Distributed Cell Biology Simulations with E-Cell System

Masahiro Sugimoto[1,2], Kouichi Takahashi[1,⋆], Tomoya Kitayama[3], Daiki Ito[3], and Masaru Tomita[1]

[1] Institute for Advanced Biosciences, Keio University, Tsuruoka, Yamagata 997-0035, Japan
msugi@sfc.keio.ac.jp, shafi@e-cell.org, mt@sfc.keio.ac.jp, http://www.e-cell.org
[2] Bioinformatics Department, Mitsubishi Space Software Co. Ltd., Amagasaki, Hyogo, 661-0001, Japan
sugi@cbo.mss.co.jp
[3] Laboratory for Bioinformatics, Keio University Fujisawa, 252-8520, Japan
{tomoyan, s02082di}@sfc.keio.ac.jp

Abstract. Many useful applications of simulation in computational cell biology, e.g. kinetic parameter estimation, Metabolic Control Analysis (MCA), and bifurcation analysis, require a large number of repetitive runs with different input parameters. The heavy requirements imposed by these analysis methods on computational resources has led to an increased interest in parallel- and distributed computing technologies.

We have developed a scripting environment that can execute, and where possible, automatically parallelize those mathematical analysis sessions transparently on any of (1) single-processor workstations, (2) Shared-memory Multiprocessor (SMP) servers, (3) workstation clusters, and (4) computational grid environments. This computational framework, E-Cell SessionManager (ESM), is built upon E-Cell System Version 3, a generic software environment for the modeling, simulation, and analysis of whole-cell scale biological systems.

Here we introduce the ESM architecture and provide results from benchmark experiments that addressed 2 typical computationally intensive biological problems, (1) a parameter estimation session of a small hypothetical pathway and (2) simulations of a stochastic *E. coli* heat-shock model with different random number seeds to obtain the statistical characteristics of the stochastic fluctuations.

1 Introduction

Computational biology requires high-performance computing facilities. There are many parallel biological applications that can be used in PC cluster environments, e.g. HMMer, FASTA, mpiBLAST, PARACEL BLAST, ClustalW-MPI[1], Wrapping up BLAST[2], and TREE-PUZZLLE[3]. We have developed

⋆ Corresponding author.

A. Konagaya and K. Satou (Eds.): LSGRID 2004, LNBI 3370, pp. 20–31, 2005.

an integrated software environment for computational cell biology, the E-Cell System[4, 5, 6, 7]. It is an object-oriented software suite for modeling the simulation and analysis of large-scale complex systems such as biological cells. E-Cell has a hierarchical parallel computing scheme in which (1) simulation sessions can be concurrently executed on remote computation nodes in grid or cluster environments, and (2) each run of the simulator can be parallelized on a shared-memory multi-processor computer employing multiple threads. Here we discuss the scheme for session-level parallelism, or distributed computing. Elsewhere we explained the other scheme, parallelization of Gillespie's Stochastic Simulation Algorithm (SSA) on E-Cell3[8], or simulator-level parallelism. As many simulation applications in computational cell biology require repetitive runs of simulation sessions with different model- and boundary parameters, the distributed computation scheme is highly useful.

Some middleware to assign jobs to distributed environments is already available, e.g. the Portable Batch System (PBS, http://www.pbs.org/), Load Sharing Facility (LSF, http://www.platform.com/), and Sun Grid Engine (SGE, http://wwws.sun.com/software/gridware/) on the cluster level, and the Globus toolkit[9] on the Grid level. While these low-level infrastructures are extremely powerful, they are not compatible with each other nor are they readily accessible to the average computational biologists. Some higher-level middleware focusing on the use of bioinformatics applications has been developed. For example, the Discovery Net System[10] is an application for Genome Annotation, Talisman[11] is a component of the ^{my}Grid[12] project that provides a framework to produce web-based applications, OBIGrid[13] accommodates BLAST tasks in a grid environment, OBIYagns is a grid-based simulation environment for parameter estimations[14], and Nimrod/G is a job-scheduler system on dynamic global resources[15]. Many other projects are ongoing under the BioGrid project (http://www.biogrid.jp/). A distributed version of E-Cell, E-Cell2D[16], which has a client-server architecture where the client uses a Web browser, has been developed. Although this version of E-Cell is useful for classroom applications, it lacks the scripting feature necessary for the automation of mathematical analysis sessions.

Here we report the development of a distributed computing module of E-Cell System Version 3 (E-Cell3), E-Cell Session Manager (ESM), which can transparently support PC-, SMP-, cluster-, and grid environments. We present performance evaluations of parallel computations using ESM, (1) a parameter estimation of a small hypothetical pathway and (2) simulations of a stochastic *E. coli* heat-shock model with different random number seeds to obtain the statistical characteristics of the model.

2 Architecture

2.1 E-Cell3 and ESM

E-Cell3 is comprised of 3 layers: 1). a class library for cell simulation (libecs) and its C++ API (libemc), 2) a Python language wrapper of libemc, PyEcs

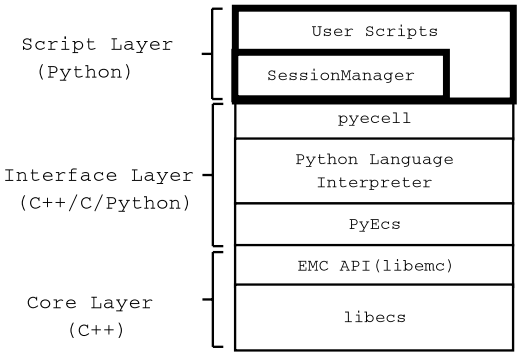

Fig. 1. ESM Architecture. The bottom layer includes a class library for cell simulation (libecs) and its C++ API (libemc). The top layer represents python front-end utilities such as SessionManager, GUI, and analysis tools.The middle layer (PyEcs, python interpreter, and pyecell) is the interface connecting the bottom and top layers

and pyecell, and 3) a library of various front-end and utility modules written in Python (Fig. 1). The pyecell library defines an object class called Session, which represents a single run of the simulator. ESM, constructed on top of the pyecell layer, enables the user to easily script procedures of mathematical analysis methods that instantiate many Session objects.

2.2 ESM Design

The fundamental design of ESM is shown in Fig. 2 as a class diagram. It is comprised of 3 classes, the SessionManager-, SessionProxy-, and SystemProxy class. The SessionManager class provides the user with a basic API to create and run simulation sessions. SessionManager generates and holds a SystemProxy object; it represents and communicates to the computing environment on which ESM is running (such as PC, SMP, cluster, or grid). On request, SystemProxy generates instances of SessionProxy; it corresponds to a process on PC- and SMP environments or a job on cluster- and grid environments, and holds the status of the process or job (waiting, running, recoverable error, unrecoverable error, or finished). The status chart of SessionProxy is shown in Fig. 3.

As depicted in Fig. 2, subclasses of Session and SystemProxy are instantiated and used by the SessionManager class according to the environment. For example, a pair of LocalSystemProxy and LocalSessionProxy is selected when the system is running on a single CPU PC or a SMP machine; the system uses SGESystemProxy and SGESessionProxy when the user request is to parallelize the computation on an Sun Grid Engine (SGE) parallel-batch system. On an SMP or a PC computer, they spawn ordinary processes in the local computer and use system calls to manage the tasks. On cluster and grid environments, these classes make contact with the system that manages the computing environment to submit jobs and obtain the job status.

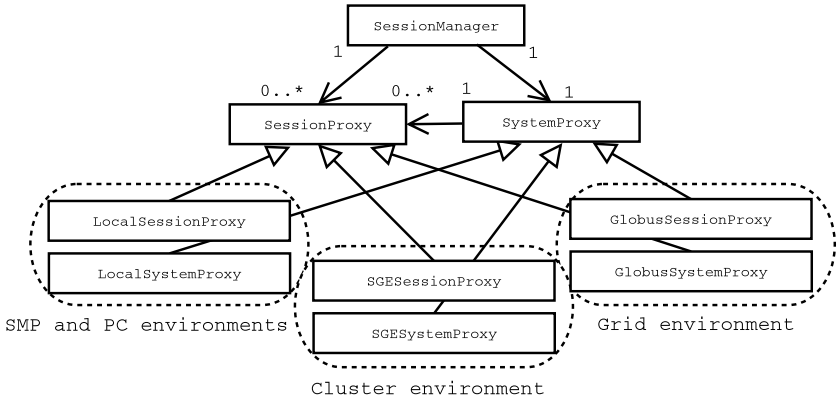

Fig. 2. ESM Design. The SessionManager class provides an API for user scripting. SystemProxy is a proxy of the computing environment such as the cluster- or grid environment. SessionProxy corresponds to a process or a job. LocalSystemProxy and LocalSessionProxy are used both on SMP- and single-CPU PC environments

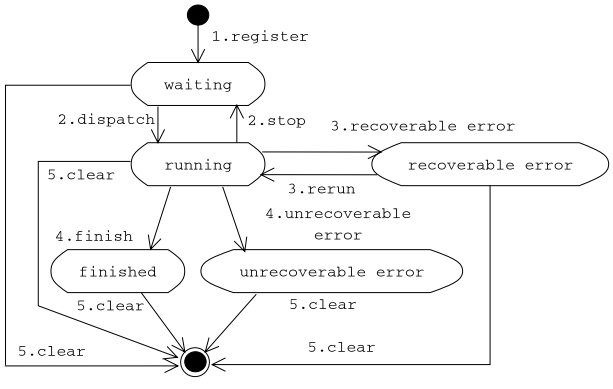

Fig. 3. Status chart representation of SessionProxy. (1) A SessionProxy is created when a job is registered; the initial status is 'waiting'. (2) It changes to 'running' when the registered job is dispatched to acomputation node to run, if the run is suspended, it returns to 'waiting'. (3) When a recoverable error occurs, the status changes to 'recoverable error', and the system retries the run until it reaches a user-specifiable limit. If the retry limit is reached, the job goes to 'unrecoverable error'. (4) A job terminates in either the 'finished' or 'unrecoverable error' status, depending on its exit status. (5) Users can terminate the job by invoking the clear() method of the SessionProxy

2.3 ESM Processing Scheme

At least 3 types of files are necessary to run ESM; (1) a model file (EML, or E-Cell model description language file), (2) a session script file (ESS, or E-Cell

```
% ecell3-session-manager --environment=Local ems.py                        (1)
% ecell3-session-manager --environment=SGE --concurrency=30 ems.py         (2)
% ecell3-session-manager --environment=Globus2 --concurrency=100 ems.py (3)
```

Fig. 4. Running ESM from command-line: This figure has three example command-lines that run the ecell3-session-manager command. '–environment='and '–concurrency=' command-line arguments specify the computing environment and concurrency, respectively. (1) Local environment (single-CPU PC or SMP). The default concurrency in this example is 1. This can be changed by explicitly giving the -concurrency= argument. (2) An SGE run with 30 CPUs. (3) A Globus run. The number of simultaneous executions of jobs is limited to 100. All runs execute the EMS file 'ems.py'

session script file), and (3) a Session Manager script file (EMS, or E-Cell session manager script file). Simple examples of command lines, an EMS, and an ESS are presented in Fig. 4, Fig. 5, and Fig. 6, respectively. Below we explain a typical flow of procedures in an EMS. Items (2) to (4) correspond to the bold comment lines in the script of Fig. 5.

(1) Set System Parameters
At least 2 system parameters, the computing environment and the concurrency, should be set when running ESM. The computing parameter specifies what type of facilities the ESM should use to run and control the jobs. The concurrency parameter specifies the maximum number of CPUs that the system can use simultaneously. These parameters are usually given as command-line arguments to the 'ecell3-session-manager' command, which runs an EMS indicated by the user. (Fig. 4)

(2) Register Jobs
The registerEcellSession() method in EMS registers a job. It accepts 3 arguments, (1) the session script (ESS) to be executed, (2) the optional parameters given to the job, (3) the input files to the script (at least the model file) that must be available to the ESS upon execution. In the example in Fig. 5, 100 copies of the session script 'runsession.py' are registered with a model file 'simple.eml'. An optional parameter to the script, 'VALUE_OF_S' is also given to each session in the range 1, 2, ..., 100. The parameter is available to the ESS as a global variable. When a job is registered to the system, a SessionProxy object for the job is instantiated with a unique ID.

(3) Run
When the run() method is called, the registered jobs start executing. In this step, SystemProxy transfers the ESS file and all the other files to the execution environment (either a directory in the local machine or a remote computation node). Next, SessionProxy starts execution of the job. When SystemProxy confirms that all jobs are either finished or have an unrecoverable error, the run() method returns. It is also possible to use an asynchronous scheme in which the

```
MODEL_FILE='model.eml'
ESS_FILE='runsession.py'

#(2)Register jobs.
aJobIDList = []
for VALUE_OF_S in range(0,100):
   aParameterDict = {'MODEL_FILE':MODEL_FILE, 'VALUE_OF_S':VALUE_OF_S}
   #        registerEcellSession(the script, parameters, files that the ESS uses)
   aJobID = registerEcellSession(ESS_FILE, aParameterDict, [MODEL_FILE,])
   aJobIDList.append(aJobID)   # Memorize the job IDs in aJobIDList

#(3)Run the registered jobs.
run()

#(4)Examine the results.
for aJobID in aJobIDList:       # Print the output of each job.
   print getStdout(aJobID)
```

Fig. 5. A sample EMS (E-Cell Session Manager Script): This script runs the session script 'runsession.py' 100 times with a changing parameter 'VALUE_OF_S'

```
loadModel( MODEL_FILE )                 # Load the model.

S = createEntityStub( 'Variable:/:S' ) # Create a stub object of
                                        # the simulator variable 'Variable:/:S'
S['Value'] = VALUE_OF_S                 # Set the value VALUE_OF_S given by the
                                        # EMS in Fig.5 to the variable.

run( 200 )                              # Run the simulation for 200 seconds.

message( S['Value'] )                   # Print the value of 'Variable:/:S'.
```

Fig. 6. A sample E-Cell session script (ESS): This script runs a simulation model for 200 sec and outputs the value of the variable 'Variable:/:S' of the model after the simulation. The initial value of the variable is changed to the value 'VALUE_OF_S' given by the EMS

run() method returns immediately and the user must check the status of the jobs explicitly.

(4) Examine the Results
After running the simulation sessions, simulation results are obtained and examined. The example script in Fig. 5 prints the output of the simulations to the screen; getStdout(aJobID) shows the standard-out of the job specified by a job ID. If more runs of the simulation are necessary, go to (2).

3 Results

This section demonstrates and evaluates the performance of ESM with 2 types of simulation experiments in computational cell biology. The first example is an

automatic estimation of 4 kinetic parameters in a hypothetical small metabolic pathway. The second demonstration runs a stochastic model of the *E. coli* heat-shock response repeatedly with different random number seeds to obtain statistical features of the trajectory yielded by the model. Both types of numerical experiments are representative of the numerical experiments most commonly conducted in computational cell biology research projects.

3.1 Kinetic Parameter Estimation Using Genetic Algorithms (GA)

In computational cell biology, it is often the case that some model parameters, e.g. rate constants, are unknown while other experimental data, e.g. concentrations at steady states and time-courses of concentrations of some molecular species, are available. We developed a parameter-estimation tool that runs on ESM. This program implements a variation of GA that is an evolutionary algorithm to search the global minimum of a given multi-variate fitness function avoiding local minima[17]. A brief overview of the procedures used in this GA program is as follows: (1) Generate individuals and randomly distribute them over the search space. (2) Evaluate each individual with a user-defined fitness function. A square-error function between given and simulation-predicted trajectories is often used as the fitness function. (3) Select a couple of individuals from the group of individuals according to some probability, and (4) crossover the individuals. (5) Mutate each individual. (6) Go to (2), unless the fitness of the best individual meets a user-specified condition. In this study, the master-slave parallel GA was employed. The individuals are evaluated on parallel computational resources in procedure (2); other procedures are conducted on the master node.

We conducted a benchmark experiment using the pathway model shown in Fig. 7. This tiny model of a hypothetical metabolic pathway contains 5 molecular species and 5 reactions, one of which is a positive feedback reaction. Each reaction is represented using an irreversible Michaelis-Menten equation. This requires 2 rate constants, the Michaelis constant KmS, and the catalytic constant Kcf. Four of the 10 parameters are unknown and to be predicted by the parameter estimation tool.

The results are shown in Table 1 and Fig. 8. We measured the time required for the relative square-error between the given and the predicted trajectories to reach the levels of 10%, 7.5%, 5.0%, and 2.5%. We show 10, 20, and 30 CPU cases for each level of the relative error. An SGE system running on a Linux cluster was used for all timings.

3.2 Stochastic Simulation of *E. coli* Heat-Shock Response

Many cellular phenomena require stochastic simulations whose results must be discussed by means of statistical measures. Therefore it is common for a numerical simulator to run a model repeatedly with different random number seeds. We conducted numerical experiments to measure the time taken to run an *E. coli* heat-shock model[18] on ESM.

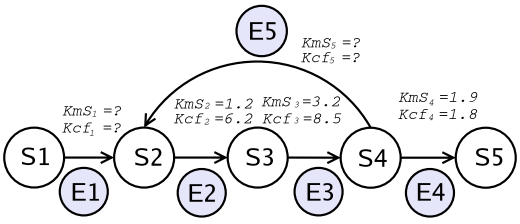

Fig. 7. An example model: S1, S2, S3, S4, and S5 are metabolites; E1, E2, E3, E4, and E5 are enzymes. All reactions are formulated by using Michaelis-Menten equations that have 2 parameters each, KmS and Kcf. The parameters for enzymes E1 and E5 are unknown and to be predicted

Table 1. Timings from the parameter estimation experiment using the GA. Of 10 rate constants in the model in Fig. 7, 4 are unknown. Four sets of training time-courses of concentrations of the 5 molecular species were given. The number of individuals is 100. Each row shows the times in sec required to reach 2.5%, 5.0%, 7.5%, and 10% of the relative error level between the given and the predicted time-courses. Each row has from 10 to 60 CPU cases, therefore, the table shows 24 results. A 40-node, 80-CPU Linux cluster running SGE was used. The CPUs were Pentium 4 Xeon 2.0GHz. Each node was connected by a 1000BaseT local area network

Relative	CPUs(Time s)					
Error %	10	20	30	40	50	60
2.5	11499	6022	4412	3354	2541	2089
5.0	1915	997	732	524	467	419
7.5	1273	604	474	373	304	283
10	627	346	250	183	152	147

The results are shown in Table 2. We measured the time required to run 100, 200, and 300 queries of 100-, 200-, and 300-sec simulations on a single CPU PC, a dual-CPU SMP machine, and a Linux cluster, setting concurrency to 1, 2, 5, 10, 20, and 30. The table, showing a total of 72 cases (3 x 3 x 8), indicates that timing increases roughly linearly with the number of queries on all the machine configurations. In the case with the shortest simulation time (100 sec), a 5-CPU run of the cluster runs as fast as the single-CPU PC; longer simulation times (500 and 1000 sec) lower the equilibrium point somewhere between 1 and 2-CPU runs of the cluster.

3.3 Discussions

The results of the numerical experiments presented in the previous section clearly show the benefits of using distributed computation in computational cell biology. Only when a small number of CPUs were used did the performance suffer from the overhead of parallel computing (see Table 2). This unacceptable overhead

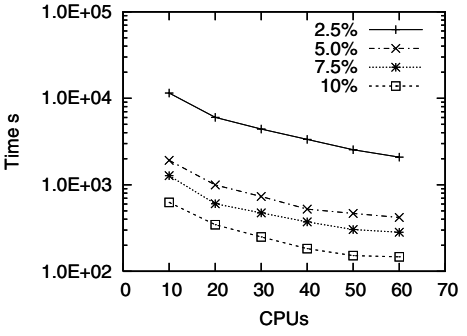

Fig. 8. Performance graph of the GA parameter estimation. Relative error levels of 10%, 7.5%, 5.0%, and 2.5% cases are shown with 10 to 60 CPU runs. The graph demonstrates that computation time increases steeply as the required relative error level decreases. See also Table 1

Table 2. Benchmark results of the stochastic heat-shock response model. We ran 100, 200, and 300 queries of simulations for 100-, 500-, and 1000 sec on a PC, a dual SMP, and 1, 2, 5, 10, 20, and 30 CPUs of a SGE Linux cluster. The CPUs, memory, and other hardware configurations were identical in all cases. The workstation cluster used for this experiments was the same as that used in section 3.1

Simulated		Environments & CPUs (Time s)							
Time	#queries	PC	SMP			SGE			
(s)		1	2	1	2	5	10	20	30
	100	302.8	153.4	1492.7	745.8	301.3	148.7	79.4	66.0
100	200	605.6	307.8	3006.7	1298.6	598.0	305.9	150.0	104.0
	300	910.0	461.9	4501.3	2280.0	902.5	452.6	232.4	152.2
	100	1117.8	557.0	1530.4	770.2	303.8	165.5	79.6	65.0
500	200	2223.4	1120.4	3011.7	1520.5	631.3	351.1	174.9	135.3
	300	3332.4	1677.2	4571.8	2268.1	912.4	475.6	258.2	158.5
	100	2106.0	1088.3	2992.1	1496.5	603.5	298.0	157.4	128.9
1000	200	4283.6	2186.5	6030.8	3011.7	1205.2	597.6	289.9	222.3
	300	6571.8	3292.1	9039.8	4522.9	1802.1	912.5	463.7	312.6

is attributable to the processing times required for job submission, transfer and data retrieval, however, these procedures are not necessary in PC- and SMP environments.

ESM can distribute simulation sessions of E-Cell transparently on virtually any distributed computation environments. All scripts can be written in Python language utilizing ESM's user-friendly API methods. In fact, the design of ESM is so generic that it can run any ordinary Python scripts (not E-Cell sessions) if the registerJob()- rather than the registerEcellSession() method is used.

In homogeneous parallel computing environments such as shared-memory machines and PC clusters, it is relatively easy to schedule jobs to minimize the

total amount of processing time. In fact, even the simplest first-come-first-served scheduling scheme, which is the current version of ESS implements, works sufficiently well in many cases. However, heterogeneous environments like Globus Grid require more sophistication in scheduling and synchronization because remote computation nodes generally vary unpredictably with respect to their running and response times. Analysis methods that require all jobs to be finished concurrently, e.g. MCA and bifurcation analysis, may suffer from this fundamental weakness of the Grid. However, some other applications of cell biology simulation of an asynchronous nature may be able to overcome this problem. For example, the statistical computation of the stochastic model demonstrated here does not demand that all dispatched jobs be finished before it wraps up the simulation and proceeds to the next step of the analysis of the results. The same strategy might apply to GA parameter estimations if the algorithm is designed to allow evaluation of fitness functions on an incomplete set of individuals in the population. Various kinds of parallel GAs have already been developed[19, 20]. Island-type GAs[21], in which each 'island' has a subset of the whole population and evolves separately with asynchronous exchanges of individuals, is an example of these types of method.

Various kinds of analysis scripts that run on ESM are now under development. These include a program for the GA-based prediction of power-law based (GMA or S-System) gene-networks[22], a sensitivity analysis toolkit based on MCA[23], a bifurcation analysis toolkit that is used to estimate the stability of non-linear models, and Genetic Programming (GP)[24] for prediction of biochemical reactions mechanisms.

3.4 Conclusions

We developed a distributed computing module for E-Cell System, E-Cell Session Manager, or ESM. This software hides the details of job creation and management, and allows the user to write scripts of numerical experiments in Python language that runs transparently on any local, cluster-, and grid computing environments. Using this software, we demonstrated the benefits of distributed computation in computational cell biology research by presenting 2 demonstrations of numerical experiments, (1) parameter estimation using a GA and (2) stochastic simulation of the *E. coli* heat-shock response model. All software described here is available at http://www.e-cell.org/, as part of E-Cell Simulation Environment Version 3, which is OpenSource software under GNU general public license (GPL) version 2.

Acknowledgements

We thank Satya Arjunan and Bin Hu for fruitful discussions. This work was supported by a grant from the Ministry of Education, Culture, Sports, Science and Technology, a grant-in-aid from the 21st Century Center of Excellence (COE) program of Keio University, "Understanding and control of life's function via systems biology". a grant from the New Energy and Industrial Technology De-

velopment and Organization (NEDO) of the Ministry of Economy, Trade and Industry of Japan (Development of a Technological Infrastructure for Industrial Bioprocess Project), a grant from the Leading Project for Biosimulation, Ministry of Education, Culture, Sports, Science and Technology, and a grant from the Japan Science and Technology Agency (JST).

References

1. Li, K.B.: ClustalW-MPI: ClustalW Analysis using Distributed and Parallel Computing. Bioinformatics **19** (2003) 1585–1586
2. Hokamp, K., Shields, D.C., Wolfe, K.H., Caffrey, D.R.: Wrapping up BLAST and Other Applications for Use on Unix Clusters. Bioinformatics **19** (2003) 441–442
3. Schmidt, H.A., Strimmer, K., Vingron, M., Haeseler, A.: TREE-PUZZLE: Maximum Likelihood Phylogenetic Analysis using Quartets and Parallel Computing. Bioinformatics **18** (2002) 502–504
4. Tomita, M., Hashimoto, K., Takahashi, K., Shimizu, T., Matsuzaki, Y., Miyoshi, F., Saito, K. Tanida, S., Yugi, K., Venter, J.C., Hutchison, C.: E-Cell: Software Environment for Whole Cell Simulation. Bioinformatics **15** (1999) 72–84
5. Takahashi, K., Yugi, K., Hashimoto, K., Yamada, Y., Pickett, C.F., Tomita, M.: Computational Challenges in Cell Simulation. IEEE Intelligent Systems **17** (2002) 64–71
6. Takahashi, K., Ishikawa, N., Sadamoto, Y., Sasamoto, H., Ohta, S., Shiozawa, A., Miyoshi, F., Naito, Y., Nakayama, Y., Tomita M.: E-Cell 2: Multi-platform E-Cell Simulation System. Bioinformatics **19** (2003) 1727–1729
7. Takahashi, K., Sakurada, T., Kaizu, K., Kitayama, T., Arjunan, S., Ishida, T., Bereczki, G., Ito, D., Sugimoto, M., Komori, T., Seiji, O., Tomita, M.: E-CELL System Version 3: A Software Platform for Integrative Computational Biology. Genome Informatics **14** (2003) 294-295
8. Arjunan, S., Takahashi, K., Tomita, M.: Shared-Memory Multiprocessing of Gillespie's Stochastic Simulation Algorithm on E-Cell3. 4th International Conference on Systems Biology in St.Louis, MO (2003) poster 253
9. Foster, I. and Kesselman, C.: Globus: A Metacomputing Infrastructure Toolkit. Int. J. Supercomput. Appl. **11** (1997) 115–128
10. Rowe, A., Kalaitzopoulos, D., Osmond, M., Ghanem, M., Guo, Y.: The Discovery Net System for High Throughput Bioinformatics. Bioinformatics **19** (2003) 225–231
11. Oinn, T.M.: Talisman - Rapid Application Development for The Grid. Bioinformatics **19** (2003) 212–214
12. Steavens, R.D., Robinson, A.J., Goble, C.A.: ^{my}Grid: Personalized Bioinformatics of The Information Grid. Bioinformatics **19** (2003) 302–304
13. Konishi, F., Shiroto, Y., Umetcu, R., Konagaya, A.: Scalable BLAST Service in OBIGrid Environment. Genome Informatics **14** (2003) 535–536
14. Kimura, S., Kawasaki, T., Hatakeyama, M., Naka, T., Konishi, F., Konagaya, A.: OBIYagns: A Grid-based Biochemical Simulator with A Parameter Estimator. Bioinformatics **20** (2004) 1646–1648
15. Buyya, R., Abramson, D., Giddy, J.: Nimrod/G: An architecture for a Resource Management and Scheduling System in A Global Computational Grid. In Proceedings of the HPC ASIA'2000, China (2000)

16. Andrew, S., Ishikawa, N., Yamazaki, T., Fujita, A., Kaneko, I., Fukui, Y., Ebisuzaki, T.: Ecell2d : Distributed E-Cell2. Genome Informatics **14** (2003) 288–289
17. Kikuchi, S., Tominaga, D., Arita, M., Takahashi, K., Tomita, M.: Dynamic Modeling of Genetic Networks using Genetic Algorithm and S-system. Bioinformatics **19** (2003) 643–650
18. Hu, B., Tomita, M.: A Stochastic *E. coli* Heat Shock Model Using E-Cell v3. 4th International Conference on Systems Biology, in St. Louis, MO (2003)
19. Cantu-Paz, E.: A Survey of Parallel Genetic Algorithms. Calculateurs Paralleles, Reseaux et Systems Repartis **10** (1997) 141–171
20. Tomassini, M.: Parallel and Distributed Evolutionary Algorithms: A Review. John Wiley and Sons (1999) 113–133
21. Horii, H., Kunifuji, S., Matsuzawa, T.: Asynchronous Island Parallel GA Using Multiform Subpopulations. Lecture Notes in Computer Science (1999) 122–129
22. Voit, E.O.: Computational Analysis of Biochemical Systems. Cambridge University Press (2000)
23. Athel, C.B.: Fundamentals of Enzyme Kinetics. Portland Press (1995)
24. Koza, J.R: Genetic Programming: On the Programming of Computers by Means of Natural Selection. Cambridge, MA: The MIT Press (1992)