MultiGrain/MAPPER: A distributed multiscale computing approach to modeling and simulating gene regulation networks

Mizeranschi, Alexandru E.; Swain, Martin T.; Scona, Raluca; Fazilleau, Quentin; Bosak, Bartosz; Piontek, Tomasz; Kopta, Piotr; Thompson, Paul; Dubitzky, Werner

Published in:
Future Generation Computer Systems

DOI:
10.1016/j.future.2016.04.002

Publication date:
2016

Citation for published version (APA):
Mizeranschi, A. E., Swain, M. T., Scona, R., Fazilleau, Q., Bosak, B., Piontek, T., ... Dubitzky, W. (2016). MultiGrain/MAPPER: A distributed multiscale computing approach to modeling and simulating gene regulation networks. Future Generation Computer Systems, 63, 1-14. https://doi.org/10.1016/j.future.2016.04.002

Accepted Manuscript

MultiGrain/MAPPER: A distributed multiscale computing approach to modeling and simulating gene regulation networks

Alexandru E. Mizeranschi, Martin T. Swain, Raluca Scona, Quentin Fazilleau, Bartosz Bosak, Tomasz Piontek, Piotr Kopta, Paul Thompson, Werner Dubitzky

PII: S0167-739X(16)30071-1
DOI: http://dx.doi.org/10.1016/j.future.2016.04.002
Reference: FUTURE 3001

To appear in: *Future Generation Computer Systems*

Received date: 28 July 2015
Revised date: 15 February 2016
Accepted date: 2 April 2016

Please cite this article as: A.E. Mizeranschi, M.T. Swain, R. Scona, Q. Fazilleau, B. Bosak, T. Piontek, P. Kopta, P. Thompson, W. Dubitzky, MultiGrain/MAPPER: A distributed multiscale computing approach to modeling and simulating gene regulation networks, *Future Generation Computer Systems* (2016), http://dx.doi.org/10.1016/j.future.2016.04.002

***Highlights (for review)**

This paper presents MultiGrain/MAPPER – a novel concept, framework and tool for modeling and simulation based on a multiscale computing paradigm.

MultiGrain/MAPPER has been designed to tackle the computational challenges of large-scale gene-regulatory networks (GRN) model modeling and simulation tasks.

In particular, MultiGrain/MAPPER realizes a distributed computing solution to reverse-engineering of GRN models from gene-expression data.

The solution is based on a distributed multi-swarm (multi-island) particle swarm optimization algorithm we implemented, where PSO islands are mapped to CPU cores.

A detailed evaluation of MultiGrain/MAPPER's concepts and performance is provided in the paper, with a particular emphasis on the tool's computational aspects.

MultiGrain/MAPPER: A Distributed Multiscale Computing Approach to Modeling and Simulating Gene Regulation Networks

Alexandru E. Mizeranschi[a,*], Martin T. Swain[b], Raluca Scona[a], Quentin Fazilleau[a], Bartosz Bosak[c], Tomasz Piontek[c], Piotr Kopta[c], Paul Thompson[a], Werner Dubitzky[a,*]

[a] *University of Ulster, Coleraine, UK*
[b] *Aberystwyth University, Aberystwyth, UK*
[c] *Poznań Supercomputing and Networking Center, Poznań, Poland*

Abstract

Modeling and simulation of gene-regulatory networks (GRNs) has become an important aspect of modern systems biology investigations into mechanisms underlying gene regulation. A key task in this area is the automated inference or *reverse-engineering* of *dynamic mechanistic GRN models* from gene expression time-course data. Besides a lack of suitable data (in particular multi-condition data from the same system), one of the key challenges of this task is the *computational* complexity involved. The more genes in the GRN system and the more parameters a GRN model has, the higher the computational load. The computational challenge is likely to increase substantially in the near future when we tackle larger GRN systems. The goal of this study was to develop a distributed computing framework and system for reverse-engineering of GRN models. We present the resulting software called MultiGrain/MAPPER. This software is based on a new architecture and tools supporting multiscale computing in a distributed computing environment. A key feature of MultiGrain/MAPPER is the realization of GRN reverse-engineering based on the underlying distributed computing framework and multi-swarm particle swarm optimization. We demonstrate some of the features of MultiGrain/MAPPER and evaluate its performance using

*Corresponding authors
Email addresses: alex.mizeranschi@gmail.com (Alexandru E. Mizeranschi), w.dubitzky@ulster.ac.uk (Werner Dubitzky)

both real and artificial gene expression data.

Keywords: Gene-regulatory networks, Reverse-engineering of gene-regulation models, Distributed multiscale computing

1. Introduction

Many complex phenomena occur across multiple spatial and/or temporal scales. Such phenomena are difficult to model and simulate within a single, monolithic approach. Multiscale modeling and simulation adopts a divide-and-conquer philosophy that solves a complex problem by decomposing it into a set of simpler scale-specific sub-models and combining these into a global integrated model, called a multi-scale model [1]. A central modeling task in multiscale modeling is how information specific to one scale-specific sub-model is transformed to another. This information transformation is called *scale-bridging*. A computing challenge in multiscale modeling and simulation is the coupling and coordinated execution of multiple codes representing the sub-models of a multiscale model. We refer to this kind of computing as *multiscale computing*. This article presents a comprehensive computing framework and tool called MultiGrain/MAPPER. This approach has been designed for modeling and simulation of large gene-regulatory networks. The tool was developed as part of the European FP7 project Multiscale Applications on European e-Infrastructures (MAPPER) [2, 3, 4, 5]. The goal of MAPPER was to develop a general framework and technology facilitating the development, deployment and execution of *distributed* multiscale modeling and simulation applications [1, 6].

Based on tools and services developed in the MAPPER project, MultiGrain/MAPPER realizes a gene-regulation model reverse-engineering process based on a multi-swarm *particle swarm optimization* (*PSO*) algorithm. In this approach, the overall particle swarm is partitioned into multiple sub-swarms (which assume the role of sub-models in the MAPPER framework), which are than mapped onto the processor cores of a (potentially distributed) computing resource. The rationale behind this approach is three-fold: First, we expect that the multi-swarm PSO approach is less prone to converge to suboptimal solutions. Second, by casting the reverse-engineering problem into the multiscale modeling and simulation framework of MAPPER, we will be able to develop future versions of MultiGrain/MAPPER which will decompose very large gene-regulation networks into modules of sub-networks

and synthesize these into a coupled multi-network overall solution. And third, by building this approach on the distributed multiscale computing capabilities of MAPPER, we should be able to process large gene-regulatory network problems by taking advantage of computing resources in distributed computing environments.

The current study focuses on the third aspect. In particular, we are interested in how well the computational performance behaves under different problem and computing environment configurations. We have evaluated the performance of MultiGrain/MAPPER based on data from real biological and artificial gene-regulatory networks.

The remainder of this article is organized as follows: Section 2 recapitulates the basic biological aspects of gene-regulatory networks and the main"ingredients" of reverse-engineering gene-regulation models from gene-expression data. In Section 3.1, we review some of the state-of-the-art tools used to address the reverse-engineering task. Section 3 presents our approach, including a description of our multi-swarm PSO algorithm and its implementation in MultiGrain/MAPPER. This is followed by Section 4, in which we describe and discuss our performance evaluation experiments and their results. Finally, Section 5 provides some concluding remarks and a brief look at future developments in this area.

MultiGrain/MAPPER software resources are available from this repository: `https://apps.man.poznan.pl/svn/sbml-toolbox/GRNApplication/`

2. Reverse-engineering GRN models

Cells regulate the expression of their genes to create functional gene products (RNA, proteins) from the information stored in genes (DNA). Gene regulation is a complex process involving the transcription of genetic information from DNA to RNA, the translation of RNA information to make protein, and the post-translational modification of proteins. Gene regulation is essential for life as it allows an organism to respond to changes in the environment by making the required amount of the right type of protein when needed. Complex gene-regulatory processes are coordinated by multiple genes, whose mutual influences are organized as a *gene-regulation network* (*GRN*). Developing *quantitative models of gene regulation* is essential to guide our understanding of complex gene-regulatory processes. For instance, understanding gene-regulatory processes in the context of diseases is increasingly important for the development of treatment and prevention strategies.

Automated reverse-engineering of dynamic mechanistic GRN *models* from gene-expression time-series data is becoming an area of growing interest in systems biology research [7, 8, 9, 10, 11].

Reverse-engineering quantitative dynamic mechanistic GRN models with accurate structure and high predictive performance is a long-standing problem [9]. Currently, some of the main challenges include:

- A lack of sufficient amounts of relevant gene expression time-course data. (1) While the number of sampling points is important (typically, 10 to 50 time points are measured), far more important is to have multiple stimulus-response datasets from the same system under different stimuli [7]. This is a challenging requirement for current experimental practice. (2) GRN stimulus-response data from GRN systems with experimentally confirmed GRN regulatory structure. A good positive example is the study of Cantone and colleagues [12].

- A lack of reverse-engineering algorithms and methods that are able to incorporate existing biological knowledge effectively.

- A lack of algorithms and tools whose computations scale well when the number of genes in the GRN system are increased. Currently, most algorithms and tools are applied to systems involving only 5- to 10 genes.

- Systematic validation studies similar to the work by Cantone et al. [12] that evaluate all aspects (predictive and structure inference aspects) of the algorithm [12].

Reverse-engineering dynamic mechanistic GRN models from gene-expression time-series data involves the following main "ingredients": *data*, *model*, *simulation* and *reverse-engineering algorithm*.

1. **Data**. We reverse-engineer a GRN model from gene-expression *data* that contains measurements over a set of consecutive time points. Thus, a GRN system with n genes corresponds to a dataset with $i = 1, \ldots, n$ time series, each representing the mRNA abundance of gene i at time point $k = 1, \ldots, m$.

2. **Model**. A system model (short: *model*) is a mathematical specification that *represents* the genes of a GRN system and their mechanistic

regulatory relationships in terms of *variables* and *parameters*. A model variable represents the time-*variant* mRNA abundance expressed by a gene, and a model parameter represents time-*invariant* biological and experimental conditions of the GRN system (e.g. the maximal expression rate of a gene).

3. **Simulation**. Based on the initial values of all variables, called the *initial condition*, a system simulation (short: *simulation*) computes the variables' response to the initial condition over a specified time interval.

4. **Algorithm**. We use a *reverse-engineering algorithm* to create (infer) a concrete GRN model from gene-expression data, by iteratively generating candidate models until we have found one that meets certain quality criteria.

The quality of a GRN model depends on two factors: the model's *explanatory power* (or model completeness) and the model's *predictive power* (or model correctness).

A model's explanatory power depends on how well the elements of a mathematical model specification correspond to the salient features of the modeled system. A model's predictive power is estimated by simulating the system's response to the initial condition captured in an independent validation dataset [13]. The greater the deviation (error) between the response time courses predicted by the model and the actual time courses in the validation data, the lower the predictive power of the model. Reverse-engineering of GRN models from data is a highly compute-intensive process, hence a crucial aspect of a reverse-engineering algorithm is the *computational performance* of its implementation.

GRN modeling and simulation software tools implement various features that realize the elements listed above. The diagram in Figure 1 depicts the basic components and "workflow" of a typical GRN model reverse-engineering algorithm. The cloud shape on the left represents the GRN system under study. In this simplified example, the GRN system has $i = 1, 2, 3$ genes corresponding to the model variables x_1, x_2, x_3, respectively. The series of dots labeled *Experimental Data* illustrates the three gene-expression time-series that have been experimentally obtained from the GRN system over 8 consecutive time points. We refer to this as the *training dataset*. The right part of Figure 1 shows the main reverse-engineering loop. The network shape

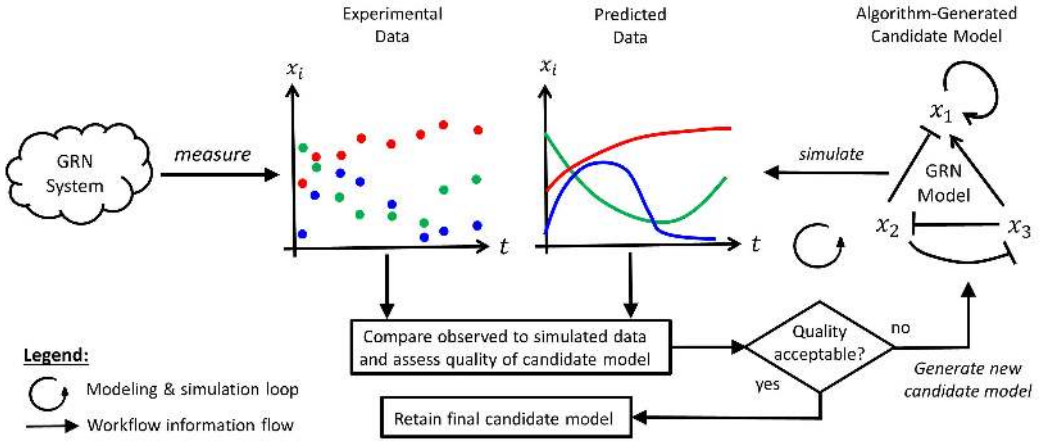


Figure 1: GRN model reverse-engineering workflow. The modeling and simulation loop keeps generating models which predict data until the quality of a candidate is deemed acceptable.

on the right (labeled *GRN Model*) is a graphical depiction of an algorithm-generated concrete candidate model with gene-regulatory interaction links between the three genes of the system. The algorithm simulates the system based on the candidate model and the initial condition of the dataset (arrow labeled *simulate*). The simulation produces a predicted or simulated dataset (curves labeled *Predicted Data*). The experimental and predicted data are then compared (diamond shape) to assess the quality of the candidate model. If the quality is deemed acceptable, the candidate model is retained as the final candidate model. The final candidate model is still subject to validation on independent data (this is not depicted in the diagram).

The simulation of the predicted time series depicted in Figure 1 involves the numeric integration of the model equations. This process and the subsequent quality assessment is carried out many times and, in terms of computation effort, is the most costly part of the reverse-engineering process.

Quantitative dynamic mechanistic reverse-engineering algorithms require a mathematical formulation of the GRN model in the form of a set of coupled *ordinary differential equations* (*ODEs*) [11]. Each equation describes the expression rate of a gene dependent on the activity of all genes in the GRN system. The choice of rate law depends mainly on three factors: First, the ability of the model to represent both gene network *structure* and the *quantitative* mechanistic gene-gene interactions. Second, the ability of the

model to capture highly non-linear variation of gene expression over time. Third, the ability of the model to provide a meaningful biological *interpretation* of the model parameters. Common rate law choices include the *artificial neural network* (*ANN*) [14], *s-system* [15] and the *general rate law of transcription* [16].

Capitalizing on the results of a comparative study by Swain et al. [11], we based our work on the ANN rate law formalism [14] defined by Equation (1).

$$\frac{dx_i}{dt} = \hat{\alpha}_i \frac{1}{1 + exp(\gamma_i - \sum_j^n \omega_{ij}\ x_j)} - \beta_i x_i \tag{1}$$

where

- $x_i, x_j \in \{x_1, x_2, ..., x_n\}$: Time-dependent transcript concentrations of gene i and j, respectively, where n is the total number of genes in the GRN system;
- $dx_i/dt \in \mathbb{R}$: Total rate of change of x_i at time t;
- $\hat{\alpha}_i \in \mathbb{R}^+$: Maximal synthesis rate of transcript x_i;
- $\omega_{ij} \in \mathbb{R}$: Type of synthesis regulation of transcript x_i by x_j, such that

  $\omega_{ij} > 0$: Synthesis activation of x_i by x_j;

  $\omega_{ij} < 0$: Synthesis repression of x_i by x_j;

  $\omega_{ij} = 0$: No synthesis regulation of x_i by x_j;

- $|\omega_{ij}| \in \mathbb{R}_0^+$: Relative weight of synthesis-regulatory influence of x_j on x_i;
- $\gamma_i \in \mathbb{R}$: Activation/repression threshold of x_i synthesis;
- $\beta_i \in \mathbb{R}^+$: Degradation rate constant modulating degradation rate of x_i.

In Equation (1), the model parameter $\hat{\alpha}_i$ represents the *maximal* rate at which x_i can be synthesized. The sum $\sum_j \omega_{ij} x_j$ represents the combined weighted influences of the regulators $x_1, x_2, ..., x_n$ on the synthesis of transcript x_i. The term $1/(1+exp(\cdot))$ maps the regulatory influences onto the unit interval with sigmoidal kinetics. Sigmoidicity (steepness of sigmoid curve) of the kinetics is controlled by the weights in $\sum_j \omega_{ij} x_j$. The larger the absolute values of the weight sum, the steeper the sigmoidicity. High sigmoidicity

means a switch-like behavior of activation (switch-on) and repression (switch-off). Low sigmoidicity represents a "smooth" or "soft" switch. The switch threshold is controlled by the parameter γ_i. In particular, for $\sum_j \omega_{ij} x_j = \gamma_i$ the rate of transcript synthesis is half its maximum value: $\hat{\alpha}_i/2$. For large *positive* values of the combined weighted influences, $\sum_j \omega_{ij} x_j >> \gamma_i$, the synthesis rate approaches $\hat{\alpha}_i$, and for large *negative* values, $\sum_j \omega_{ij} x_j << \gamma_i$, the synthesis rate approaches zero.

3. Approach

3.1. Tools of the trade

We have analyzed state-of-the-art software tools that allow the reverse-engineering of dynamic ODE-based models from time-series data. This analysis enabled us to inform the design of MultiGrain/MAPPER. The list of tools reviewed is not exhaustive, but presents an important set of available tools.

- The Complex Pathway Simulator (COPASI) [17] has become a standard modeling and simulation tool in systems biology.
- Condor-COPASI [18] automates the parallelization and deployment of COPASI simulations on high-performance computers. Condor-COPASI relies on COPASI's [17] ODE solver and optimizations capabilities.
- ByoDyn [19] facilitates the modeling and simulation of biochemical networks. ByoDyn supports SBML and runs on a server (which executes user jobs) that is accessed via a Web-based and a command-line client.
- JSim [20] is a modeling and simulation tool primarily designed for systems biology applications. JSim supports both SBML and CellML exchange formats.
- The Systems Biology Toolbox 2 (SBToolbox2) is a MATLAB toolbox for modeling and simulation in systems biology. SBToolbox2 implements a PSO variant using pattern search for global optimization [21].
- SBML-PET-MPI [22] is a parallel parameter estimation tool for SBML-based models. The tool's parallel computing capability is implemented using the message passing interface (MPI) protocol [23].

- SBMLSimulator [24] is a Java tool allowing users to model and simulate SBML-encoded models. SBMLSimulator relies on the Systems Biology Simulation Core library for its numerical methods [25].

- Virtual Cell (VCell) [26] is an open-source software enabling the modeling and simulation of cellular biological processes. VCell offers a multi-user, distributed environment designed to access resources spread over multiple sites.

- The Systems Biology Software Infrastructure (SBSI) [27] is a software suite that supports model parameter estimation tasks. SBSI emphasizes privacy and security requirements in communications between client and HPC server, while providing a standard HPC-enabled implementation for model parameter estimation.

To address some of the challenges and requirements in modeling and simulation of GRNs, we have developed a tool called MultiGrain/MAPPER. MultiGrain/MAPPER has been developed as part of a more comprehensive project called Multiscale Applications on European e-Infrastructures (MAPPER) [2, 3, 5]. Efficient execution of coupled multiscale simulations is a major challenge in multiscale modeling and simulation. One of the main goals of MAPPER was to address these and other multiscale modeling and simulation challenges by developing a general framework and technology facilitating the development, deployment and execution of *distributed* multiscale modeling and simulation applications [1, 6].

MultiGrain/MAPPER focuses on the *computational* aspects of reverse-engineering GRN models from gene-expression time-series data. Thus, we generally do *not* emphasize the spatiotemporal separation of GRN processes.

3.2. GRN model reverse-engineering algorithm

A central piece in the MultiGrain/MAPPER GRN reverse-engineering algorithm is our multi-swarm PSO approach. Particle swarm optimization is a population-based stochastic optimization technique inspired by flocking or swarming behavior of animals [28]. The main advantage of PSO is that it is ideally suited for optimization problems involving continuous dimensions. A PSO process is initialized with a population (*swarm*) of random solutions or individuals called *particles*. It searches for an optimal solution by iteratively creating new generations of the particle swarm. In each *generation*, the position and velocity of all particles in the population are updated. A

particle's *position* in the solution space represents a concrete solution. A particle moves through the solution space with a specific *velocity*.

The two basic update rules of an enhanced version of PSO [29] are shown in Equations (2a) and (2b).

$$\vec{v}_p(g+1) = \omega\ \vec{v}_p(g) + c_1\ r(\cdot)\ (\vec{y}_p(g) - \vec{x}_p(g)) + c_2\ r(\cdot)\ (\vec{z}_k(g) - \vec{x}_p(g)) \quad (2a)$$

$$\vec{x}_p(g+1) = \vec{x}_p(g) + \vec{v}_p(g+1) \quad (2b)$$

where $\vec{x}_p(g)$ and $\vec{x}_p(g+1)$ denote the *position* of particle p in an n-dimensional space at generations g and $g+1$, respectively. The n dimensions correspond to the n GRN model parameters. $\vec{v}_p(g)$ and $\vec{v}_p(g+1)$ denote the *velocity* of particle p at generation g and $g+1$, respectively. $\vec{y}_p(g)$ denotes the *best personal solution* (position) of particle p until generation g. $\vec{z}_k(g)$ denotes the *best global solution* (position) of any particle k achieved until generation g. ω denotes the *inertia weight* used to balance global and local search [29]. $r(\cdot)$ denotes a function that samples a random number from the unit interval $[0, 1]$. And c_1 and c_2 are positive *learning constants*.

In analogy to genetic algorithms, a *multi-swarm* variant of PSO partitions the particle swarm into multiple sub-swarms, each located on an *island*. In this multi-swarm or multi-island PSO approach, each sub-swarm is allowed to evolve separately for a number of generations before some randomly selected particles are *migrated* between islands. The multi-swarm version is thought to have a better chance to reach a global optimum. MultiGrain/MAPPER implements a multi-swarm PSO algorithm. Another reason for adopting this approach is that it allows us to realize reverse-engineering of GRN models using MAPPER's multiscale computing model which sub-divides a multiscale simulation into a set of coupled sub-simulations. In particular, a MultiGrain/MAPPER PSO island corresponds to a MAPPER sub-model. Furthermore, in the concrete implementation of MultiGrain/MAPPER (see Figure 2), one PSO island is mapped exactly onto one CPU core. We believe that this approach leads to a highly scalable GRN reverse-engineering process and also leaves room for accommodating future modeling and simulation strategies in which complex GRN systems are decomposed into multiple sub-systems.

3.3. MultiGrain/MAPPER software

MultiGrain/MAPPER is a software that implements the multi-swarm PSO algorithm described in Section 3.2. MultiGrain/MAPPER is an application-specific component of a distributed multiscale computing architecture called MAPPER [2, 6, 30, 1]. The MAPPER architecture has been designed to meet the large-scale computing requirements of multiscale modeling and simulation applications across a variety of domains.

Figure 2 depicts the main components of the MultiGrain/MAPPER tool. The boxes labeled *Particle Swarm Optimization* (*PSO*) and *ODE Solver* represent the main components realizing the multi-swarm PSO reverse-engineering algorithm discussed in Section 3.2. The box labeled *Model and Data Repositories* corresponds to tools and components that realize the management of various gene expression datasets (experimental and simulated) and models. Both GRN models and gene expression data can be imported/exported using various standard exchange formats (box labeled *SBML*, *SBRML*). The boxes labeled *MUSCLE* and *QCG Broker* at the bottom of the diagram stand for tools and components that handle the coordinated execution of MultiGrain/MAPPER components in distributed computing environments.

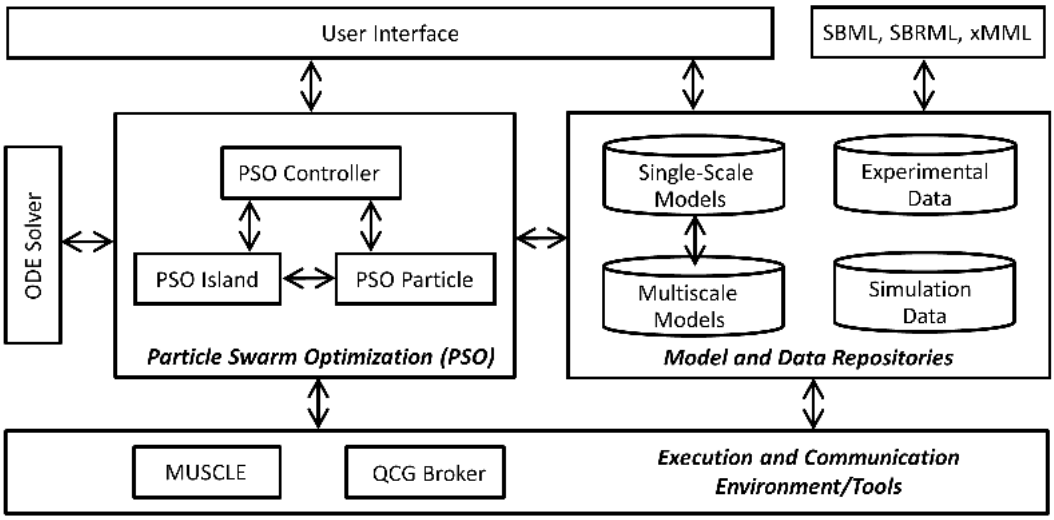


Figure 2: High-level view of MultiGrain/MAPPER software components and their relationship.

The MUSCLE library [30] facilitates the coupled execution of sub-models by enabling communication channels (called *conduits*) between the sub-models.

In MAPPER, MUSCLE is a key component facilitating the coupling of multiple scale-specific sub-models into a multiscale model. In MultiGrain/MAPPER, MUSCLE is the central component facilitating the control and interaction of PSO islands in our implementation of multi-swarm PSO reverse-engineering algorithm. We use MUSCLE, because it is well integrated with other MAPPER components, and because it will play a crucial role in future MultiGrain/MAPPER versions supporting modeling and simulation of coupled/modular GRN systems.

We used the MAPPER-developed Multiscale Modeling Language (MML) and its XML-based implementation called xMML [31, 1] to specify the computational experiments we carried out to evaluate the performance of MultiGrain/MAPPER. MML provides concepts and guidelines for specifying a multiscale model (sub-models and coupling structure of sub-models) and their hardware and software dependencies in a distributed computing environment. MML is supported by a Graphical Multiscale Modeling Language (gMML) – a graphical notation developed to visualize a multiscale model specified in MML.

Both MAPPER and MML support two fundamental forms of coupled multiscale simulations: *acyclic* and *cyclic* coupled simulations. Acyclic coupling entails a sequential execution of the model codes, and cyclic coupling occurs when two sub-models in a multiscale model exchange information (input or output) during simulation of the overall multiscale system. In cyclically coupled multiscale models, there is a need to synchronize executions. Reverse-engineering GRN models with MultiGrain/MAPPER realizes a cyclic coupling topology. In MultiGrain/MAPPER this is implemented based on the following MAPPER tools: QCG Broker, GridSpace Experimental Workbench, MAPPER Memory, and Multiscale Application Designer (Figure 2).

The QCG Broker manages the distributed (cross-cluster) computations of MultiGrain/MAPPER. It is useful for deploying experiments on execution hardware and retrieving the results after the computations are completed. It enables two important features related to resource management: *advance reservation* and *co-allocation* [32]. These features facilitate a reservation of computing resources ahead of the actual use, and the synchronized co-allocation of resources. The QCG Broker also provides functionalities enabling multi-user support and privacy through the encryption of the communication between the client (the user) and the server back-end. GridSpace Experimental Workbench offers a graphical user interface and workflow tools

for specifying and monitoring distributed MultiGrain/MAPPER computations and their results. The MAPPER Memory (MaMe) and the Multiscale Application Designer (MAD) are two MAPPER tools used for storing submodels and building multiscale models (defining their coupling topology) respectively [2]. Multiscale Application Designer is able to export the multiscale models directly into the GridSpace Experimental Workbench using the xMML file format. The ODE solver in MultiGrain/MAPPER is based on Michael Thomas Flanagan's Java Scientific Library [33], and employs the Java version of the libSBML [34] library to read and write SBML models.

3.4. MultiGrain/MAPPER in action

The diagram in Figure 3 illustrates how MultiGrain/MAPPER components interact at run-time. The diagram shows a reverse-engineering process involving 4 PSO islands (rounded boxes) each harboring 9 PSO particles (shown as circle shapes). In this scenario, the 4 islands are distributed over two execution sites (outer boxes labeled *Execution Site/Cluster 1* and *2*) in a distributed computing environment, with 2 PSO islands on each site. Each PSO island is implemented via a MUSCLE *kernel*. MUSCLE kernels are software containers representing various multiscale modeling and simulation elements in a distributed computing environment. The PSO controller component, which coordinates the overall PSO multi-swarm process, is also realized via a MUSCLE kernel. In a distributed MultiGrain/MAPPER process, each site hosts a MUSCLE instance (large box labeled *MUSCLE*). In a cross-cluster scenario, involving more than one site or cluster (e.g. Figure 3), an additional component called MUSCLE Transport Overlay (MTO) is needed to facilitate communication among MUSCLE instances. The MTO works either with non-blocking TCP/IP sockets or with using the MPWide library (which can provide higher message-passing performance over a wide-area network for larger messages). The one-directional arrows in Figure 3 represent MUSCLE *conduits*. Conduits facilitate 1-to-1 communication (or coupling) between MultiGrain/MAPPER components. MUSCLE *mappers* (hexagonal shape in Figure 3) combine the data of multiple sources or extract multiple messages from a single observation. The red arrows in the diagram depict the migration of particles between islands in a ring fashion.

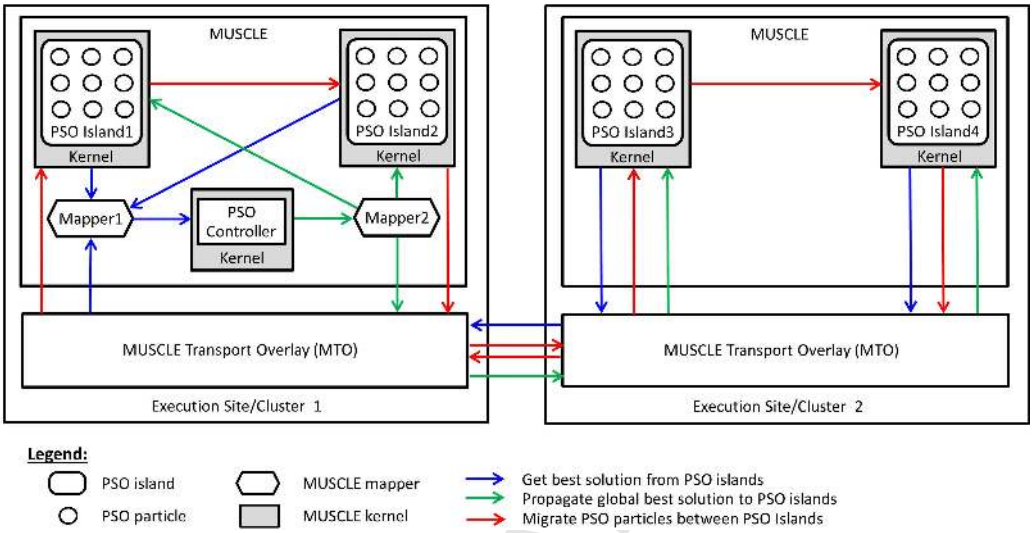


Figure 3: Run-time view of MultiGrain/MAPPER components and their interaction.

4. Evaluation

We performed two sets of experiments to evaluate the effectiveness and efficiency of MultiGrain/MAPPER. The first set of experiments was designed to compare the quality (*effectiveness*) of GRN models reverse-engineered with MultiGrain/MAPPER with those reverse-engineered by related and relevant state-of-the-art tools. We chose the training and validation errors as the main criteria for quantifying the quality of the models generated with these tools. In a second set of experiments we assessed the computational performance (*efficiency*) of MultiGrain/MAPPER in different problem and system configurations. Here, we were particularly interested in how MultiGrain/MAPPER scales with the complexity of the problem (number of genes of the investigated system), the number of PSO islands in our multi-island PSO reverse-engineering algorithm, and with different local and distributed computing configurations. In our experiments we employed data from artificial and real biological (yeast and sea urchin) GRN systems.

4.1. Comparison with state-of-the-art tools

To evaluate how MultiGrain/MAPPER fares in terms of the quality of the reverse-engineered models, we compared MultiGrain/MAPPER with various state-of-the-art tools that implement the automated inference of SBML-encoded GRN models from time-series gene-expression data. The selected

tools vary largely in terms of software and hardware requirements and the level at which processing can be controlled by the user. These constraints make an absolutely fair comparison difficult.

These comparisons are based on the data obtained from 3 GRN systems: 2 real biological systems (yeast and sea urchin organisms), and 1 artificially created GRN system. For each system, we had a training dataset (used to infer a model) and an independent validation dataset (used to validate a model).

Yeast: The data obtained from the yeast GRN system was published by Cantone et al. [12], it consists of 5 genes which interact through 5 synthesis activators and 3 synthesis repressors. From the total pool of data generated by Cantone and colleagues, we selected two datasets containing the averages from five 'switch-on" and four "switch-off" experiments, respectively. Hence, we call the two datasets *switch-on* and *switch-off*, corresponding to a medium shift of the yeast cells from a glucose- to a galactose-containing environment, and vice-versa. We removed the first data points from both datasets as these measurements corresponded to an initial preparatory experimental phase which included the perturbation (the medium shift). The final (edited) datasets that we used contained 15 time points per gene for the switch-on (we used as training data to infer models) and 20 time points per gene for the switch-off dataset (we used to validate the inferred models).

Sea urchin: We extracted the time-series data (48 sampling points) of 11 genes from two gene expression published datasets from sea urchin development studies [35, 36, 37].

Artificial: We employed the Java Artificial Gene Network tool [38] to create an 11-gene artificial GRN system with an average regulatory connectivity of 3. From this system, we generated gene-expression time-series comprising 16 time points under different initial conditions.

We focused our comparison on tools that support the import of SBML-encoded GRN models and feature parameter optimization. The tools vary considerably in the way they process and report errors. To ensure a fair comparison based on training and validation errors, we decided to import all models created by the tools into a separate R code developed in-house. The R code was used to simulate (predict the response to the initial conditions in the training and validation data) the models and to compute the following three errors for each model-dataset combination: the *root mean squared error* (*RMSE*) and two types of *normalized root mean squared error* (*NRMSE*), namely *NRMSE1* and *NRMSE2*. *NRMSE1* divides the *RMSE* by the *mean*

error, and *NRMSE2* divides *RMSE* by the error *range*. The NRMSEs have the advantage that comparison across data of different scales is possible [39].

To account for the inherent stochastic variability in the reverse-engineering process (Eq. (2)), we reverse-engineered 5 GRN models with each tool from the same training dataset and validated each model against the validation dataset. We report the *average* training and validation RMSE, NRMSE1 and NRMSE2 in Table 1. All experiments are based on the ANN GRN model formulation shown in Eq. (1). Our ANN-based GRN models were encoded in SBML format and then imported by the tools. The following tools were subject to this comparison: COPASI, JSim, SBML-PET-MPI, SBMLSimulator and MultiGrain/MAPPER. For various technical reasons, we could not include ByoDyn and SBToolbox2 in the comparison.

The parameters of all tools were set to comparable values where possible. In particular, we used the ANN rate law model formulation defined by Equation (1) for all models and set the training error to near-zero to allow sufficient time for inferring optimal parameter values. To constrain the parameter optimization search space, we set the allowed ANN model parameter intervals in all experiments as follows: maximal transcript synthesis rate $\hat{\alpha}_i \in [0, 10]$; sensitivity $\gamma_i \in [-10, 10]$; transcript degradation rate constant $\beta_i \in [0, 3]$, and transcript synthesis regulation type and strength $\omega_{ij} \in [-10, 10]$. The tool configurations for the optimization part are described below.

MultiGrain/MAPPER: We set the PSO algorithm control parameters in line with the guidelines by Pedersen [40]. The main PSO configuration on MultiGrain/MAPPER was as follows:

- The total PSO swarm of 500 particles was distributed over 10 PSO islands, each hosting 50 particles.
- The maximum number of generations per island was set to 1000. After 25 generations, 3 particles were migrated from one island to the next in a ring topology.
- The PSO learning factors and inertia weight defined by Equation (2) were set as follows: $c_1 = -0.2$, $c_2 = 4.0$ and $\omega = -0.2$.

COPASI: We used "value scaling" as the weight method and set all weights to 1. For the optimization method, we chose a PSO with a population size of 500 particles and a maximum number of 1000 particle generations. We converted the mean squared error reported by COPASI to the RMSE. Our experiments were based on COPASI version 4.14 (Build 89).

JSim: JSim does not implement PSO but a genetic algorithm approach to optimize GRN model parameters. We have set the maximum number of generations to 1000 and the population size to 500.

SBML-PET-MPI: We have configured the tool's stochastic ranking evolution strategy [41] to execute 1000 evolutionary generations. The authors claim that the parallel speedup of their tool is directly proportional to the total number of model parameters and the number of time points in the data. Our experiments are based on SBML-PET-MPI, version 1.2.

SBMLSimulator: We selected the tool's PSO implementation and a grid topology for the particles. We set the swarm size to 500 particles and we allowed a maximum number of 500,000 fitness evaluations, and set the identical learning factor and inertia PSO parameters as in MultiGrain/MAPPER. We selected the Runge-Kutta ODE solver method (the same used by MultiGrain/MAPPER). We used SBMLSimulator, version 1.2.1.

Table 1 shows the training and validation errors of the GRN models that were reverse-engineered with the different tools from three sets of data. The errors in the top three sets of rows represent the average over 5 replicates of the entire reverse-engineering process. The panel of rows at the bottom shows the average across the three organisms. Notice, we did not include the average of the RMSE errors, as it does not make sense to compute the mean for this type of error as it is scale-dependent. The table highlights the lowest error in bold for each type of error (training, validation) and panel.

The main objective of this comparison was to demonstrate that MultiGrain/MAPPER is comparable to state-of-the-art tools in terms of the quality of the reverse-engineered models. We believe that the results reported in Table 1 provide sufficient evidence that this is actually the case. The core focus of this study lies on evaluating the computational performance of MultiGrain/MAPPER in large-scale computing environments. The remaining sections concentrate on this aspect.

4.2. Efficiency in local and distributed scenarios: artificial data

To evaluate the computational performance of MultiGrain/MAPPER, we assessed the the reverse-engineering process in both local and distributed computing setups. Our goal was to determine how the reverse-engineering computations scale dependent on the complexity of the problem (number of genes in the GRN system), the PSO island configuration of our multi-island PSO reverse-engineering algorithm, and the computing environment (clusters, resource management).

Table 1: Comparison of reverse-engineering performance of various state-of-the-art tools.

Tool	Mean Training Error			Mean Validation Error		
	RMSE	**NRMSE1**	**NRMSE2**	**RMSE**	**NRMSE1**	**NRMSE2**
	Yeast System (5 genes)					
COPASI	0.0077	0.4489	0.2398	0.0165	0.8736	0.1613
JSim	0.0131	0.7652	0.4086	0.0477	2.5281	0.4667
SBML-PET-MPI	**0.0055**	**0.3211**	**0.1715**	0.0154	0.8148	0.1504
SBMLSimulator	0.0091	0.5290	0.2825	0.0232	1.2289	0.2269
MultiGrain/MAPPER	0.0064	0.3723	0.1988	**0.0141**	**0.7476**	**0.1380**
	Sea Urchin System (11 genes)					
COPASI	4916.56	2.2204	0.1193	3271.11	1.7427	0.1440
JSim	8904.45	4.0213	0.2160	4842.55	2.5799	0.2132
SBML-PET-MPI	3366.14	1.5202	0.0817	**2316.79**	**1.2343**	**0.1020**
SBMLSimulator	**3254.08**	**1.4696**	**0.0790**	2368.01	1.2616	0.1043
MultiGrain/MAPPER	3589.43	1.6210	0.0871	2629.34	1.4008	0.1158
	Artificial System (11 genes)					
COPASI	2.0850	0.3470	0.1423	2.5714	0.4163	0.1680
JSim	3.7563	0.6252	0.2563	3.7296	0.6038	0.2436
SBML-PET-MPI	6.2134	1.0341	0.4239	5.6354	0.9124	0.3681
SBMLSimulator	1.9710	0.3280	0.1345	2.3349	0.3780	0.1525
MultiGrain/MAPPER	**0.8688**	**0.1446**	**0.0593**	**1.5234**	**0.2467**	**0.0995**
	Average					
COPASI	-	1.0054	0.1671	-	1.0109	0.1578
JSim	-	1.8039	0.2937	-	1.9039	0.3079
SBML-PET-MPI	-	0.9585	0.2257	-	0.9871	0.2068
SBMLSimulator	-	0.7756	0.1653	-	0.9562	0.1612
MultiGrain/MAPPER	-	**0.7127**	**0.1151**	-	**0.7984**	**0.1178**

Reverse-engineering a GRN model from data is a stochastic process that involves the repeated construction *and* quality evaluation (training error) of a potentially very large number of candidate models. In MultiGrain/MAPPER, a PSO particle essentially represents a candidate model (strictly speaking, a PSO particle represents a set of concrete values of the model parameters). Approximately 80% to 85% of the total computation effort in reverse-engineering goes into fitness evaluation of candidate models. The computationally expensive task is the numerical integration of the model equations.

We use the total *number of fitness evaluations* and the *wall time* to benchmark the computational performance of the reverse-engineering process. In our reverse-engineering experiments, each PSO island runs in its own Java virtual machine on a separate CPU core. Therefore, the total number of cores is the same as the number of islands in our multi-island PSO algorithm. Adding islands and cores in this way is also known as weak scaling. How well (relating to efficiency) a reverse-engineering process is handled by MultiGrain/MAPPER software depends on the problem (size parameter space), the availability and allocation of resources (CPU cores), and the communication between the MultiGrain/MAPPER components implementing the algorithm. Based on fitness evaluations and wall time, we use *throughput* as a performance metric:

$$throughput = \frac{evaluations}{time} \tag{3}$$

where *evaluations* refers to the total number of fitness evaluations performed, and *time* denotes the wall time, i.e. the time period from initiation to termination of the entire optimization process.

In particular, we were interested in the computing performance (wall time, throughput) of MultiGrain/MAPPER dependent on the following conditions: (1) The size (number of genes) of the GRN system. (2) The number of PSO islands (and hence CPU cores) of our multi-island PSO algorithm. (3) The protocol used to allocate computing resources. (4) The computing environment: local (single cluster) and distributed (two clusters)

Notice, since we set the training error threshold to near-zero and kept the number of PSO particles/island and the number of particle generations constant, adding a PSO island increases the number of fitness evaluations by a predefined amount. Because the MultiGrain/MAPPER computing model maps a PSO island to exactly one CPU core, adding an island increases

both the computational load (more particles need to processed) as well as the computational resources (one core per island) in a well-defined manner (weak scaling). This allows us to benchmark the system's computational performance. In particular, if our multi-island PSO algorithm implementation scales well, we should see an increase of throughput approximately proportional to the increase of PSO islands. For instance, a doubling of islands should roughly lead to a doubling of throughout.

The first experimental condition was the size (number of genes or gene expression data series) of the target GRN system. We carried out experiments with both artificial and real data, using data consisting of 11, 22 and 33 gene expression time-series, respectively.

To generate artificial gene-expression data, we employed the Java Artificial Gene Network tool [38] to create artificial GRN systems with an average regulatory connectivity of 3. We simulated each artificial GRN system under two different experimental conditions over 16 time points (similar to the Cantone data we used in the tool comparison) to generate two datasets (one for training, one for validation) from each system.

In addition to the artificial GRN systems and data, we used two publicly available gene expression datasets from sea urchin development studies [35, 36, 37]. Each sea urchin dataset contains the gene expression time-course data from 176 genes measured at 48 time points. We used one dataset for model inference, and the other for model validation. From each of the two datasets, we randomly selected the same set of 66 genes and partitioned these into six non-overlapping subsets containing 11 genes each. Based on these data, we constructed four pairs of datasets for model training and validation. Two full sets containing the same 66 genes each, two sets containing the same 33 genes each, two sets containing the same 22 genes each, and two sets containing the same 11 genes each. Thus, we ended up with four training datasets and four validation datasets, three of these pairs matching the size (number of gene expression time series) of the artificial data (11, 22 and 33, respectively). Apart from the different training and validation data and the swarm size (which was fixed to 500 PSO particles divided into 10 PSO islands), the reverse-engineering conditions were identical to those employed for the artificial GRN scenarios (Section 4.2).

For each scenario, we used both a random core assignment and structured core assignment process allocation approaches as described below.

The second experimental condition was the number of PSO islands (i.e. CPU cores). We used eight configurations with a different number of islands,

namely 1, 2, 5, 10, 20, 50, 100 and 200 islands. The particle migration pattern between islands is organized in a ring topology.

The third experimental condition was the type of resource allocation or job scheduling protocol. We repeated all experiments with two protocols: the *random core assignment* (*RCA*) protocol and the *structured core assignment* (*SCA*) protocol. These protocols are implemented by the QCG middleware, that is, they are part of components depicted in the bottom box labeled *Execution and Communication Environment/Tools* in Figure 2. To choose between the two protocols, the modeler must edit a QCG input file, which describes the MultiGrain/MAPPER computing job (the executables, the input and output files, directories, and other required settings). The RCA approach allows the scheduler to automatically allocate each process to any of the available cluster nodes. The main advantage of this protocol is its short resource allocation time. Since the individual processes may run on any core within the cluster, the jobs can be scheduled and started relatively quickly. However, this also results in slower process-process communication due to frequent communication between different nodes of the cluster. The jobs rely more heavily on networked communication instead of the more optimal intra-node communication (through shared memory). In the SCA protocol, the user specifies the structure (topology) in which the cores should be assigned to cluster nodes and CPU cores. This has the advantage of more efficient communication by exploiting cache memory for communication between the processes that are allocated on a single processor/node. However, the SCA protocol suffers from longer waiting times for resources, since the processes have to be located on common nodes. Some time is needed to wait for finishing jobs before the node is free to take new jobs. For the RCA experiments, we were unable to execute 2-cluster scenarios with 200 PSO islands. Due to the large number of processes that needed to be synchronized across the two clusters, the scheduler failed to allocate the resources in a timely manner and the MultiGrain/MAPPER jobs failed to start properly. Similarly, for the SCA scenarios, we experienced large waiting times when using advance reservation (created by the QCG Broker [42]) with complex MultiGrain/MAPPER jobs. In order to ensure proper access to resources, we obtained static reservations (provided by the Inula and Zeus administrators) for 100 cores on each cluster and used these reservations for all the SCA scenarios. However, this restricted the size of the jobs that we could execute – we were unable to perform single-cluster MultiGrain/MAPPER tests with 200 PSO islands using the SCA allocation mode. Furthermore, the Zeus

reservation offered us access to AMD Opteron nodes, but only to 16 cores per node. This limited the size of the process allocations that we could use for the SCA scenarios.

The final condition was the executing environment configuration we used. We executed the MultiGrain/MAPPER reverse-engineering process in three configurations involving two computing sites or clusters called Inula and Zeus. In the first configuration, we allocated all PSO islands on Inula, in the second all islands were allocated on Zeus, and in the third, the islands were allocated equally (where possible) on both sites. The particular hardware details of Inula and Zeus were as follows:

- Inula (cluster 1) was provided by the Poznan Supercomputing and Networking Center (Poznan, Poland). It consists of 68 nodes (24 CPU cores per node, 2.40 GHz) based on an AMD Opteron 6234 architecture with 48-64 GB of memory per node.

- Zeus (cluster 2) was made available by the Cyfronet Academic Computer Centre (Krakow, Poland). Zeus has 1088 Intel nodes consisting of 12 CPU cores per node clocked at 2.26 to 2.66 GHz, and 6267 AMD Opteron nodes with 64 CPU cores per node clocked at 2.30 GHz. The memory per node varies between 16 and 24 GB. Zeus offers a peak performance of 120 teraflops.

Notice, the Zeus specification is considerably more powerful than that of Inula.

To precisely control the computational load (total number of particle evaluations), we set the training error threshold to 0.001. Thus, each reverse-engineering process would exhaust the maximal total number of particle evaluations dictated by the number of PSO islands, the number of particles per island, and the number of PSO generations per island.

Each PSO island was populated with a constant PSO particle swarm size of 50. The total number of PSO generations per island was 1000. In order to increase the chance for a *global* error minimum, we allowed 3 particles to migrate to a neighboring island every 25 generations (resulting in a total of 40 migrations). The migration pattern followed a classical ring topology (particles from the first island migrate to the second, and so on, and particles from the last island migrate to the first). Following the guidelines by Pedersen [40], we determined the values of the remaining PSO parameters empirically.

We also recorded the training and validation errors for all reverse-engineered models as normalized root-mean-square error (NRMSE1, NRMSE2) [39]. Thus, the error figures across the different artificial and real systems should be directly comparable.

To account for the variability in the main benchmark indicators (errors, wall time, throughput) due to the stochastic nature of the algorithm, we performed 5 replicates for each experiment. The results of our computational performance evaluation experiments with the artificial GRN system are summarized in Table 2. Similar experiments we conducted with real biological GRN systems and data are presented in Table 3. The wall time and throughput figures in the tables represent the mean values from the 5 experimental replicates.

Consider Table 2. The table shows the computational performance of MultiGrain/MAPPER, in terms of wall time and throughput, for reverse-engineering the data created from the three artificial GRN systems. A first glance at the table suggests that the SCA protocol leads to higher performance than the RCA protocol. This is likely due to the inherent nature of the two approaches. RCA involves a random assignment of processes to CPU cores on cluster nodes. Because of this, different processes from a single job may end up distributed over various nodes in the cluster and thus computation times may vary significantly from run to run due to the communication overhead. This is due to the network interface linking the cluster nodes; this type of communication is slower compared to the shared memory communication between processor cores within the same node. This kind of communication overhead typically leads to longer wall times of RCA jobs. As the jobs get more complex (more PSO Islands), the communication overhead is also expected to increase, because a larger number of processes increases the chance that they will be allocated on different computing nodes on the cluster. In contrast, the SCA protocol divides processes across one or more process groups and each of the processes within a single group is guaranteed to be executed on the same cluster node. This means that communication within the groups is realized through shared memory and only the communication between the groups has to be realized via network transfer (less performant). This reduces the communication overhead compared to the RCA protocol and is likely to result in shorter wall times (and increased throughput).

Figure 4 highlights a subset of the data shown in Table 2 with a constant number of 50 PSO islands. Consider the wall time plots in the top row of

Table 2: Wall time and throughput performance of MultiGrain/MAPPER based on artificial GRN systems and data.

		RCA Protocol						**SCA** Protocol					
GRN	PSO	Wall Time [s]			Throughput [1/s]			Wall Time [s]			Throughput [1/s]		
Genes	Islands	Inula	Zeus	Both	Inula	Zeus	Both	Inula	Zeus	Both	Inula	Zeus	Both
11	1	14	11	-	3609	4595	-	11	11	-	4470	4436	-
11	2	30	14	18	3384	9560	5714	11	12	12	8939	8300	8205
11	5	69	28	33	3614	19826	7913	13	15	14	19872	16696	18168
11	10	144	36	63	3489	15159	8509	14	16	17	35788	31667	30237
11	20	319	103	141	3144	10970	7127	16	19	21	61765	52690	48139
11	50	763	240	291	3282	11824	8651	19	26	28	133187	95055	88237
11	100	1449	656	707	3482	8741	7471	20	28	31	247870	178343	163441
11	200	2822	592	1206	3567	16968	8312	-	-	44	-	-	226062
22	1	40	32	-	1262	1674	-	30	33	-	1659	1526	-
22	2	81	47	40	1240	2290	2561	32	35	34	3108	2850	2979
22	5	179	128	102	1411	2153	2929	36	43	41	6882	5788	6121
22	10	437	226	163	1144	2518	3118	43	60	51	11537	8319	10008
22	20	787	228	370	1279	7630	2757	59	73	67	17014	13791	15488
22	50	1890	595	852	1333	4802	2940	68	85	79	36646	29690	31624
22	100	3839	2203	1566	1307	4065	3220	73	92	89	68752	54615	56633
22	200	7011	3648	3180	1431	3038	3153	-	-	112	-	-	89501
33	1	115	74	-	438	719	-	71	71	-	708	705	-
33	2	194	110	117	521	974	913	79	87	78	1277	1154	1282
33	5	472	206	401	534	1339	659	93	125	92	2704	2036	2747
33	10	887	601	372	564	840	1412	115	180	139	4372	2869	3630
33	20	2088	673	792	489	2401	1276	179	208	187	5608	4940	5398
33	50	4526	1322	1915	554	2280	1317	210	270	224	11916	9283	11721
33	100	10040	4376	3503	515	1471	1439	238	276	287	21058	18111	17931
33	200	17760	10179	-	568	1522	-	-	-	315	-	-	32348

the figure. As expected, the wall times for larger GRN systems goes up with the number of genes. This effect is much more pronounced in the RCA than the SCA protocol. For all computing environment setups (Inula, Zeus, 2-cluster), the wall times are considerably longer in the RCA compared to the SCA protocol. Within the RCA protocol, the wall times on the Zeus cluster are much shorter than those on the Inula cluster, and in the cross-cluster setup, the wall times are still shorter than on the Inula cluster alone. The throughput plots in the bottom row of Figure 4 show that SCA protocol throughput (at least for a lower number of genes) is much higher than that of the RCA protocol. Furthermore, the throughput increase with smaller number of genes in the SCA protocol is much more pronounced than that in the RCA protocol. The plots in Figure 4 also show an interesting seemingly contradictory pattern. While Inula is clearly the slowest in terms of wall time in the RCA protocol, Inula seems to outperform Zeus in terms of throughput in the SCA protocol. This pattern is more pronounced for smaller number of genes.

Figure 5 highlights a subset of the data with a constant number of 22 genes. Consider the wall time plots in the top row. We immediately notice two things. First, the wall times produced with the RCA protocol are considerably longer (at least for larger numbers of PSO islands) than those in the SCA configuration. Second, the wall times increase in a roughly linear fashion with the addition of PSO islands in RCA, and are roughly constant in the SCA configuration. Because an increasing number of PSO islands means a proportional increase of both load (number of particle fitness evaluations) and resources (PSO islands corresponding to CPU cores), the latter indicates good scalability whereas the former indicates poor scalability. This view of scalability is corroborated in the throughput plots (bottom row of Figure 5). As load and resources are increased in a proportional manner (increasing number of PSO islands), the throughput in the SCA protocol increases approximately linearly (as one would expect in a system with good weak scalability), whereas the throughput in the RCA protocol remains roughly constant. So in our case, an ideal weak scalability scenario is one in which the wall time remains constant and the throughput increases linearly with the number of PSO islands. Recall, for each PSO island we allocate a CPU core. This is clearly seen in the plots for the SCA protocol in Figure 5. For the wall time plots, the ideal scalability line would be parallel to the x-axis and pass through the point corresponding of 1 PSO island (or, if the previous is not plotted, the point corresponding to the 2-island scenario). Similar to the

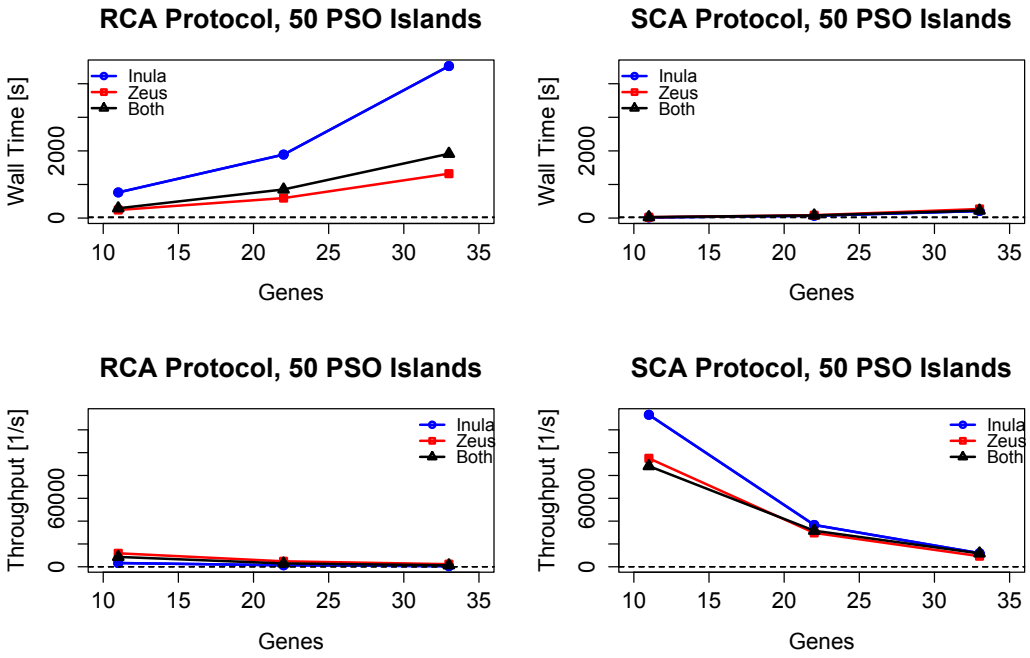


Figure 4: Wall times and throughput (both on y-axes) dependent on number genes (11, 22, 33), resource allocation protocol (RCA, SCA), and computing environment (local, distributed). Number of PSO islands constant at 50.

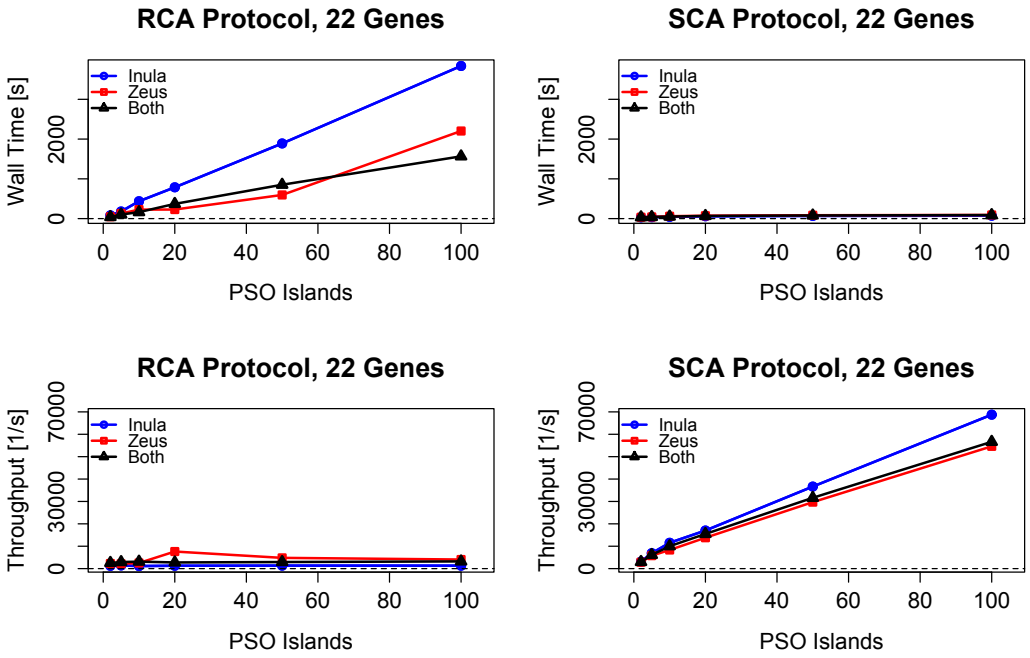


Figure 5: Wall times and throughput (both on y-axes) dependent on number of PSO islands (2, 5, 10, 20, 50, 100), resource allocation protocol (RCA, SCA), and computing environment (local, distributed). Number of genes is constant at 22.

data with varying gene number (11 to 33) and constant PSO islands (50) in Figure 4, we notice that within the RCA setup, Inula shows the longest wall times, while in the SCA setup Inula seems to outperform the other configurations in terms of throughput. This pattern seems more pronounced for scenarios involving a higher number of PSO islands.

Because many factors could potentially affect the performance in both protocol setups, it is difficult to pinpoint the exact causes of performance variation. The most likely reason for the lower performance of the RCA protocol is the delay due to communication among the nodes over which we randomly distribute the PSO islands. Some of the observed performance patterns depend on some subtle aspects of the computing configuration (protocols, hardware) and the particle PSO configuration. For instance, for jobs using the RCA allocation method, each island can be scheduled on any of the available nodes, so in the worst case, all PSO island processes could end up on a different node. This results in an increased communication latency.

With the SCA jobs, however, we manually specify the allocation of processes across computation nodes, which is likely to produce better communication performance. The core allocation size was limited to 24 processes per node for Inula, and 16 processes per node for Zeus (the latter was due to the specification of the reservations we obtained for SCA jobs). In order to achieve the best possible performance, we distributed the total number of processes for each job in allocations of sizes up to 24 processes on Inula and 16 processes on Zeus. This led to a smaller number of process allocations on Inula, especially for larger jobs (with a larger number of PSO islands). Fewer allocations means fewer communications across different cluster nodes, which leads to a lower communication overhead in the case of Inula scenarios. Finally, another critical factor for the efficiency of the SCA scenarios was the frequency of the CPU cores used to execute the MultiGrain/MAPPER computing jobs. Although we used AMD nodes from both execution sites, the Inula nodes contained CPU cores clocked at higher frequencies than those on Zeus. These two factors are likely to have led to increased efficiency for the SCA results on the Inula cluster.

In terms of training and validation errors of the GRN models that we reverse-engineered from the data of the three artificial systems, the results paint a much simpler picture. As expected, the model errors do not depend on the type of process allocation protocol. However, the errors did show a degree of dependence on the number of islands. The largest errors were obtained by scenarios with only one or two PSO islands. The values then decreased slightly, before stabilizing around a specific value. For the training error, this value was $0.08 - 0.09$ for 11-gene scenarios, $0.06 - 0.07$ for 22-gene scenarios and 0.10 for 33-gene scenarios.

However, we notice that the number of PSO islands that produced the lowest training and validation errors seems to depend on the size of the GRN that was reverse-engineered. Thus, for 11-gene scenarios, the lowest errors were achieved with as little as 2 or 5 PSO islands. For 22-gene scenarios, the optimal number of PSO islands was 10, while for 33-gene scenarios, the lowest errors were achieved with 20 islands. This was expected, as the number of PSO islands dictated the size of the total PSO swarm of particles (each PSO island had a fixed size of 50 particles). As the size of the GRN increases, the number of parameters also grows – the number of ANN parameters is $N(N+3)$, where N is the number of genes. As the number of parameters increases, larger PSO swarms are required in order to produce good results, when the number of PSO iterations (and fitness evaluations) is kept constant.

4.3. Efficiency in local and distributed scenarios: sea urchin data

In addition to the performance benchmarking on the three artificial GRN systems, we also reverse-engineered three models from the sea urchin data [35, 36, 37] with 11, 22 and 33 gene expression data series, respectively, as well as with the combined set of 66 data series. Apart from the different training and validation data and the swarm size (500 PSO particles over 10 PSO islands), the reverse-engineering conditions were identical to those employed for the artificial GRN scenarios. We performed each scenario using both process allocation methods (RCA, SCA) on the Inula cluster only. The results are shown in Table 3.

Table 3: Wall time and throughput performance of MultiGrain/MAPPER based on real gene-expression data (sea urchin development). Superior performance figures are highlighted in bold.

GRN	PSO	**RCA** Protocol (Inula)		**SCA** Protocol (Inula)	
Genes	Islands	Wall Time [s]	Throughput [1/s]	Wall Time [s]	Throughput [1/s]
11	10	392	1279	**125**	**8222**
22	10	942	536	**182**	**3807**
33	10	1707	297	**299**	**1985**
66	10	7989	64	**1426**	**360**

The results obtained from the real biological data (sea urchin) paint a similar picture as those of the artificial GRN data. The wall times were larger than the corresponding values for artificial data (i.e. for 10-island scenarios). However, this was to be expected, as the real datasets that we used here contained a larger number of time points (48) compared to the artificial datasets (16). As the number of time points grows, the fitness evaluator needs to perform more computations as the ODE solver needs to estimate additional data points.

Nevertheless, we noticed similar "trends" with the real data results in terms of the RCA and SCA protocols. Wall times for the RCA protocol were approximately three times larger than for the SCA protocol, when reverse-engineering 11-gene GRNs. This value grew to a 7-fold difference for the 66-gene GRN. The throughput was consistenly 6 or 7 times higher for the SCA scenarios than for RCA, regardless of the GRN size. Finally, training and validation errors (not shown in the table) were unaffected by the choice

of allocation protocol. However, they depended on the GRN dataset that was reverse-engineered.

4.4. Efficiency results summary for both artificial and sea urchin data

The computing performance assessment of MultiGrain/MAPPER revealed a number of interesting insights. The main computational performance results are summarized in Table 4. What is not shown in the table is the observation that an increase of the number of PSO islands beyond 10 does not lead to a substantial improvement of training and validation errors on artificial GRN systems. This is the reason why we used 10 PSO islands in the performance evaluation with the sea urchin data.

Table 4: Summary of main computational performance characteristics of MultiGrain/MAPPER.

A: Absolute Performance at Constant System Size and PSO Islands

Constant Number of Genes/Islands				RCA		SCA	
Site	Organism	Genes	Islands	WT [s]	TP [1/s]	WT [s]	TP [1/s]
Inula	Artificial	22	10	437	1144	**43**	**11537**
Inula	Sea Urchin	22	10	942	536	**182**	**3807**
Zeus	Artificial	22	10	226	2518	**60**	**8319**

B: Scaling Performace Dependent on Incresing System Size and PSO Islands

Increasing Number of Genes/Islands				RCA		SCA	
Site	Organism	Genes	Islands	WTSF	TPSF	WTSF	TPSF
Inula	Artificial	22	5 to 50	10.572	0.944	**1.879**	**5.325**
Inula	Artificial	11 to 33	10	**6.171**	**0.162**	8.186	0.122
Inula	Sea Urchin	11 to 33	10	4.355	0.232	**2.392**	**0.241**
Zeus	Artificial	22	5 to 50	2.887	2.231	**1.879**	**5.130**
Zeus	Artificial	11 to 33	10	16.511	0.055	**11.367**	**0.091**

WT & TP: *Absolute* wall times and throghputs for 22 genes and 10 PSO islands.
WTSF & TPSF: Wall time & throughput *scale factors* from 11 to 33 genes & 5 to 50 PSO islands.
RCA & SCA: Random and structured core assignment approach, respectively.

Consider the upper panel labeled A in Table 4. It shows the *absolute* wall times and throughput figures based on 22-gene systems on 10 PSO islands. The gene size of 22 genes is chosen as a medium-complexity GRN system, and the 10 PSO islands are chosen, as explained, because errors did not

show a great improvement above 10 PSO islands. The figures demonstrate the superiority of the SCA over the RCA protocol. The RCA wall times are by a factor of 3.76 to 10.14 longer than the SCA wall times, and the SCA throughput is by a factor 3.30 to 10.08 higher than those of the RCA protocol. Clearly, this is quite an interesting result. Future large-scale GRN model reverse-engineering should take these insights into account and consider to adopt a resource allocation strategies similar to SCA.

The lower panel B in Table 4 summarizes MultiGrain/MAPPER's performance characteristics in response to increasing problem size (number of genes) and increasing number of PSO islands. Bold figures highlight the protocol with the better performance (short wall time, higher throughput). The figures in panel B show *scale factors* of wall time and throughput (not absolute values) figures based on varying the number of genes/PSO islands from a smaller to a larger number. The *wall time scale factor* ($WTSF$) describes the factor by which the wall time increases as we go from 11 to 33 genes and from 5 to 50 PSO islands, respectively. The *throughput scale factor* ($TPSF$) describes the factor by which the throughput increases as we go from 11 to 33 genes and from 5 to 50 PSO islands, respectively.

First, we look at the increase of the number of genes from 11 to 33 with a fixed number of 10 PSO islands (rows 2, 3 and 5 in panel B). Notice that increasing the number of genes from 11 to 33 corresponds to a *7.71-fold* increase in the number of model parameters that need to be optimized. The number of parameters in the ANN rate law shown in Equation (1) grows with $N(N+3)$, where N is the number of genes in the GRN system. We do not observe a clear difference between the two protocols. For the Inula/artifical data combination, the WTSF and TPSF for RCA are slightly better than SCA, whereas for the Inula/sea urchin data and Zeus/artificial data combinations the scale factors are slightly in favor of the SCA protocol.

Second, we look at the increase of the number of PSO islands from 5 to 50 with a constant number of 22 genes in the GRN system (rows 1 and 3 in panel B). Increasing the number of PSO islands from 5 to 50 corresponds to a 10-fold increase of work load (number of particle fitness evaluations that need to be performed) and resources (number of CPU cores that are being used). When we increase the number of PSO islands by a factor of 10, we should see a TPSF close to 1 if the system has good scalability. For the RCA protocol, we observe a $TPSF = 0.944$ for the Inula cluster and $TPSF = 2.231$ for Zeus. The SCA protocol TPSF are approximately 5 for both protocols. So this corresponds to a kind of superlinear scalability. This

means that MultiGrain/MAPPER is likely to scale very well indeed for larger problems where we would need to increase the PSO islands to better explore the large parameter space. Overall, MultiGrain/MAPPER maps the multi-swarm PSO algorithm to the MAPPER distributed (multiscale) computing model. This solution seems to have very good scalability properties, when used with the right resource allocation protocol.

5. Conclusion

Modeling and simulation of complex biological systems is playing an increasingly important role in systems biology research. As more and more complex biological phenomena are being tackled, the need to capture and model information on multiple levels of biological organization grows. Hence, multiscale modeling and simulation approaches in biology are on the rise [43, 44]. The majority of current multiscale modeling and simulation solutions in the life sciences are based on ad hoc implementations. Some of the key challenges of these applications include the access and use of large-scale computing resources, and the development, deployment and execution of models and simulations consisting of multiple coupled sub-models [3, 44, 1].

Based on the MAPPER suite of multiscale modeling and simulation solutions [3, 5, 6], MultiGrain/MAPPER has been designed to tackle the computational challenges of large-scale GRN model modeling and simulation tasks. In particular, MultiGrain/MAPPER realizes a distributed computing solution to the problem of reverse-engineering GRN models from gene-expression data. The solution is based on a distributed multi-swarm (multi-island) particle swarm optimization algorithm that we implemented, where PSO islands are mapped to CPU cores. In this paper, we demonstrated that MultiGrain/MAPPER GRN model reverse-engineering performance (in terms of training and validation error of the inferred models) is equal or better than those of state-of-the-art modeling and simulation tools 1. We also evaluated the MultiGrain/MAPPER in terms of its *computational* performance based on larger GRN systems and varying computing environment dimensions (number of PSO islands, resource allocation protocols, and cluster configurations). In particular, we investigated how the reverse-engineering performance varies under two different research allocation protocols: the random core assignment (RCA) protocol, and the structured core assignment (SCA) protocol. The former allocates resources relatively quickly, but typically leads to suboptimal process-process communication, while the latter

is slower at allocation time but can take advantage of fast process-process communication links. We demonstrated that under the SCA protocol in particular, MultiGrain/MAPPER can produce superlinear speedup for larger reverse-engineering problems (see Tables 2 and 4).

MultiGrain/MAPPER components are implemented in Java, but MultiGrain/MAPPER takes advantage of various components, including those from the MAPPER stack tools and components [3, 1, 45, 46, 1, 47, 48, 49]. These components include user-friendly workflow editors and many other features that are beyond the scope of this article.

We envisage that future multiscale modeling and simulation challenges will involve the decomposition of a larger GRN system into modules of subsystems. Some of these sub-systems may operate on different scales (particularly time scales). By design, MultiGrain/MAPPER should be able to model such multiscale GRN systems.

Acknowledgment

This research received funding from the MAPPER EU-FP7 project (grant no. RI-261507) and was supported in part by PL-Grid infrastructure. This work was also supported by the National Science Centre (NCN) in Poland under the MAESTRO grant Nr DEC-2013/08/A/ST6/00296.

References

[1] J. Borgdorff, J.-L. Falcone, E. Lorenz, C. Bona-Casas, B. Chopard, A. G. Hoekstra, Foundations of distributed multiscale computing: Formalization, specification, and analysis, Journal of Parallel and Distributed Computing 73 (4) (2013) 465–483.

[2] K. Rycerz, M. Bubak, E. Ciepiela, D. Harelak, T. Gubaa, J. Meizner, M. Pawlik, B. Wilk, Composing, execution and sharing of multiscale applications, Future Generation Computer Systems 53 (2015) 77–87.

[3] J. Borgdorff, M. Belgacem, C. Bona-Casas, L. Fazendeiro, D. Groen, O. Hoenen, A. Mizeranschi, J. Suter, D. Coster, P. Coveney, W. Dubitzky, A. Hoekstra, P. Strand, B. Chopard, Performance of distributed multiscale simulations, Royal Society A: Mathematical, Physical and Engineering Sciences 372 (2021).

[4] B. Chopard, J. Borgdorff, A. G. Hoekstra, A framework for multi-scale modelling, Philosophical Transactions of the Royal Society of London A: Mathematical, Physical and Engineering Sciences 372 (2021).

[5] M. Ben Belgacem, B. Chopard, J. Borgdorff, M. Mamonski, K. Rycerz, D. Harezlak, Distributed multiscale computations using the MAPPER framework, Procedia Computer Science 18 (2013) 1106–1115.

[6] D. Groen, J. Borgdorff, C. Bona-Casas, J. Hetherington, R. W. Nash, S. J. Zasada, I. Saverchenko, M. Mamonski, K. Kurowski, M. O. Bernabeu, A. G. Hoekstra, P. V. Coveney, Flexible composition and execution of high performance, high fidelity multiscale biomedical simulations, Interface Focus 3 (2) (2013) 20120087–20120087.

[7] N. Kennedy, A. Mizeranschi, P. Thompson, H. Zheng, W. Dubitzky, Reverse-engineering of gene regulation models from multi-condition experiments, in: IEEE Symposium Series on Computational Intelligence 2013 (SSCI 2013), Singapore, 2013, pp. 112–119.

[8] A. Villaverde, J. Banga, Reverse engineering and identification in systems biology: Strategies, perspectives and challenges, Journal of the Royal Society Interface 2014 (11) (2013) 20130505.

[9] D. Marbach, J. C. Costello, R. Küffner, N. M. Vega, R. J. Prill, D. M. Camacho, K. R. Allison, C. The DREAM5, M. Kellis, J. J. Collins, G. Stolovitzky, Wisdom of crowds for robust gene network inference, Nature Methods 9 (2012) 796–804.

[10] S. Baker, B. Kramer, Systems biology and cancer: Promises and perils, Progress in Biophysics and Molecular Biology 106 (2011) (2011) 410–413.

[11] M. T. Swain, J. J. Mandel, W. Dubitzky, Comparative study of three commonly used continuous deterministic methods for modeling gene regulation networks, BMC Bioinformatics 11 (1) (2010) 459.

[12] I. Cantone, L. Marucci, F. Iorio, M. Ricci, V. Belcastro, M. Bansal, S. Santini, di Bernardo M., di Bernardo D., C. M.P., A yeast synthetic network for in vivo assessment of reverse-engineering and modeling approaches, Cell 137 (2009) 172–181.

[13] Y. Barlas, Model validation in systems dynamics, in: International Systems Dynamics Conference, 1994, pp. 1–10.

[14] J. Vohradský, Neural network model of gene expression, The FASEB Journal 15 (3) (2001) 846–854.

[15] M. Savageau, Biochemical systems analysis: A study of function and design in molecular biology, Addison-Wesley, Reading, Mass., 1976.

[16] P. Mendes, W. Sha, K. Ye, Artificial gene networks for objective comparison of analysis algorithms, Bioinformatics 19 (suppl 2) (2003) ii122–ii129.

[17] S. Hoops, S. Sahle, R. Gauges, C. Lee, J. Pahle, N. Simus, M. Singhal, L. Xu, P. Mendes, U. Kummer, COPASI—A complex pathway simulator, Bioinformatics 22 (24) (2006) 3067–3074.

[18] E. Kent, S. Hoops, P. Mendes, Condor-COPASI: High-throughput computing for biochemical networks, BMC Systems Biology 6 (1) (2012) 91.

[19] A. López, Computational approaches to the modelling of topological and dynamical aspects of biochemical networks, Ph.D. thesis, Pompeu Fabra University, Barcelona, Spain (2010).

[20] E. Butterworth, B. E. Jardine, G. M. Raymond, M. L. Neal, J. B. Bassingthwaighte, JSim, an open-source modeling system for data analysis and reproducibility in research, F1000Research 2.

[21] A. I. Vaz, L. Vicente, A particle swarm pattern search method for bound constrained global optimization, Journal of Global Optimization 39 (2) (2007) 197–219.

[22] Z. Zi, SBML-PET-MPI: a parallel parameter estimation tool for systems biology markup language based models, Bioinformatics 27 (7) (2011) 1028–1029.

[23] M. Snir, S. W. Otto, D. W. Walker, J. Dongarra, S. Huss-Lederman, MPI: The complete reference, MIT Press, Cambridge, MA, USA, 1995.

[24] A. Dörr, R. Keller, A. Zell, A. Dräger, SBMLSimulator: A Java tool for model simulation and parameter estimation in systems biology, Computation 2 (4) (2014) 246–257.

[25] R. Keller, A. Dörr, A. Tabira, A. Funahashi, M. J. Ziller, R. Adams, N. Rodriguez, N. L. Novère, N. Hiroi, H. Planatscher, others, The systems biology simulation core algorithm, BMC Systems Biology 7 (1) (2013) 55.

[26] I. Moraru, J. Schaff, B. Slepchenko, M. Blinov, F. Morgan, A. Lakshminarayana, F. Gao, Y. Li, L. Loew, Virtual Cell modelling and simulation software environment, IET Systems Biology 2 (5) (2008) 352.

[27] R. Adams, A. Clark, A. Yamaguchi, N. Hanlon, N. Tsorman, S. Ali, G. Lebedeva, A. Goltsov, A. Sorokin, O. E. Akman, C. Troein, A. J. Millar, I. Goryanin, S. Gilmore, SBSI: an extensible distributed software infrastructure for parameter estimation in systems biology, Bioinformatics 29 (5) (2013) 664–665.

[28] J. Kennedy, R. Eberhart, Particle swarm optimization, in: Proceedings of IEEE International Conference on Neural Networks, Vol. 4, 1995, p. 1942–1948.

[29] Y. Shi, R. Eberhart, A modified particle swarm optimizer, in: Proceedings of IEEE International Conference on Evolutionary Computation, 1998, pp. 69–73.

[30] J. Borgdorff, M. Mamonski, B. Bosak, K. Kurowski, M. Ben Belgacem, B. Chopard, D. Groen, P. V. Coveney, A. G. Hoekstra, Distributed multiscale computing with MUSCLE 2, the Multiscale Coupling Library and Environment, Journal of Computational Science, in press.

[31] J.-L. Falcone, B. Chopard, A. Hoekstra, MML: Towards a multiscale modeling language, Procedia Computer Science 1 (1) (2010) 819–826.

[32] M. Radecki, T. Szymocha, T. Piontek, B. Bosak, M. Mamoski, P. Wolniewicz, K. Benedyczak, R. Kluszczyski, Reservations for compute resources in federated e-infrastructure, in: M. Bubak, J. Kitowski, K. Wiatr (Eds.), eScience on Distributed Computing Infrastructure, Vol. 8500 of Lecture Notes in Computer Science, Springer International Publishing, 2014, pp. 80–93.

[33] M. T. Flanagan, Michael thomas flanagan's java scientific library (2012). URL http://www.ee.ucl.ac.uk/~mflanaga/java/

[34] B. J. Bornstein, S. M. Keating, A. Jouraku, M. Hucka, LibSBML: an API library for SBML, Bioinformatics 24 (6) (2008) 880–881.

[35] E. Davidson, M. Levine, Gene regulatory networks, Proceedings of the National Academy of Sciences of the United States of America 102 (14) (2005) 4935.

[36] S. Materna, J. Nam, E. Davidson, High accuracy, high-resolution prevalence measurement for the majority of locally expressed regulatory genes in early sea urchin development, Gene Expression Patterns 10 (4–5) (2010) 177–184.

[37] S. Damle, E. Davidson, Synthetic in vivo validation of gene network circuitry, Proceedings of the National Academy of Sciences 109 (5) (2012) 1548–1553.

[38] F. M. Lopes, R. M. Cesar, Jr, L. F. Costa, AGN simulation and validation model, in: Proceedings of the 3rd Brazilian symposium on Bioinformatics: Advances in Bioinformatics and Computational Biology, BSB '08, Springer-Verlag, Berlin, Heidelberg, 2008, p. 169–173.

[39] R. J. Hyndman, A. B. Koehler, Another look at measures of forecast accuracy, International Journal of Forecasting 22 (4) (2006) 679 – 688.

[40] M. E. H. Pedersen, Good parameters for particle swarm optimization, Technical Report HL1001, Hvass Laboratories.

[41] T. P. Runarsson, X. Yao, Stochastic ranking for constrained evolutionary optimization, IEEE Transactions on Evolutionary Computation 4 (2000) 284–294.

[42] K. Kurowski, W. Back, W. Dubitzky, L. Gulyás, G. Kampis, M. Mamonski, G. Szemes, M. Swain, Complex system simulations with QosCosGrid, in: G. Allen, J. Nabrzyski, E. Seidel, G. Albada, J. Dongarra, P. Sloot (Eds.), Computational Science – ICCS 2009, Vol. 5544 of Lecture Notes in Computer Science, Springer Berlin Heidelberg, 2009, pp. 387–396.

[43] J. O. Dada, P. Mendes, Multi-scale modelling and simulation in systems biology, Integrative Biology 3 (2) (2011) 86–96.

[44] D. Groen, S. Zasada, C. P.V, Survey of multiscale and multiphysics applications and communities, Computing in Science and Engineering 16 (2) (2014) 34–43.

[45] B. Bosak, P. Kopta, K. Kurowski, T. Piontek, M. Mamoski, New QosCosGrid middleware capabilities anditsintegration with European e-infrastructure, in: M. Bubak, J. Kitowski, K. Wiatr (Eds.), eScience on Distributed Computing Infrastructure, Vol. 8500 of Lecture Notes in Computer Science, Springer International Publishing, 2014, pp. 34–53.

[46] V. Kravtsov, A. Schuster, D. Carmeli, K. Kurowski, W. Dubitzky, Grid-enabling complex system applications with QosCosGrid: An architectural perspective, in: Proceedings of the 2008 International Conference on Grid Computing & Applications, Las Vegas, Nevada, USA, 2008, pp. 168–174.

[47] E. Ciepiela, D. Harezlak, J. Kocot, T. Bartynski, M. Kasztelnik, P. Nowakowski, T. Gubała, M. Malawski, M. Bubak, Exploratory Programming in the Virtual Laboratory, in: Proceedings of the 2010 International Multiconference on Computer Science and Information Technology, Institute of Computer Science, AGH, Krakow, Poland, 2010, pp. 621–628.

[48] D. Groen, S. Rieder, P. Grosso, C. de Laat, S. F. Portegies Zwart, A lightweight communication library for distributed computing, Computational Science & Discovery 3 (1) (2010) 015002.

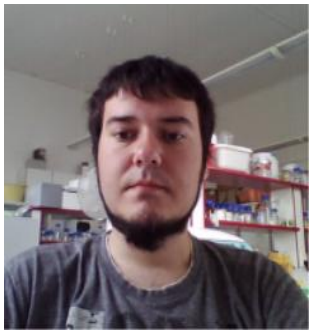
[49] S. J. Zasada, M. Mamonski, D. Groen, J. Borgdorff, I. Saverchenko, T. Piontek, K. Kurowski, P. V. Coveney, Distributed infrastructure for multiscale computing, in: 2012 IEEE/ACM 16th International Symposium on Distributed Simulation and Real Time Applications (DS-RT), IEEE, 2012, pp. 65–74.

*Biographies (Text)

Short biography: Alexandru Mizeranschi

Alexandru Mizeranschi is a bioinformatician at Charles University, Prague. His work focuses on the analysis of high-throughput, next-generation sequencing data. His general research interests include the application of general software engineering and modeling and simulation techniques to multidisciplinary problems spanning the life sciences, cognitive science and artificial intelligence.

Photograph:



Short biography: Martin T. Swain

Martin Swain is a Lecturer in Bioinformatics at the Institute of Biological Environmental and Rural Sciences, Aberystwyth University. His main interests are in high-throughput DNA sequencing, knowledge discovery and high performance computing.

Photograph:

Short biography: Raluca Scona

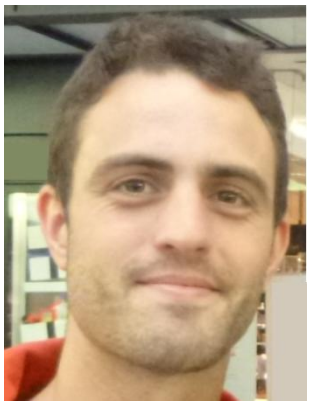
Raluca Scona is a student in the EPSRC Robotics and Autonomous Systems Centre for Doctoral Training offered by the University of Edinburgh and Heriot-Watt University. Her main interests are machine learning, probabilistic robotics, state estimation and SLAM.

Photograph:

Short biography: Quentin Fazilleau

Quentin Fazilleau is working at Capgemini (Toulouse, France) as a SQL developer. His educational background is in computer science with specialization in databases and statistics.

Photograph:



Short biography: Bartosz Bosak

Bartosz Bosak received his M.Sc. degree in computer science from Poznań University of Technology in Poland (Laboratory of IT Systems in Management). Since 2007 he has been working at the Application Department of Poznań Supercomputing and Networking Center as a system analyst and developer. His research interests include grid technology, high-performance computing, communication in distributed environments and service integration in SOA.

Photograph:

Short biography: Tomasz Piontek

Tomasz Piontek received his MSc in computer science in 1998 at Poznan University of Technology (Parallel and Distributed Computing). After his MSc, he joined the programmers group at PUT and worked on mobile network protocol analyzers software for Siemens A.G. and Tektronix, Inc. Since 2002 he has been working in Poznań Supercomputing And Networking Center in Application Department and was involved in many EU-funded and national R&D projects in the area of grids and HPC computing: GridLab, ACGT, QosCosGrid, PL-Grid, MAPPER, CoolEmAll. His research interests include distributed computing, large-scale simulations and resource management. He is an author or co-author of several papers in professional journals and conference proceedings. At PSNC he leads a team developing QosCosGrid middleware services and tools and is responsible for collaboration with domain research groups in the PLGrid PLUS project.

Photograph:

Short biography: Piotr Kopta

Piotr Kopta received his M.Sc. degree in Computer Science from the Technical University of Częstochowa in 2002. Currently he is an systems analyst at the Poznań Supercomputing and Networking Center. His research interests include high-performance computing and novel computating architectures.

Photograph:

Short biography: Paul Thompson

Dr Paul Thompson is a Senior Lecturer of Molecular Biology at the Biomedical Sciences Research Institute, University of Ulster. His main interest is in nuclear receptor signaling and transcriptional regulation of gene networks.

Photograph:

Short biography: Werner Dubitzky

Werner Dubitzky is a Professor of Bioinformatics at Biomedical Sciences Research Institute, University of Ulster. His main interest is in computational systems biology, machine learning, data mining and large-scale computing.

Photograph: