BIOINFORMATICS ORIGINAL PAPER

Vol. 29 no. 10 2013, pages 1292–1298
doi:10.1093/bioinformatics/btt132

Systems biology

Advance Access publication March 16, 2013

Programming strategy for efficient modeling of dynamics in a population of heterogeneous cells

Bjørn Olav Hald[1,*], Morten Garkier Hendriksen[2] and Preben Graae Sørensen[2,*]

[1]Department of Biomedical Sciences, Blegdamsvej 3, 2200 Copenhagen, Denmark and [2]Department of Chemistry, Universitetsparken 5, 2100 Copenhagen, Denmark

Associate Editor: Jonathan Wren

ABSTRACT

Motivation: Heterogeneity is a ubiquitous property of biological systems. Even in a genetically identical population of a single cell type, cell-to-cell differences are observed. Although the functional behavior of a given population is generally robust, the consequences of heterogeneity are fairly unpredictable. In heterogeneous populations, synchronization of events becomes a cardinal problem—particularly for phase coherence in oscillating systems.

Results: The present article presents a novel strategy for construction of large-scale simulation programs of heterogeneous biological entities. The strategy is designed to be tractable, to handle heterogeneity and to handle computational cost issues simultaneously, primarily by writing a generator of the 'model to be simulated'. We apply the strategy to model glycolytic oscillations among thousands of yeast cells coupled through the extracellular medium. The usefulness is illustrated through (i) benchmarking, showing an almost linear relationship between model size and run time, and (ii) analysis of the resulting simulations, showing that contrary to the experimental situation, synchronous oscillations are surprisingly hard to achieve, underpinning the need for tools to study heterogeneity. Thus, we present an efficient strategy to model the biological heterogeneity, neglected by ordinary mean-field models. This tool is well posed to facilitate the elucidation of the physiologically vital problem of synchronization.

Availability: The complete python code is available as Supplementary Information.

Contact: bjornhald@gmail.com or pgs@kiku.dk

Supplementary information: Supplementary data are available at *Bioinformatics* online.

Received on November 13, 2012; revised on February 20, 2013; accepted on March 11, 2013

1 INTRODUCTION

Quantitative modeling of the dynamic behavior of living systems has to be realistic in terms of both biochemistry and biophysics, and in the description of the structural complexity of living matter. Compared with equivalent problems in other parts of natural science, complexity in biological systems is increased because structural elements with similar function, e.g. the cells in a tissue, are not completely identical. Although biological function seems stable and robust, the functional consequences of various kinds of heterogeneity in cellular systems are rather difficult to predict, and are fairly unexplored in the literature. This article describes a new programming strategy aiming to model heterogeneous populations of biological entities. From a modeling perspective, cellular heterogeneity entails a huge increase in the number of parameters and variables, as (i) diversity among cells is introduced by allowing kinetic parameters to vary (even for systems with identical metabolic networks, where kinetic expressions and the species are the same for all cells), and (ii) the chemistry in each individual cells is described through unique sets of corresponding species. Most reactions of a metabolic network are composite reactions where rates may depend on several elementary rate constants and/or stationary concentrations varying from cell to cell.

The programming strategy has been designed to produce code that is computationally effective and at the same time manageable for the programmer. This is accomplished by writing a code generator program, which automatically generates C++ code from descriptions of cell interactions, the metabolic network and the kinetic parameters. The strategy handles spatial as well as temporal models and allows for easy change of scale and of chemical composition.

The strategy was developed to study the dynamics of oscillatory behavior in a population of heterogeneous cells. Notably, usual mean-field models, per definition, neglect the problem of intercellular signaling and synchronization, vital for biological function, which are at the heart of the presented strategy. The modeling is illustrated through the classical example of metabolic oscillations in a stirred suspension of yeast cells first studied by Chance *et al.* (1964). In this system, thousands of modeled cells are coupled by exchange of chemical species through a common extracellular environment. A model of a single cell embedded in a cell solution (Hald and Sørensen, 2010) is used as an illustrative example. The strategy allows for the generation of a large number of cells, each with distinct properties. The overall compartments and reaction network are given in Figure 1. Because the system easily can be scaled up to a very large system [>100.000 ordinary differential equations (ODEs)], we briefly discuss powerful features to speed up computations.

We illustrate the usefulness of the strategy by analyzing and comparing the resulting model simulations using various methods. The yeast system is a particular simple example of an interacting cellular system. Glycolytic oscillations within individual cells give rise to oscillations in the concentrations of extracellular species. If extracellular oscillations can synchronize the individual cells, the whole system will be phase locked in an oscillatory state. This is precisely what is observed in the experimental

*To whom correspondence should be addressed.

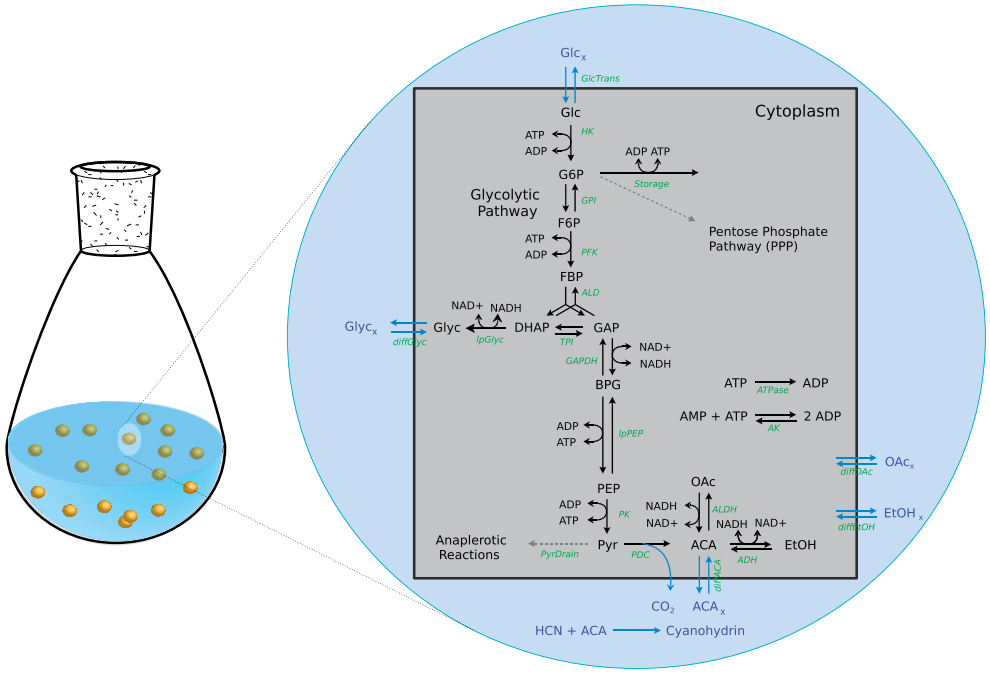

Fig. 1. Drawing of the system and the reaction network. A yeast cell suspension consists of many heterogeneous cells that are coupled through the extracellular medium. In every cell, glucose is used through the glycolytic pathway containing the same enzymes but in different amounts

system. Simulating systems of 1000 cells on the basis of two well-established kinetic models (Hald and Sørensen, 2010; du Preez *et al*., 2012a), however, did not produce in-phase oscillations even with periodic forcing with extracellular acetaldehyde (ACA_x). The results show that realistic kinetic models of a large population of heterogeneous oscillators are sometimes surprisingly hard to synchronize. Thus, it underpins the need for this type of modeling to facilitate understanding of the physiologically vital synchronization process in populations of cellular oscillators.

2 METHODS

The central part of the method is a generator program, which takes *input* (see Fig. 2) in the form of descriptions of kinetic expressions of reactions within and between the different compartments. In the yeast system, a cell and the extracellular medium constitute two *functional units*, where individual cells also constitute individual compartments. Note that cells need not be of the same size nor need they be connected in the same way. The generator program processes the *input* before the generation of a simulator program (here in C++ code), which can then be compiled and executed. The processing stages reduce the risk of introducing errors during manual coding, but the main point is it allows for an efficient reduction in the computational costs of the C++ program owing to an increase in system enlargement.

In this descriptive example, we use three primary techniques to reduce computational load:

(a) To solve the system of ODEs, a Krylov iterative method of CVODE as the linear solver is used. This iteration is an inexact Newton iteration using the current Jacobian, but this is done through the *matrix-free* $\mathbf{J} \times \mathbf{z}$ product, where $\mathbf{J}$ is the Jacobian and $\mathbf{z}$ is an arbitrary vector.

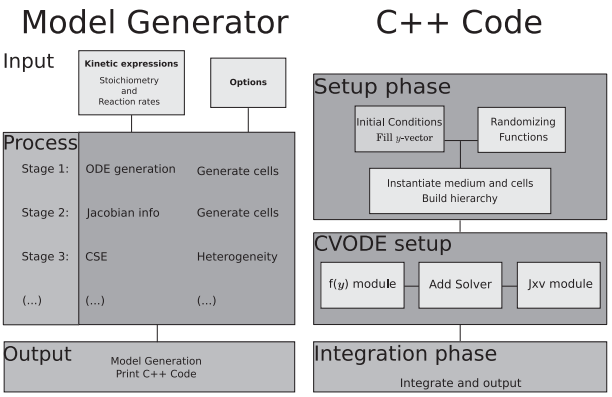

Fig. 2. Overall charts of the strategy

(b) To reduce the sheer number of floating-point arithmetic, elimination of common sub-expressions (CSE) in a list of mathematical expressions is performed.

(c) As each cell constitutes a single compartment, the code should be very amenable to parallelization. This is implemented using OpenMP on a single computer with shared memory.

Introduction of heterogeneity and handling of large-scale systems are facilitated by the use of object-oriented programming (OOP), which is well suited to this task (see the Supplementary Information for a brief description of OOP).

2.1 General layout

Figure 2 shows graphically the overall structure of the proposed strategy. Large-scale modeling requires problem-dependent solutions to reduce

computational costs. Implementation of cost-reducing sub-programs is facilitated by a model generator, which is written in Python. The Python library, SymPy, allows for symbolic manipulations of mathematical expressions. This is used to (i) eliminate common sub-expressions (CSE) within a set of mathematical expressions and (ii) generate symbolic expressions of partial derivatives to produce a function that calculates $\mathbf{J} \times \mathbf{z}$. The model generator works in three stages:

- Input: Compartment descriptions, i.e. models, stoichiometry and initial conditions, are given in turn to the model generator. From this input, the ODEs and stoichiometric matrix are produced.
- Process: Here, cost-reducing or information-enhancing manipulations may be performed on the input.
- Output: As this stage writes out the C++ program, it consists of somewhat messy production code.
- The C++ code is organized as follows (Fig. 2):
  - Setup phase: Initialize variable vector and system objects to be collected in a hierarchical data structure (see below). The data hierarchy is important for introduction of heterogeneity generated at this stage.
  - Initialize solver and output structures.
  - Solve system and produce output.

The generated (yet readable) C++ code is compiled and executed.

The Supplementary Information contains the Python code that is used to implement the strategy and a more technical description of the files. Here, a short presentation of the yeast system will be followed by a section that describes the strategy in a more conceptual frame.

2.2 Yeast system

The modeling of a population of yeast cells is based on a previously published mean-field model of global synchronous glycolytic oscillations within a well-stirred population of yeast cells [refer to Hald and Sørensen (2010) for details]. Experimentally, a global [NADH] oscillation is observed, indicating that the individual cells must be well synchronized (Danø *et al.*, 1999, 2001). Thus, this full-scale model describes in detail the behavior of transient oscillations in a single modeled cell on fermentation of a pulse of glucose and addition of cyanide. This system is modeled by 22 ODEs (17 reactions and 85 parameters, see Supplementary Information). By definition, however, a mean field cannot be used to study the synchronization process. To study the effect of intercellular heterogeneity, many cells have to be simulated each with distinct properties. This is introduced by varying cytosolic volumes and enzymatic activities, i.e. the $V_{\max}$ values of all enzymes within the fermentation network. K_{m} values, etc., can be considered to be constant among monoclonal cells. The parameter values are drawn from a normal distribution, with mean and standard deviation (SD) as given by the literature (any detailed knowledge about parameters in individual cells could also be set explicitly). This is in contrast to stochastic differential equations (SDEs) where noise is introduced explicitly to the ODEs of the system. Here, models of thousands of distinct yeast cell oscillators that couple through a common extracellular medium by linear diffusion processes (see Fig. 1) are used to illustrate the power of the strategy.

2.3 Kinetic expressions and options

Each functional unit, i.e. a yeast cell compartment or the extracellular compartment, is described by a single generic set of ODE expressions. Thus, the two functional units of the system can be described succinctly and similarly through sets of expressions describing reaction kinetics, stoichiometry and initial conditions. Compartments are coupled through metabolites that may cross the plasma membrane. A set of placeholder variables describes these metabolites across every cell in the system.

From the input, the model generator must produce an internal representation of (i) internal and external variables (relative to the compartment), (ii) parameters and (iii) ODEs (as some reaction models are used to in more than one ODEs, individual model expressions are also stored). The representations must be in SymPy format (docs.sympy.org) to allow for symbolic manipulations of the data.

Heterogeneous parameters within cells are given along with the corresponding SD to the model generator. The model generator simply removes these parameters from the global list and treats them separately on writing out the C++ model code (see below).

2.4 Model processing stages

The input data may be manipulated in various ways to decrease run times. Here, we perform the following manipulations: (i) symbolic derivation of all partial derivatives needed to produce the non-zero content of the Jacobian (using sympy.diff), i.e. $\frac{\partial f(\mathbf{y})_i}{\partial y_j}$, where $f(\mathbf{y})_i$ is every ODE and y_j is every variable within $f(\mathbf{y})_i$; and (ii) a search for all common sub-expressions across a list of mathematical expressions (using sympy.cse). CSE on such expressions may significantly decrease calculation load during run time.

2.5 Generation of C++ code

After input processing, the collected information is provided to a set of writer modules that writes out the C++ code. Designing the 'writer modules' is very similar to writing a normal simulation program, e.g. connecting the model with a solver, and will not be described [see e.g. Hindmarsh *et al.* (2005)]. Only features essential to the strategy used in the C++ code are described below.

2.5.1 Initial conditions and variable vector All variables are organized and initialized in a vector, $\mathbf{y}$ (see Fig. 3A): on initializing $\mathbf{y}$, the values of the n_I intracellular variables are stored in a particular sequence (each variable is given an 'enum' to facilitate readability), and the n_I variables of each cell are stored in a back-to-back manner. The end of $\mathbf{y}$ contains the values of the n_X extracellular variables (also in an 'enum'-given sequence). This gives a system size of $N = n_{cells} \cdot n_I + n_X$, where n_{cells} is the number of cells. Each element is therefore easily accessed by formulae. To reduce calculation load during run time, the starting variable index of each cell is stored in the data hierarchy (for the ith cell, c_i: index $= c_i \cdot n_I$). Any variable can then be accessed safely and without copying in C++ code by:

$$\text{const double \& vname} = \text{y[index} + \text{variable\_enum]}$$

This organization is key for (i) optimization of cache coherence, (ii) readability and (iii) fast and safe data retrieval across multiple cells.

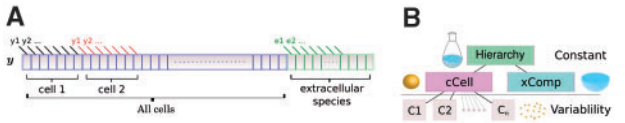

Fig. 3. Variable and parameter organization. **(A)** Variables are stored in a 1D vector, $\mathbf{y}$. Intracellular variables from every cell are stored in a particular sequence, and cells are stored back to back, ending with the extracellular variables. This provides for access by formulae and optimization of cache coherence. **(B)** Parameters are stored in a data hierarchy where system-wide parameters are kept in a hierarchy object at the top-most level, constant parameters for the cell or extracellular compartments in two middle-level objects and the heterogeneous parameter values at the individual cell objects at the bottom level. The four different colors illustrate four different classes

2.6 Data hierarchy

Variable indexes and parameters are also organized to maximize run-time efficiency and readability. Data are stored at relevant levels in a data hierarchy that resembles the morphological hierarchy and chemical structure of the system (see Fig. 3B).

The hierarchy is build such that any parameter is only stored once in the hierarchy but applies to all downstream objects. Thus, a parameter stored in a higher-level object (e.g. the top-most hierarchy object, here simply called *Hierarchy*) applies to all downstream types. In effect, higher-level objects *contain* the lower-level objects as nested lists (we *do not* use an inheritance hierarchy even though the hCell-type 'is a' cCell-type as well). As cells may be heterogeneous, variability is introduced as different parameter values in the multiple objects of hCell type that describe individual cells. This implies that a class is generated for each type of object in the hierarchy, i.e. four classes describe the yeast system (different shades in Fig. 3B).

(i) The top-most *Hierarchy* class contains all 'global' system parameters (none in this example) and single cCell and xComp objects as data members. (ii) The extracellular *xComp* class only contains parameters specific for the extracellular compartment as members. (iii) The 'constant' intracellular *cCell* class contains constant cell parameters as well as a vector of cell objects. (iv) Lastly, data members of the 'heterogeneous' intracellular *hCell* class are parameters to be heterogeneous among the cells as well as the index where the local cell variables are found within the variable vector, **y**. Classes with 'constant' parameters (all except the hCell class) are only instantiated once. In contrast, the hCell class is instantiated n_{cells} times, and each object is given individual values for the parameters to be heterogeneous among cells.

2.6.1 Introduction of heterogeneity In this example, the values of heterogeneous parameters are drawn from a normal distribution, with mean and SD given by the user (see above). The heterogeneous parameters are then given to the hCell object at the point of instantiation. Other mathematical distributions or functions may also be used. These hCell objects are collected in a vector used to instantiate the cCell object. Thus, object instantiation works from bottom-up as we collect lower-level objects into higher-level objects of the data hierarchy.

The $\dot{\mathbf{y}} = f(\mathbf{y})$ *function* is evaluated by traversing the data hierarchy from top to bottom; data from each object are extracted. At the ODE level, all relevant data are available to calculate $\dot{\mathbf{y}}$ (see listing 1 in the Supplementary Information). Note that by accessing parameter values, variables and indexes as references to const doubles or const ints, copying of data is eliminated and computational load is reduced.

The $\mathbf{J} \times \mathbf{z}$ *function*, which returns the matrix-free $\mathbf{J} \times \mathbf{z}$ product, requires proper indexing and non-zero Jacobian information. The derivation of symbolic expressions of partial derivatives reduces this task to an indexing problem, which is expedited by use of the data hierarchy. Again, traversing the data hierarchy provides relevant variable indexing and parameter data (Fig. 3) needed to calculate this vector (see Supplementary Information for technical details).

2.6.2 Parallelization Traversing the data hierarchy in the $f(\mathbf{y})$ and $\mathbf{J} \times \mathbf{z}$ functions makes the code very amenable to parallelization. In this example, we use OpenMP to handle the distribution of calculations among processors. See the python code for an example of implementation. Consult Chapman *et al.* (2007) regarding details.

2.6.3 Storing large datasets The HDF5 file format was used to store large datasets fast and efficiently in a binary format. Moreover, HDF5 offers a number of third-party bindings, including python, Matlab, IDL, etc. See python code for an example of implementation, and refer to the HDF Group (2011) for further information.

2.7 Measuring synchronization

Synchronization between the heterogeneous cells was monitored by probing the long-time distribution of phases, ϕ. The instantaneous phase is derived as the argument of the Hilbert transform, H, of each individual [NADH] oscillation, i.e. $\phi_i(t) = arctan(\frac{H([\mathrm{NADH}_i](t))}{[\mathrm{NADH}_i](t)})$. As a measure of global synchronization at time t, the mean phase distance (*mpd*) between oscillators was calculated as in Gil *et al.* (2009):

$$d_{i,j} = min\{|\phi_i - \phi_j|, 2\pi - |\phi_i - \phi_j|\} \quad (1)$$

$$mpd = \frac{1}{N^2}\sum_{i,j=1}^{N} d_{i,j} \quad (2)$$

3 RESULTS

The main result in this article is the python code that generates the C++ code to be compiled and run (see Supplementary Information). To illustrate the power of the proposed programming strategy, we also include some basic run-time figures.

Our basic test runs included heterogeneity in all V_{max} values of enzymatic reactions (with an SD of 5% relative to the value of V_{max}, see below), as these parameters are heterogeneous among cells. Simulating an experiment of 30 min, corresponding to consumption of 25 mM dissolved glucose, Figure 4 shows that run-time scales almost linearly with system size.

3.1 Enhancing run-time performance

Run-time reductions are, of course, dependent on the particular system and the chosen size. We chose 10 000 cells (16 006 ODEs) as a benchmark system size to illustrate the power of our three basic run-time reduction schemes (i) CSE, (ii) Krylov solver and (iii) parallelization. Table 1 shows bench-marked run times of the

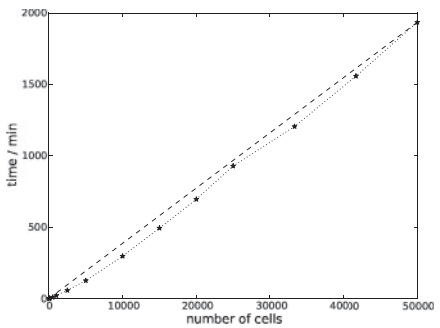

Fig. 4. Run time versus system size. This graph shows the almost proportional relationship between run time and system size

Table 1. Run times in minutes for a 30-min simulation of 10 000 coupled yeast cells with and without system enhancement

Enhancements	All	None	no CSE	One core	Diag[a]
Run time/min	287	1506	311	834	1028
Ratio	1.0	5.25	1.08	2.91	3.58

Normal configuration (All) uses CSE, eight cores and GMRES solver. [a]Diag: linear solver that uses a diagonal approximation to **J**.

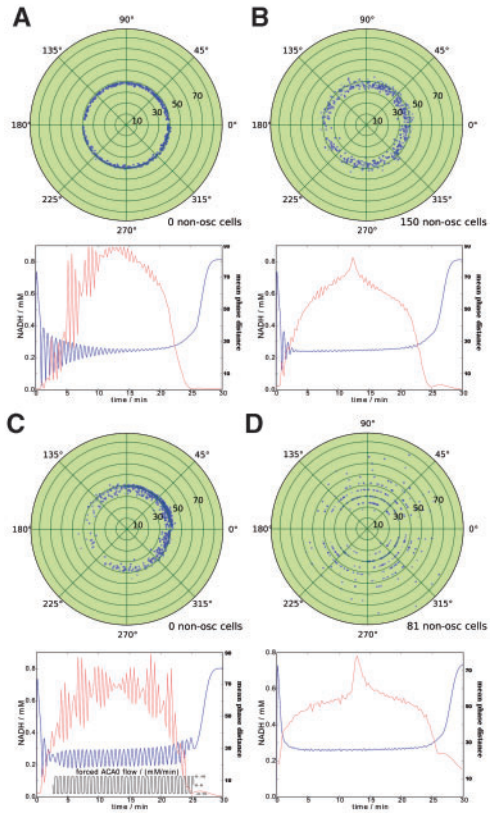

Fig. 5. Heterogeneity desynchronizes oscillations. A total of 1000 cells were simulated. In (**A**), the heterogeneity constant $k = 0.01$ (see text); in (**B** and **C**), $k = 0.05$; and in (**D**), $k = 0.2$. In C, a forcing flow of ACA_x was introduced (see inset). *Upper panel:* Phase-/period-plots of individuals [NADH] oscillations, where each dot has a transparency proportional to the amplitude. Note that the phases are extremely sensitive on heterogeneity, whereas the periods remain more constant (except for D). *Lower panel:* The averaged [NADH] of all cells (blue curves) clearly show that global oscillations diminish with increasing heterogeneity. The mean phase distance (red curve) is high throughout the oscillatory domain. With a forced ACA_x flow (C), global sinusoidal [NADH] oscillations are present, as a small proportion of cells oscillate in-phase, reflected in oscillations of the otherwise high mean phase distance

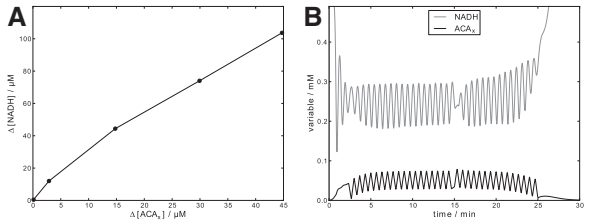

Fig. 6. Glycolytic oscillations depend on $[ACA_x]$ forcing. A total of 1000 cells and $k = 0.05$. (**A**) Increasing forced ACA_x-flow amplitudes from 0.0, 0.02, 0.1, 0.2 to 0.3 $\frac{mM}{min}$ both increased the amplitude of sawtooth-shaped $[ACA_x]$ oscillations, $\Delta[ACA_x]$ (black curve in **B**), and the global amplitude of [NADH] oscillations, Δ[NADH] (grey curve in B), in a near-linear manner. (**B**) Introducing a 180° phase shift in the $[ACA_x]$ forcing lead to a very fast re-entrainment of the simulated [NADH] signal

normal configuration (All), without any optimizations (None) and without just a single optimization.

In this case, excluding CSE only increases run time $\sim$ 10%. In general, this increase will depend on the number of identical sub-expressions found (in some models, up to a factor of 6 can be gained). Excluding parallelization with eight cores increases run time by a factor of $\sim$ 3, and excluding the Krylov-enhanced solver using a Jacobian information yields a factor of $\sim$ 3.5. Excluding all optimizations did not decrease performance proportionally.

3.2 Synchronization

Heterogeneity in the population of yeast cells was simulated by drawing a parameter value, P_{value}, from a normal distribution, with an average equal to the P_{value} in the model from Hald and Sørensen (2010) and an SD set to $k \cdot P_{\text{value}}$, where $0 \leq k \leq 1$. If real-time cytometric data had been available, a more realistic distribution could have been obtained by the method of Hasenauer *et al.* (2011). To keep run times within 20 min, we choose $n_{cells} = 1000$. In simulations, we used $k = 0.01$, 0.05 and 0.2, and observed increasingly low levels of total synchronization as shown in upper and lower panels of Figure 5A, B and D [under similar growth conditions as in our experiments, Johnston *et al.* (1979) estimated up to 30% differences in cell size]. The red curves in the lower panels of Figure 5 show that the mean phase distance between oscillators was high throughout the oscillatory period, and that the rate of desynchronization correlated with level of heterogeneity. Desynchronization is partly a consequence of the lack of phase synchronization between individual oscillators, causing a lack of oscillations in ACA_x, the main synchronizer of the system (see Fig. 6A). Introducing an external oscillatory ACA_x flow to simulate an oscillatory ACA_x flux from synchronized cells led to forced oscillations in global [NADH] oscillations as shown in Figure 6 (no and maximal forcing correspond to Fig. 5B and C, respectively). Even small-amplitude $[ACA_x]$ oscillations produced global [NADH] oscillations, with a roughly linear amplitude relationship (Fig. 6A—note that for $[ACA_x]$ to be non-negative, the amplitude cannot increase much further). Global [NADH] oscillations correlate with a narrowing phase distribution (Fig. 5C and 6). This clearly shows that a heterogeneous population can remain fairly unsynchronized in terms of its phase distribution but still display global oscillations when most of phase angles cluster around the natural global frequency.

As global oscillations are observed experimentally, despite the high degree of heterogeneity, $[ACA_x]$ must indeed oscillate (Richard *et al.*, 1996; Hald and Sørensen, 2010). Usually, a phase-resetting, i.e. a synchronizing, perturbation by cyanide acting on ACA_x is required to achieve global oscillations besides the addition of glucose. A phase-resetting signal was modeled at $t = 5$ min by forcing $[ACA_x]$ for a single period. This, however, did not synchronize the cells (data not shown), corroborating the fact that the intrinsic oscillators of the model mostly are too insensitive to ACA_x. Moreover, a 180° phase shift in the forcing flow of ACA_x led to an instant quench of oscillations followed by swift re-entrainment to the forcing (Fig. 6B). Overall, this indicates that the oscillator sensitivity to ACA_x is too low, and that the forcing mostly entrains the otherwise non-oscillatory or low-amplitude cells to display coherent global oscillations.

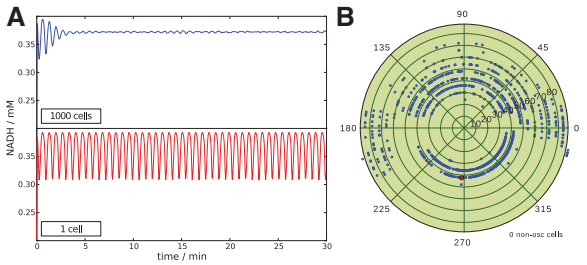

Fig. 7. Cell synchronization within an open-flow system. The modeling strategy was applied to a recently developed open-flow model (du Preez *et al*., 2012a). (**A**) Global [NADH] signal in a system of 1000 heterogeneous ($k = 0.05$) cells (*upper panel*) or a single cell (original model in red color, *lower panel*). (**B**) Phase-/period-plot of the systems in (A): blue dots from *upper panel*, red dot from *lower panel*

This problem is not particular to the metabolic model chosen here. We also applied the modeling strategy to a recent open-flow model (du Preez *et al*., 2012a, b), which also failed to show global synchronous oscillations on introduction of heterogeneity of $k = 0.05$ (see Fig. 7).

It is beyond the scope of this article to elucidate the current lack of our understanding of synchronization in yeast, but the prospects of the proposed modeling strategy for studying the effects of heterogeneity should be evident.

4 DISCUSSION

This article presents a flexible and efficient strategy for generating large-scale models of heterogeneous biological systems from existing single-cell models. A heterogeneous yeast population model was implemented as an example of the concept. The possibility of introducing various forms of heterogeneity at any structural level facilitates numerical studies of the question posed in the introduction: How does biochemical and morphological diversity affect the overall functions and synchronization of a biological system?

The implemented model generator of heterogeneous cell population models was used to study the synchronization processes using previously established models of (i) transient mean-field oscillations in a stirred suspension of yeast cells (Hald and Sørensen, 2010) and (ii) a recent open-flow model of the yeast system (du Preez *et al*., 2012a). This demonstrates the relative ease of switching between models on creation of a semi-generic model generator (the C++ code–generating modules are dependent on the particular problem and have to be revised between very different models). In the former system, we also showed that the phase of individual cellular oscillations is very sensitive to heterogeneity, whereas the frequency is more robust. The lack of in-phase synchronization leads to lack of [ACA_x] oscillations, i.e. the cells do not synchronize in the model as opposed to the global synchronous signal achieved experimentally. Thus, the details of the synchronization process within a yeast cell population are clearly not quantitatively understood.

Owing to incomplete knowledge of the distribution of model parameters for the actual yeast cells, we use a random sample of parameter values to describe the cell heterogeneity. This should not be confused with treating incomplete knowledge of the system environment using SDEs for the simulations (Kampen, 1992). Our particular aim is to investigate synchronization among many heterogeneous oscillators, not the study of random fluctuations in single oscillator in a heterogeneous medium, where the SDE approach would have been appropriate (Bachar *et al*., 2012). Applying an SDE description of the entire ensemble of oscillators would increase the computational load prohibitively. Thus, for the present method, the ODE description is fully retained, and any detailed knowledge of actual morphology and parameter values for the individual cells can be seamlessly implemented.

4.1 Modeling concept

The flexibility and efficiency of the presented strategy mainly stem from the introduction of a model generator: a program that generates code to be compiled instead of writing compilable code directly. This may, at first, seem complicated and time-consuming, but it does have major advantages, as it provides a platform for model processing, customization and readability. Here, the symbolic mathematical library, 'SymPy', was used both to build a matrix-free $\mathbf{J} \times \mathbf{z}$ function needed for the Krylov-based solver and to perform CSE on all lists of mathematical expressions. Moreover, the model generator automatically generates C++ code dependent on selected levels of heterogeneity or custom-made sub-routines (e.g. CSE). Finally, the modularity of the model generator increases code readability and minimizes error introduction, i.e. no system-wide changes are required when changing subsets of the model.

Three code optimizations were performed (Table 1): (i) a Krylov iterative method as solver, which led to an ~4-times reduction in run time. The requirement of the $\mathbf{J} \times \mathbf{z}$ sub-routine requires most work compared with the other optimizations, but it also had the biggest return. (ii) CSE only led to ~10% reduction. However, in other systems with more sub-expressions and reuse, we have observed up to ~6-times reductions in run times (Hald *et al*., 2012). CSE is easily implemented, but may also be the slowest sub-routine of the model generator if many expressions are searched. (iii) Parallelization does not, in itself, require use of a model generator, but the efficiency is enhanced by the data hierarchy of the strategy that allows for easy threading in a shared-memory multiprocessor computer. In the yeast system, parallelization to eight cores led to ~ 3-times reduction in run time. Finally, simulation run times are reduced by letting the generated model trade RAM for speed, as all information that can be calculated before the actual solving of ODEs is stored within the data hierarchy. The efficacy of this approach compared with usual simulator setups is harder to quantify, but the hierarchical structure of our strategy obviously (i) facilitates optimization of cache coherence, (ii) helps optimization of parallelization schemes, (iii) allows for easy data access for the $\dot{\mathbf{y}} = f(\mathbf{y})$ and $\mathbf{J} \times \mathbf{z}$ sub-routines and (iv) improves readability.

The novelty of the proposed strategy is more conceptual than technical: without resorting to a completely new domain-specific language, the existing computational resources are used in the construction of a model generator. The modeling concept is independent of particular implementations or

run-time systems, but different implementations naturally require a rewrite of the presented code. A natural set of future extensions would be to generate code for other ODE integrators and to develop a modeling environment that may generate model code in different formats that can be imported by existing simulation environments. The amount of work required for such extensions depends on the match between the current and new backends.

4.2 Synchronization of heterogeneous oscillators

Introducing heterogeneity in the activities of intracellular enzymes (V_{max}) of a model consisting of 1000 cells led to desynchronization. Increasing levels of heterogeneity showed that the phase of oscillations was particularly sensitive on heterogeneity in V_{max} values as compared with the frequency (Fig. 5). The uniform phase distribution led to a lack of [ACA_x] oscillations, i.e. absence of coupling signal, although most individual cells showed independent oscillations. This finding has recently been demonstrated experimentally using optical tweezers during feeding with a transient glucose pulse (Gustavsson *et al*., 2012). The initial conditions of the cells were identical under each simulation, but differences in the initial transient (during filling of intermediary metabolite pools) might disallow for a synchronization event. However, introducing a 'global' forcing event of [ACA_x] after 5 min, i.e. a phase-resetting signal, did not synchronize cells. Only by external forcing of [ACA_x] during glucose consumption, global [NADH] oscillations could be observed. These arose from forcing the subset of oscillators that showed weak or no oscillations (Fig. 6). This suggests that the glycolytic oscillator sensitivity of real cells on [ACA_x] is larger than in the model, a finding consistent with the experimentally reported [ACA_x] amplitude of only 5–8 μM (Danø *et al*., 2007). We have previously measured 180° phase-shift responses from a yeast cell suspension forced with [ACA_x] at their natural frequency [Fig. 1 in Danø *et al*. (2007)]. These experiments showed a weak amplitude response and fast re-entrainment to the new phase in ~8 cycles owing to a combined phase and amplitude response. The simulation of a 180° phase shift in forced [ACA_x] (Fig. 6) showed an even stronger re-entrainment to the forcing frequency because the system shows no intrinsic [ACA_x] oscillations. Thus, low-amplitude and non-oscillating cells are just driven by the forcing.

As this synchronization problem occurred in two of the most recent quantitative models of yeast synchronization, the modeling strategy highlights a fundamental problem in global synchronization of heterogeneous populations. However, our simulations do show that a complete in-phase synchronization is unnecessary to observe oscillations. As long as the phase distribution is clustered (Fig. 5C), global oscillations might be observed.

The degree of heterogeneity is currently not known in detail. However, it is fair to assume that after the exponential growth phase, cells in all kinds of cell cycle phases are present in the suspension, leaving the cells very heterogeneous in terms of protein content and size.

5 CONCLUSION

A novel strategy for construction of temporal (or spatio-temporal) large-scale models of heterogeneous biological entities has been presented. The strategy was illustrated by modeling a single cell type in a stirred suspension, but coupled cells of different types could be modeled by the same principle. The strategy is particularly well suited to study synchronization processes in heterogeneous cell populations. Large models of coupled cells tend to be stiff and have off-diagonal couplings in their Jacobians. However, the present strategy is flexible enough to allow for system-specific implementations of user-defined processing to enhance run-time performance. Simulations of 1000 yeast cells in two recent models of yeast cell oscillations showed that the sensitivity of the core glycolytic oscillator on [ACA_x] most likely is too weak in the models to allow for globally synchronized oscillations.

Funding: This work was supported by the European Science Foundation FuncDyn Program. B.O.H. was recipient of a PhD fellowship from the Faculty of Health and Medical Sciences, University of Copenhagen.

Conflict of Interest: none declared.

REFERENCES

Bachar,M. *et al*. (2012) *Stochastic Biomathematical Models: with Applications to Neuronal Modeling*. Springer, New York, NY, USA.

Chance,B. *et al*. (1964) Damped sinusoidal oscillations of cytoplasmic reduced pyridine nucleotide in yeast cells. *Proc. Natl Acad. Sci. USA*, **51**, 1244–1251.

Chapman,B. *et al*. (2007) *Using OpenMP: Portable Shared Memory Parallel Programming*. The MIT Press, Cambridge, MA, USA.

Danø,S. *et al*. (1999) Sustained oscillations in living cells. *Nature*, **402**, 320–322.

Danø,S. *et al*. (2001) Synchronization of glycolytic oscillations in a yeast cell population. *Faraday Discuss*., **120**, 261–266.

Danø,S. *et al*. (2007) Quantitative characterization of cell synchronization in yeast. *Proc. Natl Acad. Sci. USA*, **104**, 12732–12736.

du Preez,F.B. *et al*. (2012a) From steady-state to synchronized yeast glycolytic oscillations i: model construction. *FEBS J*., **279**, 2810–2822.

du Preez,F.B. *et al*. (2012b) From steady-state to synchronized yeast glycolytic oscillations ii: model validation. *FEBS J*., **279**, 2823–2836.

Gil,S. *et al*. (2009) Common noise induces clustering in populations of globally coupled oscillators. *Europhys. Lett*., **88**, 60005.

Gustavsson,A.K. *et al*. (2012) Sustained glycolytic oscillations in individual isolated yeast cells. *FEBS J*., **279**, 2837–2847.

Hald,B.O. and Sørensen,P.G. (2010) Modeling diauxic glycolytic oscillations in *yeast*. *Biophys. J*., **99**, 3191–3199.

Hald,B.O. *et al*. (2012) Applicability of cable theory to vascular conducted responses. *Biophys. J*., **102**, 1352–1362.

Hasenauer,J. *et al*. (2011) Identification of models of heterogeneous cell populations from population snapshot data. *BMC Bioinformatics*, **12**, 125.

Hindmarsh,A.C. *et al*. (2005) Sundials: suite of nonlinear and differential/algebraic equation solvers. *ACM Trans. Math. Softw*., **31**, 363–396.

Johnston,G.C. *et al*. (1979) Regulation of cell size in the yeast *Saccharomyces cerevisiae*. *J. Bacteriol*., **137**, 1–5.

Kampen,N.V. (1992) *Stochastic Processes in Physics and Chemistry*. Vol. 1, 2nd edn. North-Holland Personal Library, North Holland.

Richard,P. *et al*. (1996) Acetaldehyde mediates the synchronization of sustained glycolytic oscillations in populations of yeast cells. *Eur. J. Biochem*., **235**, 238–241.

The HDF Group. (2011) *HDF5 User's Guide (v1.8.7)*. www.hdfgroup.org/HDF5/doc/UG (28 March 2013, date last accessed).