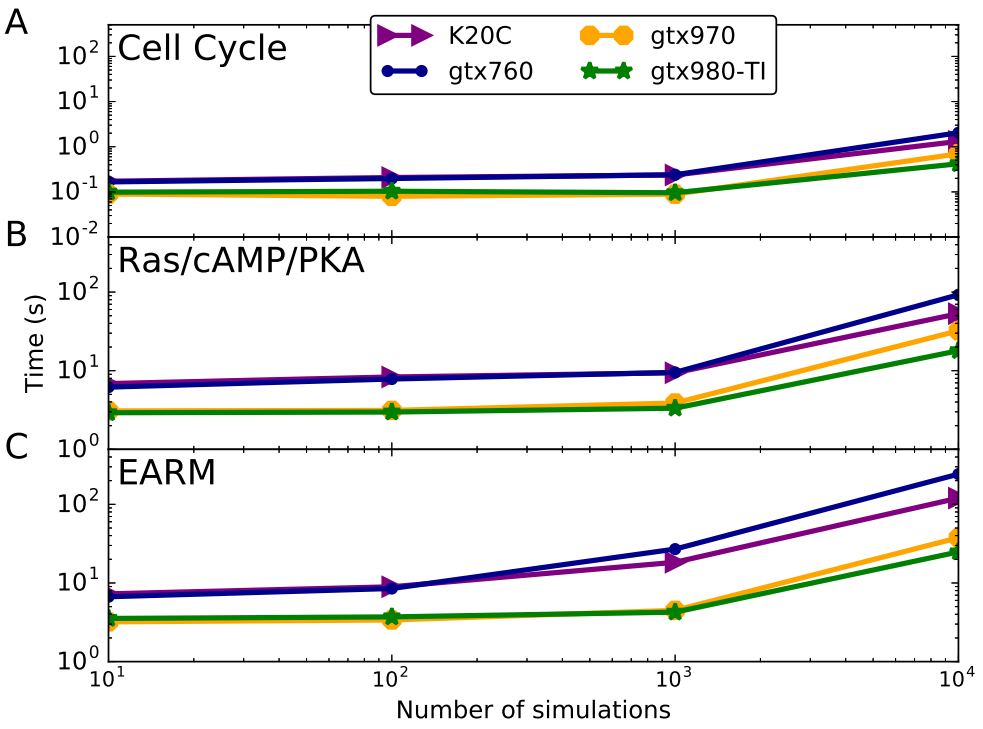
Supplementary Fig. S3: Run time comparisons between LSODA (as implemented in the SciPy scientific computing package; www.scipy.org), PySB/cupSODA, and cupSODA alone (absent the computational overhead associated with the PySB frontend): (A–C) plots for Tyson's cell cycle model [6], the Ras/cAMP/PKA signaling model [18], and EARM [3]; (D) run time values plotted in (A–C). In column 3 of (D), speedups with respect to LSODA are given in parentheses; in column 4, overheads as a percentage of the associated PySB/cupSODA run time are given in parentheses. All simulations were performed on a GeForce GTX 980 Ti GPU with 16 threads/block or on an Intel Xeon E5-2667 v3 CPU (*Diablo*; Supplementary Table **S1**). Note that the "cupSODA" curve is the same as "sharedconstant" in Supplementary Fig. **S4** and "gtx980-TI" in Supplementary Fig. **S5**. The "SciPy" and "PySB/cupSODA" curves are the same as those in Fig. **1** of the main text.



D

	# sims	global (s)	shared (s)	sharedconstant (s)
Cell cycle	10	0.111	0.106 (−4.4%)	0.099 (−6.7%)
	100	0.098	0.101 (−2.2%)	0.103 (2.9%)
	1000	0.100	0.102 (2.6%)	0.096 (−6.7%)
	10 000	0.48	0.45 (−5.8%)	0.42 (−7.2%)
	100 000	4.4	4.023 (−8.6%)	3.87 (−3.9%)
Ras/cAMP/PKA	10	4.08	3.30 (−19.1%)	2.92 (−11.6%)
	100	4.21	3.39 (−19.4%)	2.98 (−12.2%)
	1000	4.62	3.77 (−18.5%)	3.34 (−11.5%)
	10 000	23.70	19.78 (−16.5%)	18.0 (−9.1%)
	100 000	225.23	184.07 (−18.3%)	169.11 (−8.1%)
EARM	10	5.11	4.07 (−20.4.7%)	3.54 (−12.9%)
	100	5.43	4.28 (−21.1%)	3.70 (−13.6%)
	1000	6.09	4.86 (−20.3%)	4.25 (−12.4%)
	10 000	33.71	26.97 (−20.0%)	24.82 (−8.0%)

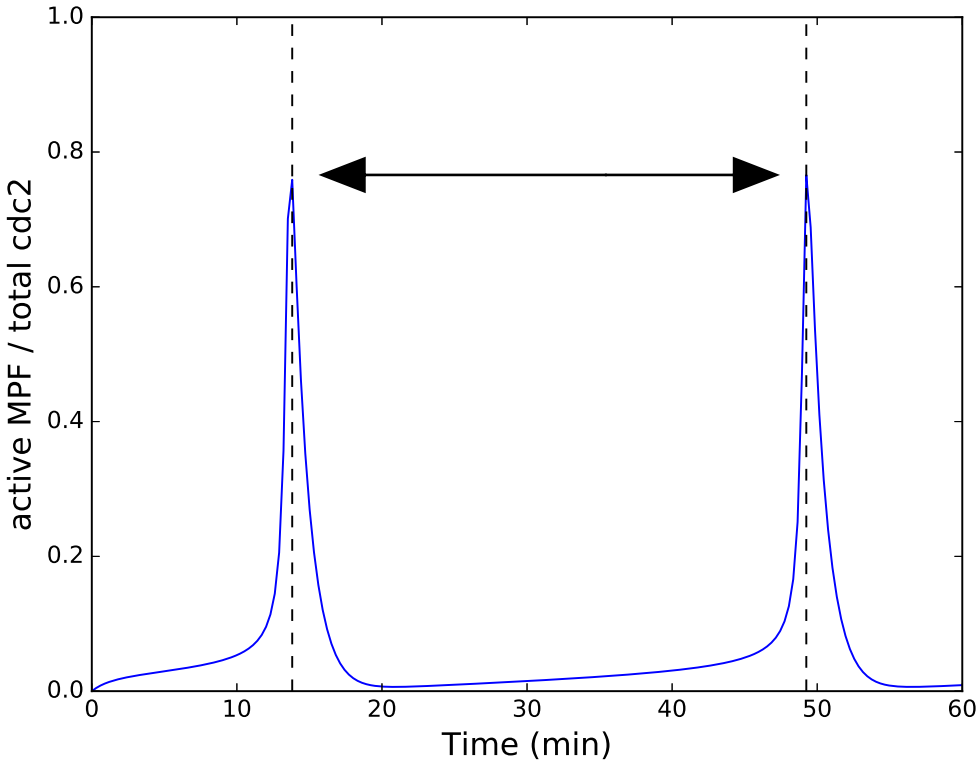
Supplementary Fig. S4: Run time comparisons (raw cupSODA times) between different cupSODA memory schemes: (A–C) plots for Tyson's cell cycle model [6], the Ras/cAMP/PKA signaling model [18], and EARM [3]; (D) run time values plotted in (A–C). In column 3 of (D), run time decreases (as a percentage) relative to the `global` configuration are given in parentheses; in column 4, run time decreases relative to the `shared` configuration are given in parentheses. All simulations were performed on a GeForce GTX 980 Ti GPU (*Diablo*; Supplementary Table **S1**) with 16 threads/block. Note that the "sharedconstant" curve is the same as "cupSODA" in Supplementary Fig. **S3** and "gtx980-TI" in Supplementary Fig. **S5**. Results reported in Fig. **1** of the main text were obtained using the `sharedconstant` configuration.



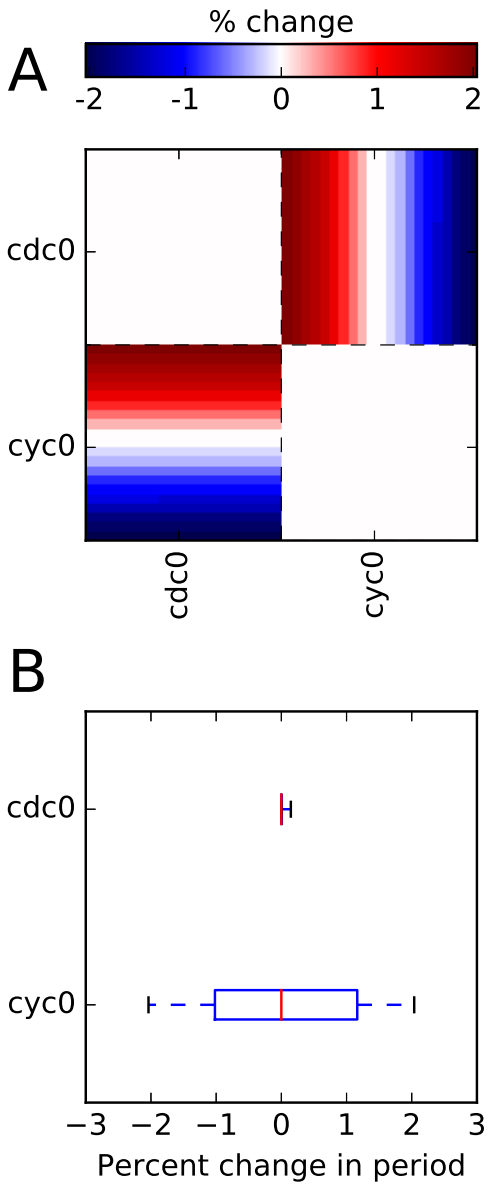
D

	# sims	gtx980-TI (s)	gtx970 (s)	gtx760 (s)	K20C (s)
Cell cycle	10	0.10	0.09	0.17	0.17
	100	0.10	0.08	0.20	0.21
	1000	0.10	0.09	0.24	0.24
	10 000	0.42	0.70	2.05	1.31
	100 000	3.87	6.22	19.38	11.19
Ras/cAMP/PKA	10	2.92	3.08	6.17	6.86
	100	2.98	3.13	7.83	8.32
	1000	3.34	3.88	9.51	9.41
	10 000	17.98	32.19	92.37	53.17
	100 000	169.11	*N/A*	*N/A*	481.18
EARM	10	3.54	3.20	6.65	7.27
	100	3.70	3.38	8.51	8.95
	1000	4.25	4.49	26.95	18.32
	10 000	24.82	37.62	242.06	120.02

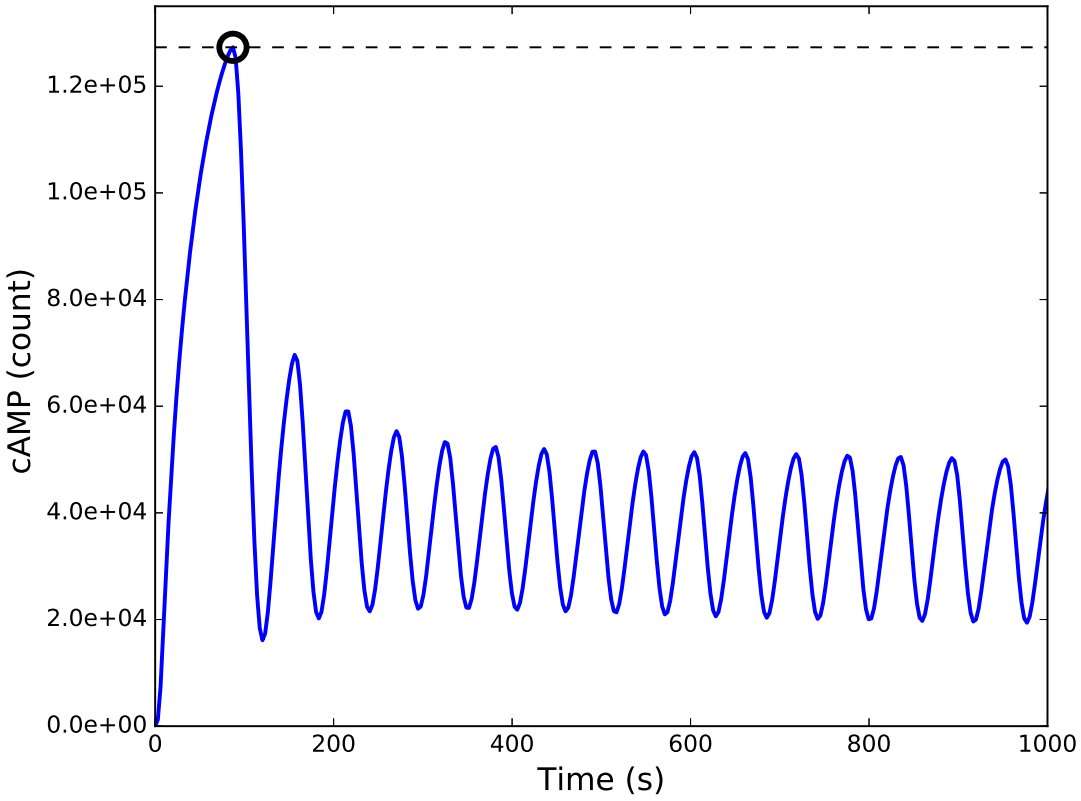
Supplementary Fig. S5: Run time comparisons (raw cupSODA times) between different GPUs (Supplementary Table **S1**): (A–C) plots for Tyson's cell cycle model [6], the Ras/cAMP/PKA signaling model [18], and EARM [3]; (D) run time values plotted in (A–C). All simulations were performed with the `sharedconstant` memory configuration and 16 threads/block. Note that the "gtx980-TI" curve is the same as "cupSODA" in Supplementary Fig. **S3** and "sharedconstant" in Supplementary Fig. **S4**. Note that 100 000 simulations of the Ras/cAMP/PKA model could not be performed on the GeForce GTX 970 or on the GeForce GTX 760 because of global memory limitations.



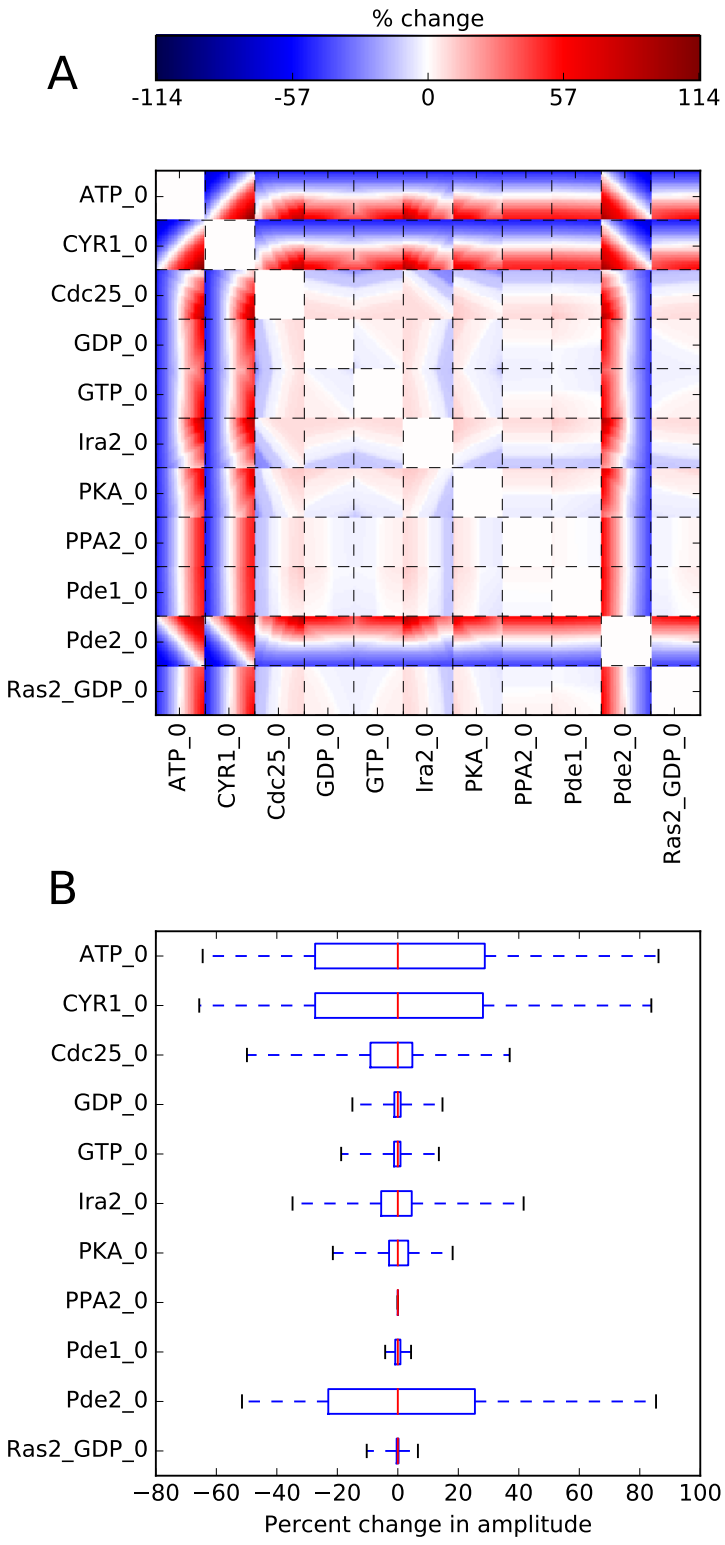
Supplementary Fig. S6: Time course for active maturation-promoting factor (MPF), as a ratio with respect to total cdc2, for the Tyson cell cycle model [6]. Model sensitivity to changes in the initial populations of cyclin and cdc2 was assessed with respect to the period of oscillation of active MPF (double arrow and black dashed lines).



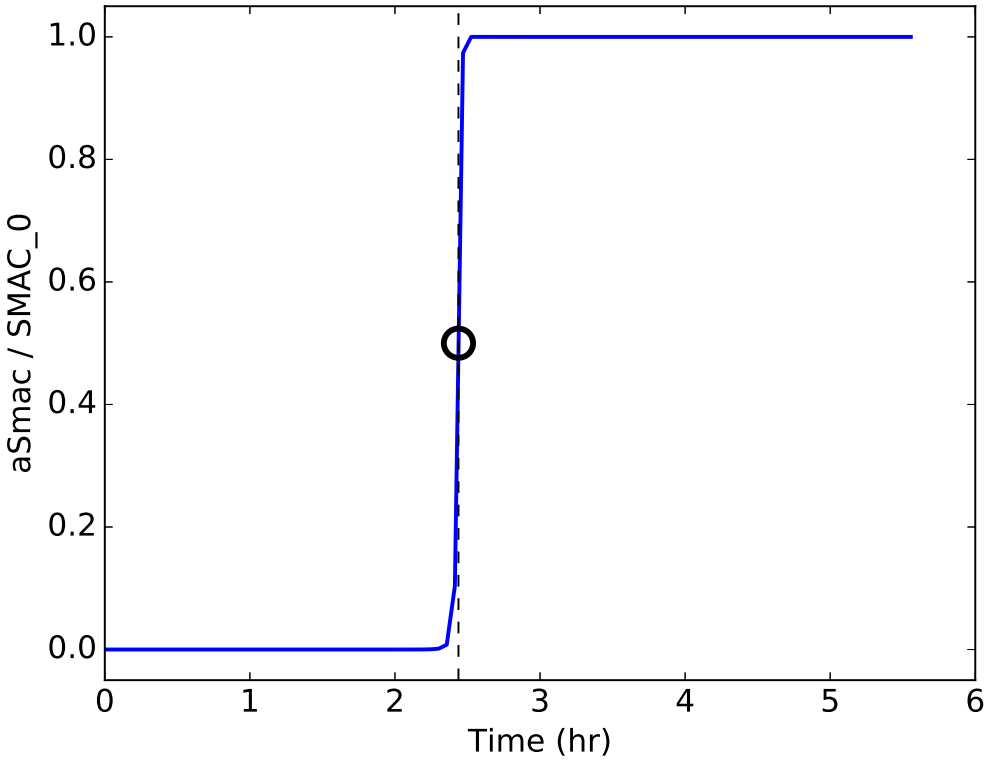
Supplementary Fig. S7: Sensitivity analysis for the Tyson cell cycle model [6]: (A) pairwise sensitivity matrix $\mathbf{P}$ (Eq. S5); (B) single-parameter sensitivity multisets $\mathbf{Q}_k$ (Eq. S6), plotted as boxplots (red lines are medians; boxes range from the first to third quartile; whiskers extend to the minimum and maximum values). `cdc0`: initial population of cdc2; `cyc0`: initial population of cyclin.



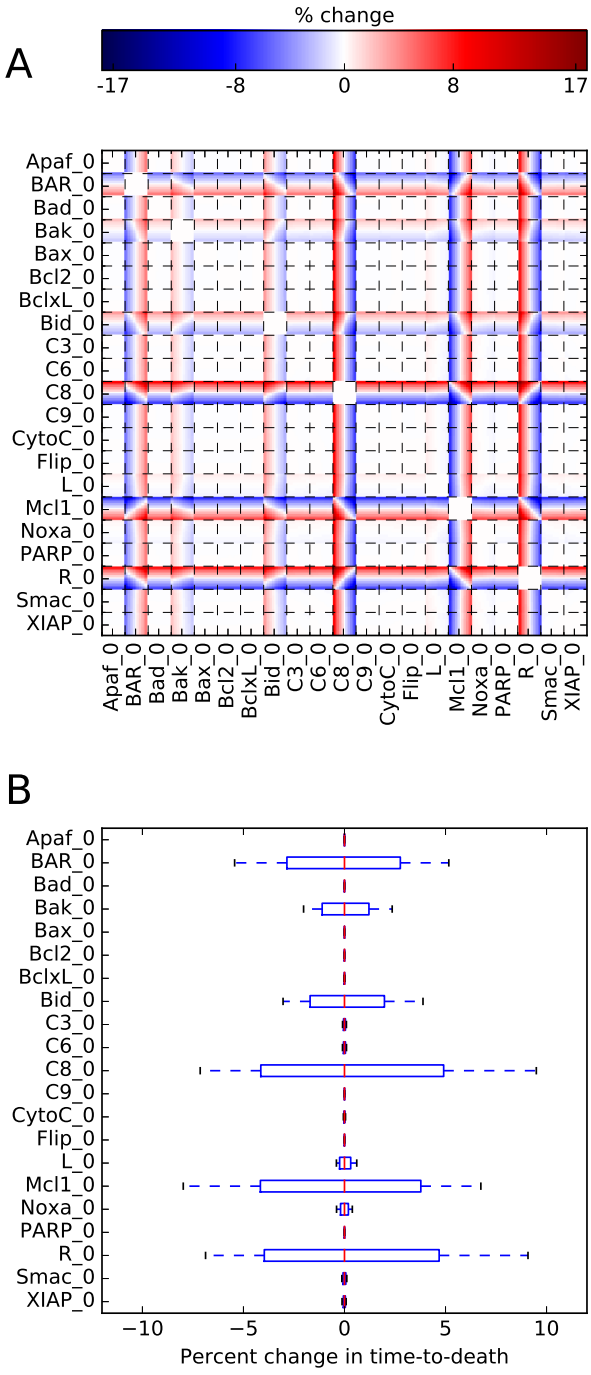
Supplementary Fig. S8: Time course for cAMP in the Ras/cAMP/PKA model [18]. Model sensitivity to changes in initial species populations was assessed with respect to the amplitude of the initial peak (the maximum value achieved; black circle and dashed black line).



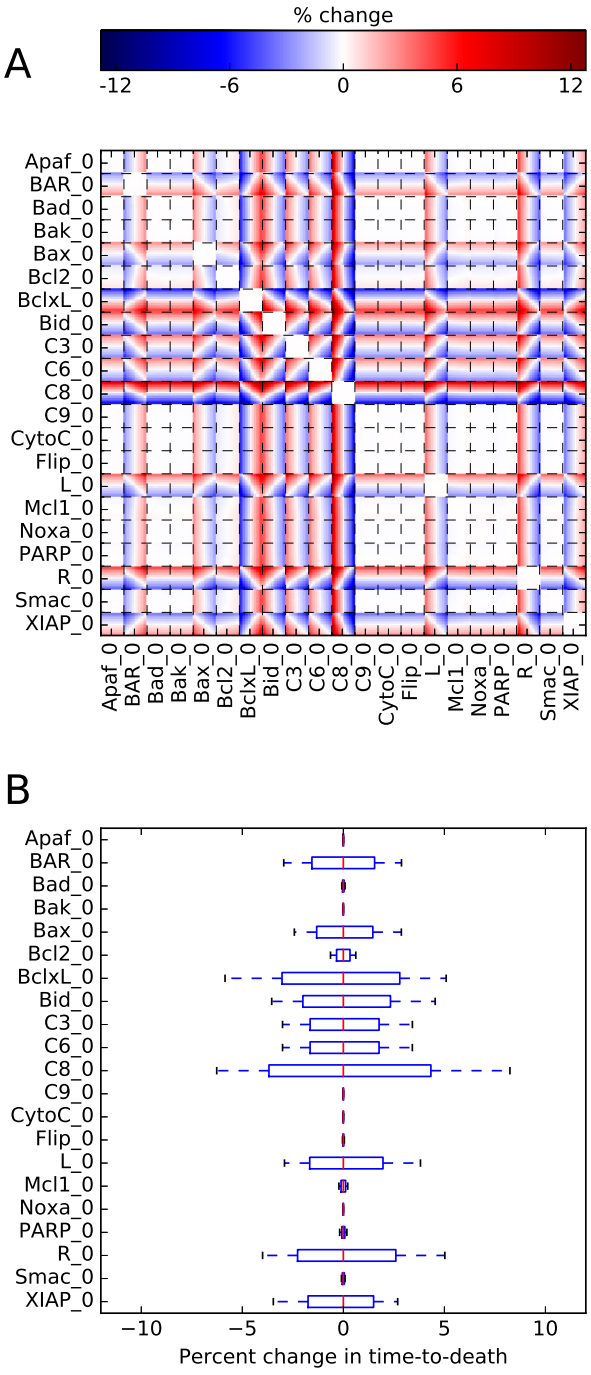
Supplementary Fig. S9: Sensitivity analysis for the Ras/cAMP/PKA model [18]: (A) pairwise sensitivity matrix $\mathbf{P}$ (Eq. S5); (B) single-parameter sensitivity multisets $\mathbf{Q}_k$ (Eq. S6), plotted as boxplots (red lines are medians; boxes range from the first to third quartile; whiskers extend to the minimum and maximum values).



Supplementary Fig. S10: Time course for active Smac (aSmac), as a ratio with respect to total Smac (SMAC_0), in EARM [3]. Model sensitivity to changes in the initial species populations was assessed with respect to the "time-to-death," defined as the time point at which Smac reaches 50% cleavage (aSmac/SMAC_0 = 0.5; black circle and black dashed line).



Supplementary Fig. S11: Sensitivity analysis for parameter set 1 of EARM [3]: (A) pairwise sensitivity matrix $\mathbf{P}$ (Eq. S5); (B) single-parameter sensitivity multisets $\mathbf{Q}_k$ (Eq. S6), plotted as boxplots (red lines are medians; boxes range from the first to third quartile; whiskers extend to the minimum and maximum values). Note that (B) is the same as Fig. **1**D in the main text.



Supplementary Fig. S12: Sensitivity analysis for parameter set 2 of EARM [3]: (A) pairwise sensitivity matrix $\mathbf{P}$ (Eq. S5); (B) single-parameter sensitivity multisets $\mathbf{Q}_k$ (Eq. S6), plotted as boxplots (red lines are medians; boxes range from the first to third quartile; whiskers extend to the minimum and maximum values). Note that this parameter set fits the experimental data comparably to that used in Supplementary Fig. **S11**, yet shows sensitivity to a significantly different set of initial species populations (see Sec. 5.3 for further discussion).