CMST 12(1) 79-87 (2006)

DOI:10.12921/cmst.2006.12.01.79-87

COMPUTATIONAL METHODS IN SCIENCE AND TECHNOLOGY **12(1)**, 79-87 (2006)

Design and implementation of flexible, robust and efficient Grid-enabled hybrid QM/MD simulation

Yoshio Tanaka*, Hiroshi Takemiyia, Hidemoto Nakada and Satoshi Sekiguchi

National Institute of Advanced Industrial Science and Technology
Tsukuba Central 2, Umezono 1-1-1, Tsukuba, Ibaraki 305-8568, Japan
**e-mail: Yoshio.tanaka@aist.go.jp*

Abstract: This paper describes the implementation of Grid-enabled hybrid Quantum Mechanics/Classical Molecular Dynamics (QM/MD) Simulation. The Grid-enabled QM/MD simulation is capable of (1) dynamic resource allocation and migration, (2) automatic recovery from faults, and (3) managing large number of CPUs in geographically distributed sites. In the implementation, both GridRPC and MPI are used to complement each other, that is, GridRPC allows dynamic resource allocation and migration and automatic recovery from faults while MPI provides high-performance parallel processing on each cluster. We ran the hybrid QM/MD simulation on an international Grid testbed in the Asia Pacific Region for about 52 days. The experimental results indicated that the hybrid QM/MD simulation could (1) adapt to the dynamic behavior of the simulation and can change the number of CPUs and the number of clusters, (2) adapt to unstable Grid infrastructure and recovers from faults automatically, and (3) manage hundreds to thousands of CPUs on distributed locations, and (4) survive for several weeks without interrupting manual operation. This paper reports on the details of the implementation, strategies for long-run, and experimental results.

Key words: Grid, GridRPC, Grid-enabled application, hybrid QM/MD simulation

1. INTRODUCTION

Grid [1] is regarded as a viable next generation IT infrastructure enabling huge-scale scientific computations which are not yet realized due to the limit of computing capacity of a single system. Hardware resources such as distributed large-scale clusters connected by high-speed networks are enough attractive for metacomputing and the Grid middleware technology that the Globus Toolkit [2] represents is being developed and matured based on past research. Indeed, various experiences on development and evaluation of Grid-enabled applications, such as a climate simulation [3], solving Einstein's equations [4], and *etc*., have been reported and they showed the possibility of the Grid as a ready and realistic infrastructure to be used by real science. There is, however, still a gap between current experimental level and expected production level at which Grid-enabled applications are easily developed and executed on a real Grid infrastructure by daily use. Since the definition of "Grid-enabled application" is still vague, we figure out the following three points as requirements of Grid-enabled applications.

(1) Flexibility

Grid-enabled applications are expected to be executed on a large number of CPUs for long time, but it is not practical to occupy a large-scale single system for long duration. In addition, Grid-enabled applications may change necessary CPUs during their executions on-the-fly. Therefore, Grid-enabled applications should be capable of dynamic resource allocation and migration.

(2) Robustness

Since Grid infrastructure is less stable compared to a single system and large-scale infrastructure and long-run execution increase possibility of encountering faults, Grid-enabled applications should have a capability for error detection and automatic recovery from the faults.

(3) Efficiency

Naturally, Grid-enabled applications should be able to handle hundreds to thousands of CPUs with reasonable performance.

We have developed Grid-enabled hybrid Quantum Mechanics/Classical Molecular Dynamics (QM/MD) simulation, which satisfies the above three requirements, and run the simulation on the PRAGMA [5] Grid Testbed for about 50 days. The implementation is based on a new programming model, which combines GridRPC [6] and MPI [7] so that the simulation can take advantages of both programming model, *i.e*. GridRPC has useful functions for implementation of flexible and robust applications while MPI can manage large number of CPUs by fine-grained data parallelism. In this paper, we report on the details of our

implementation, strategy for and results of the long-run execution of Grid-enabled hybrid QM/MD simulation on the PRAGMA Grid Testbed.

The next Section introduces our target application, hybrid QM/MD simulation. Implementation of Grid-enabled hybrid QM/MD simulation and strategy for long-run are described in Section 3. Section 4 presents long-run experiments on the PRAGMA Grid Testbed and conclusion and future works are described in Section 5.

2. HYBRID QM/MD SIMULATION

This section introduces hybrid QM/MD simulation.

In recent years, various forms of nanostructured materials such as nanoparticles, thin films, and nanophase materials have attracted much attention due to their improved mechanical, catalytic, and opto-electronic properties resulting from their nanometer-size domains [8]. Understanding formation dynamics of nanostructured materials and their resulting unique physico-chemical properties requires accurate dynamic simulations of chemically reacting atoms in these materials. Highly accurate simulations based on energy density-functional theory (DFT) [9] for electronic structures have been used extensively to study chemical reactions in clusters of atoms. Despite considerable progress in formulation, algorithms, and parallel computing techniques, DFT-based dynamic simulations are currently limited to systems containing less than a few hundred atoms due to long computation times. To overcome this limitation, a hybrid quantum-mechanical/classical molecular-dynamics (QM/MD) approach to dynamic simulation of materials on parallel computers has been proposed and developed [10]. The hybrid QM/MD simulation seamlessly embeds accurate QM calculations to handle chemical reactions (below a length scale of 10^{-8} m) within a MD simulation to describe large-scale atomistic processes (up to a length scale of 10^{-6} m) (See Fig. 1).

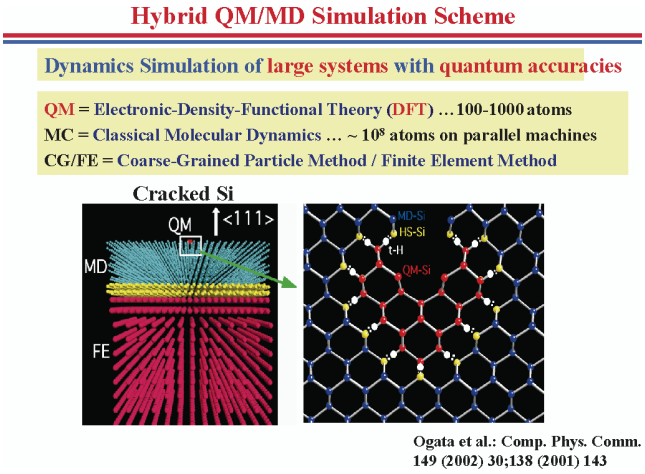

Fig. 1. Hybrid QM/Classical MD simulation

The hybrid QM/MD approach enables dynamic simulations of material processes involving chemical reactions such as oxidation. Modern design of high-performance materials and devices focuses on controlling structures at diverse length scales from atomic to macroscopic, and such multiscale QM/MD simulations are expected to play an important role in scaling down engineering concepts to nanometer scales.

The hybrid QM/MD simulation has the following two important characteristics which indicate that the hybrid QM/MD simulation is appropriate to be Grid-enabled and it is expected to be executed on Grid infrastructure efficiently:

(1) Any number of QM regions can be defined according to the target simulation and application user's interests. Since QM simulations are independent with each other, each QM region can be simulated on different machines. Results of QM simulations are propagated through the MD simulation. In addition, the size of transferred data between MD simulation and QM simulations are very small (usually several tens to hundreds KB) compared to the size of computation for QM simulations (few minutes to few hours).

(2) In this simulation, QM regions will be reconstructed during the simulation according to the behavior of chemical reaction. The reconstruction of QM regions introduces the change of number of atoms in each QM region and the number of QM regions. Processes of the reconstruction consist of two steps, extension of entire QM region and division of the entire QM region into sub QM regions. In the first step, atoms in a MD region are moved to the entire QM region if they are enough closed to the QM region. The extension is necessary to avoid lowering of accuracy of the simulation. In the second step, the entire QM region is divided into sub QM regions if possible. Since the order of QM calculation is $O(N^3)$, it is desired to divide a QM region into several sub QM regions though the division should not affect the accuracy of the simulation. Once the QM regions are reconstructed, the number of QM regions and the number of atoms in each QM region are fixed and necessary computing resources (number of systems and number of CPUs of each system) are decided according to the pre-defined translate rule between the number of atoms and the number of CPUs.

3. IMPLEMENTATION OF GRID-ENABLED HYBRID QM/MD SIMULATION

We have implemented Grid-enabled hybrid QM/MD simulation using GridRPC and MPI. The developed simulation is able to survive few months on unstable Grid infrastructure by its capabilities for fault detection and automatic recovery from faults. In addition, the simulation allows dynamic resource allocation according to the in-

crease of the number of QM atoms. This section describes the details of our implementation and strategy for long-run.

3.1. Hybrid QM/MD Simulation code design

In our implementation, the following three requirements are deemed to be very important to execute a large scale simulation on the Grid for a long time.

(1) Flexibility

When considering the long run simulation, it is unrealistic to expect exclusive access to computing resources over the entire simulation time, because these resources on the Grid are generally shared by many users. One must assume that each cluster will be available only for a part of the simulation time. The code should, therefore, be flexible enough to continue simulation on different target clusters. In addition, as described in the previous section, the number of atoms in each QM region will change dynamically as the simulation proceeds and the region itself will be dynamically created and destroyed, *i.e*. the number of QM regions is changed during simulation on-the-fly. Therefore, the simulation should be able to change the number of utilizing CPUs and the number of clusters on-the-fly according to the requirements by the QM calculations.

(2) Robustness

The Grid is inherently unstable and heterogeneous. To the matter worse, the more resources used for the simulation, the higher the probability of trouble events. The code should, therefore, be robust against the network/cluster trouble events, including long queuing time.

(3) Efficiency

The code should be highly parallelized to reduce the computation time. Although several Grid programming models have been proposed, it is difficult for these models to satisfy all the requirements at the same time. For example, Grid-enabled MPI enables the code to execute efficiently, but it does not provide the mechanism for dynamic resource switching and for fault detection and recovery.

3.2. Overview of GridRPC

GridRPC [6] is a programming model based on a Remote Procedure Call (RPC) mechanism tailored for the Grid. Although when viewed at a very high level, the programming model provided by GridRPC is that of standard RPC plus asynchronous, coarse-grained parallel tasking, in practice there are a variety of features that will largely hide the dynamic, insecure, and unstable aspects of the Grid from programmers. Providing simple, yet powerful, client-server-based frameworks for programming on the Grid, those systems have seen successful usage in various Grid application projects such as a general Monte Carlo simulator of cellular micro-physiology [11], short- to middle-term weather forecasting on an international Grid testbed [3], and Accurate molecular simulation using replica exchange Monte Carlo simulation [12].

A GridRPC system generally consists of the following four components: **Client Component, Server Component, Remote Executable, and Information Service**. Client Components are programs that issue requests for GridRPC invocations. Each component consists of a user's main program and the GridRPC library. A Server Component invokes Remote Executables as described below. Remote Executables perform the actual computation at the Server. Each component consists of a user supplied server-side compute routine, a system generated stub main program, and a system-supplied communication library. Information service provides information for the client component to invoke and to use to communicate with the Remote Executable component. The GridRPC API is in the process of the standardization at the Global Grid Forum [13] and now is published as a proposed recommendation [14].

Two fundamental objects in the GridRPC model are *function handles and session IDs*. The function handle represents a mapping from a function name to an instance of that function on a particular server. The GridRPC API does not dictate the mechanics of resource discovery since different underlying GridRPC implementations may use vastly different protocols. Once a particular function-to-server mapping has been established by initializing a function handle, all RPC calls using that function handle will be executed on the server specified in that binding. A session ID is an identifier representing a particular non-blocking RPC call. The session ID is used throughout the API to allow users to obtain the status of a previously submitted non-blocking call, to wait for a call to complete, to cancel a call, or to check the error code of a call.

3.3. Ninf-G: Programming middleware based on GridRPC

Ninf-G [15, 16] is a reference implementation of the GridRPC API. As of January 2006, Ninf-G provides two versions, Version 2 and Version 4. Ninf-G Version 2 (Ninf-G2) is built on the Globus Toolkit Version 2 and invokes remote processes via Pre-WS GRAM. Ninf-G Version 4 (Ninf-G4) is also tightly coupled with the Globus Toolkit and invokes remote processes via WS-GRAM as well as Pre-WS GRAM. In addition, Ninf-G4 enables remote process invocation via Grid middleware other than the Globus Toolkit, such as UNICORE. In this research, we used Ninf-G2 (to be exact, Ninf-G Version 2.4.0). The software architecture of Ninf-G is depicted in Fig. 2.

The program on the client side, called client component, consists of a user client program and the Ninf-G client library which provides GridRPC API. The client component controls the execution of server programs, called remote executables, through function handles each of which represents a mapping from a function name to an

instance of the function on a particular server. The remote executable is an entity of a callee routine, that is, a stub routine linked with a user server program. The stub is automatically generated by a Ninf compiler from an IDL (Interface Definition Language) file in which the interface of a remote executable is described. It should be noted that no IDL handling is necessary on the client side, as opposed to traditional RPC systems such as CORBA. The client component dynamically gets the interface information in place of using a statically generated client stub. This mechanism is useful for utilizing remote executables developed by other providers. In this case, a user can realize an RPC call merely by inserting Ninf-G functions into a user client program, without worrying about the stub information.

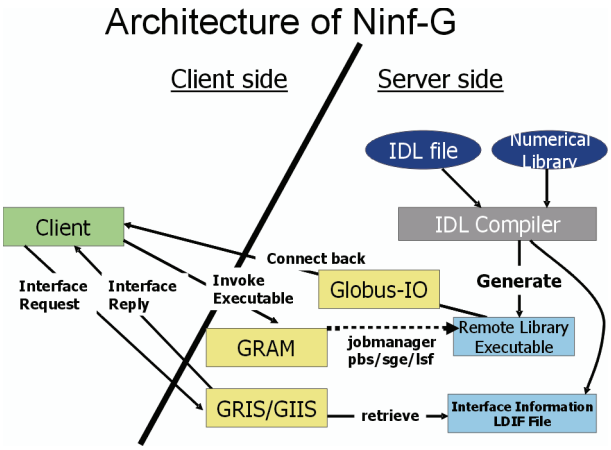

Fig. 2. Architecture of Ninf-G

Ninf-G has the following valuable functions for implementing flexible and robust Grid-enabled simulation.

(1) Dynamic addition / removal of computing resource to / from a resource pool

Available computing resources should not be statically defined at the time of the execution of the simulation. For example, some clusters may become unavailable due to scheduled/unscheduled maintenance and new clusters may join to the experiments during the execution. In Ninf-G, information of available server computers is described in a configuration file which is read by a Ninf-G client in the initialization process. For updates of the server information, *grpc_config_file_read_np()* function and *grpc_handle_attr_set_np()* function can be used. The former function let Ninf-G client re-read the configuration file and the latter function set server attributes when invoking server processes.

(2) Fault detection

Ninf-G provides two functions for fault detection. One is based on the detection of explicit errors such as network disconnection and termination of server processes and the other is based on timeout. Since Ninf-G keeps connections between a Ninf-G client and servers, Ninf-G can detect some faults such as unexpected termination of server processes and network troubles via disconnection of the Ninf-G client and servers. In addition, Ninf-G provides functions to detect timeout. In Grid environment, both computing and networking resources are usually shared by many users and their availability is sometimes unpredictable. For example, submitted jobs may be stacked in the queue and never activated for long time. Decrease of network throughput may cause unexpected long-time data transfer. Ninf-G provides functions to detect such unexpected situations by three kinds of timeout, *invocation timeout, execution timeout, and heatbeat timeout*. *Invocation timeout* is used to detect timeout for remote process invocation, *i.e.* if a remote process is not activated in a pre-defined time, Ninf-G detects a timeout error. *Execution timeout* is used to detect errors on exceeding limits related to the execution. For example, *job_maxWallTime* specifies the maximum wall clock time of the execution and Ninf-G detects an error if the execution time exceeds *job_maxWallTime*. *Heatbeat* is used to send keep-alive messages from a server to the client. If Ninf-G client cannot receive pre-defined number of successive heatbeat signals from a server, it is detected as a heatbeat timeout error.

3.4. Combined GridRPC and MPI strategy

In order to implement a flexible, robust and efficient QM/MD simulation as described in Section 3.1, we combined two programming models, GridRPC and MPI. GridRPC has functions for dynamic execution of server programs, for detection of network/server errors, and for time-outing to avoid waiting for a long time. These functions can be used to satisfy the first two requirements in Section 3.1. On the other hand, MPI is used to realize efficient execution of both MD and QM simulation. In our

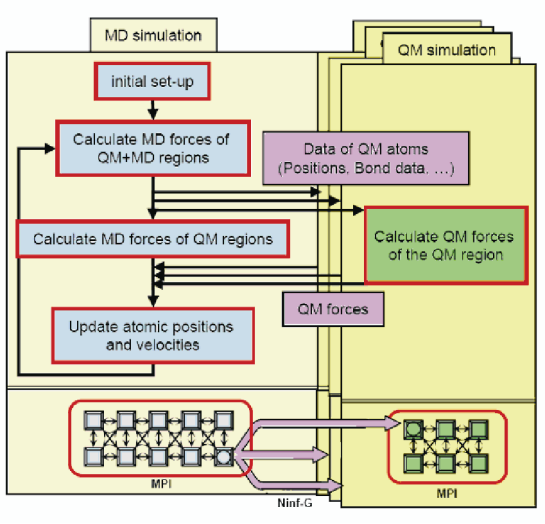

Fig. 3. Execution flow of the QM/MD simulation

implementation, we used Ninf-G and MPICH [17] as reference implementations of GridRPC and MPI, respectively. Ninf-G is able to launch a MPI program on the server, *i.e*. Ninf-G server program can be a MPI program. The execution flow of the code, divided into two parts, MD part and QM part, is illustrated in Fig. 3.

First of all, MD part calculates the behavior of atoms in the entire region in parallel using MPI. Then, it calls remote libraries for QM simulations on different computers using GridRPC. Necessary data for the QM simulation such as positions, velocities, and temperature of QM atoms will be transferred to the remote library as input arguments. Each remote library is implemented as an MPI program which calculates the force in each QM region. If all QM simulations are completed, MD routine collects the simulation results for updating atomic positions and velocities. By repeating the cycle, the simulation proceeds. When some troubles on servers take place, the Ninf-G client stops the execution of the target QM simulation and tries to call the QM simulation on the other (or the same) cluster to resume the simulation.

3.5. Implementation strategy for long-run

The hybrid QM/MD simulation is capable of dynamic resource allocation/migration and automatic recovery from faults which are implemented using functions provided by Ninf-G.

(1) Dynamic migration of QM calculation

Dynamic migration of QM calculation is triggered when (1) Ninf-G client detects errors and server computer becomes unavailable, (2) server computer becomes unavailable due to scheduled maintenance or excess of the maximum execution time, or (3) current server computer becomes inappropriate for use due to the change of the size of QM region. For case (1), Ninf-G client uses various fault detection mechanisms described in Section 3.3. In order to adapt to both cases (1) and (2), Ninf-G client re-reads configuration file in every time step to update resource information on-the-fly. For case (3), Ninf-G client reconstructs QM region in every 10 time steps and Ninf-G client changes utilizing server computers if the number of atoms in a QM region has been increased and Ninf-G client decides the migration.

(2) Strategy of resource selection

In our implementation of resource selection, Ninf-G client selects most stable resources according to the past statistics of the execution. This strategy is based on the observation that troubles on networking and computing resources are not solved in a short time and retry to the same server tends to cause the same error. If the migration is triggered by the excess of the maximum execution time, Ninf-G client excludes the computing resource from candidates.

(3) Automatic recovery from faults

Ninf-G client keeps the results of the QM calculation for each time step by which Ninf-G client can restart QM calculation on a different (or the same) server from the status of at least one time step before the faults. If Ninf-G detects an error, Ninf-G client selects the next computing resources according to the strategy described in the previous paragraph and restart the QM calculation by sending the snapshot data of the previous time step.

(4) Adaptation to the change of computing requirements

As described in Section 3.1, the number of atoms in each QM region and the number of QM regions may change during the execution. Therefore, the number of CPUs used for the QM calculation and the number of clusters used for QM calculations need to be changeable on-the-fly. If the number of atoms in a QM region is increased, MPI processes for the QM calculation may need to be increased as well. In Ninf-G, the number of MPI processes is specified by *GRPC_HANDLE_ATTR_MPI_NCPUS* attributes. When a function handle is initialized, Ninf-G client launches MPI program with the number of MPI processes specified by the attribute. In order to change the number of MPI processes, Ninf-G client need to destroy the current function handle and re-initialize a new function handle with the new number of MPI processes. But re-initialization of a function handle may cause unexpected situation. For example, resources may become unavailable due to other user's jobs. In order to avoid frequent destroy and re-initialize of a function handle, which leads to restarting MPI program on the server, *GRPC_HANDLE_ATTR_MPI_NCPUS* is set to the larger value than actual requirements by the QM calculation when it is started. For example, *GRPC_HANDLE_ATTR_MPI_NCPUS* is set to 32 even if the QM calculation needs 16 CPUs. The number of excessive MPI processes is specified in the configuration file.

(5) Assignment of multiple QM calculation to a single computing resource

If the number of available computing resources becomes less than the number of QM regions, multiple QM calculation is assigned to a single computing resource. In this case, QM calculations are serialized.

4. EXPERIMENTS ON THE PRAGMA TESTBED

In order to evaluate the validity of our code, the long run experiment was conducted on the PRAGMA testbed. This section presents our target simulation, experiments on the PRAGMA testbed and the results of the experiments.

4.1. Target simulation

The target simulation was the simulation of an atomic-scale stick-slip phenomenon which is commonly observed in the surface measurement using an atomic force microscope (AFM) [18] (see Fig. 4).

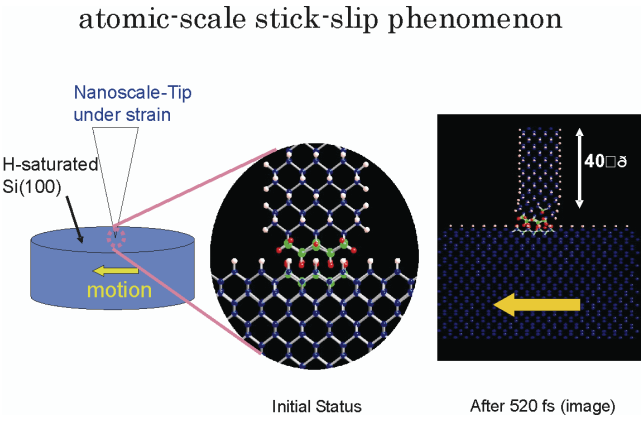

Fig. 4. Atomic-scale stick-slip phenomenon

Atomic scale friction can be modeled using temporary bonds between the outer atoms of two surfaces in contact. As the surfaces slide along each other these bonds are constantly being broken and recreated in stick-slip fashion. We have simulated this stick-slip phenomenon using an AFM in contact with Silicon substrates. Water from the ambient humidity forms a droplet between the tip and the substrate and the substrate is locally oxidized.

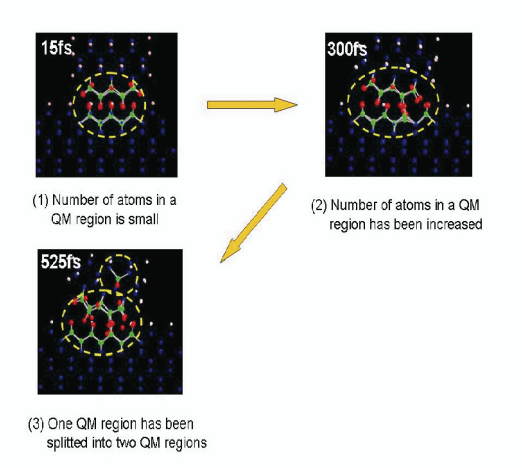

Fig. 5. Stick-slip phenomenon

Initially, we prepared two QM regions for the contact point of the AFM and the substrate. As the substrate slide, the size of the QM region (number of atoms in the QM region) changes (increase or decrease). The number of QM regions may also be changed during the simulation (See Fig. 5).

4.2. PRAGMA Testbed

The simulation was experimented on the PRAGMA Grid Testbed. The Pacific Rim Applications and Grid Middleware Assembly (PRAGMA), founded in 2002, is an institution-based organization, consisting of twenty-seven institutions around the Pacific Rim. PRAGMA is operating a Grid testbed called PRAGMA Testbed, which is build based on "grass roots" approach in which each contributing institutions brings its own resources and share each others resources. As of January 2006, 19 institutions from 5 continents and 13 countries have contributed both physical resources and human resources to the testbed. The software requirements of the PRAGMA testbed are local job scheduler and the Globus Toolkit Version 2 [2]. PRAGMA also deploys additional software such as Ninf-G for GridRPC programming, Nimrod/G [19] for Distributed Parametric Modelling, SCMSWeb [20] for Grid monitoring, and MOGAS [21] for Grid accounting.

Table 1. Resource used in our experiment

	Cluster	Site	Used #CPU	Physical #CPU
1	F32-2	AIST[1]	128	136 (2 × 68)
2	F32-3	AIST	128	264 (2 × 132)
3	P32	AIST	128	256 (2 × 128)
4	M64	AIST	64	256 (4 × 64)
5	ISTBS	U. Tokyo[2]	128	340 (2 × 170)
6	POOL	Tokushima U.[3]	32	47 (1 × 47)
7	ALAB	TITECH[4]	32	60 (2 × 30)
8	Rocks-52	SDSC[5]	16	120 (4 × 30)
9	AMATA	KU[6]	8	12 (1 × 12)
10	ASE	NCHC[7]	8	16 (2 × 8)
11	UME	AIST	8	28 (2 × 14)
12	TGC	NCSA[8]	8	48 (4 × 12)

[1] National Institute of Advanced Industrial Science and Technology, Japan
[2] The University of Tokyo, Japan
[3] Tokushima University, Japan
[4] Tokyo Institute of Technology, Japan
[5] San Diego Supercomputer Center, USA
[6] Kasetsart University, Thailand
[7] National Center for High-Performance Computing, Taiwan
[8] National Center for Supercomputing Applications, USA

In this experiment, we used a part of the PRAGMA testbed and clusters in Japan which are not involved in the PRGMA testbed. Table 1 shows a list of resources in our experiment. As shown in Fig. 5, our target simulation is expected to have one large main QM region and small sub QM regions in the surrounding area of the main QM region. Therefore, we prepared five large scale clusters (F32-2, F32-3, P32, M64, and ISTBS) and seven small scale clusters. F32-2 and F32-3 are reserved and dedicated for our experiments, but we did not make advance reservation for the other clusters.

4.3. Results and discussions

We ran the hybrid QM/MD simulation on the PRAGMA testbed for about 52 days by which we could simulate 386 time steps. In this section, we analyze the experimental

results and discuss how the hybrid QM/MD simulation has been implemented as a Grid-enabled application.

We had to manually stop and restart the simulation two times and the longest continuous execution was about 22 days. The reasons of the two manual restarts were updates of the simulation code and miss-operation. Table 2 shows the summary of the experimental results. In this experiment, re-allocations of QM calculation occurred 192 times. 165 re-allocations were triggered by faults. Table 2 indicates that the allocation to the same computing resource tends to cause failures again. Although our strategy dislikes to select the same computing resource if Ninf-G client detects an error, Ninf-G client had to select the same computing resource for the main QM simulation due to unavailability of other computing resources since they were not reserved for this experiment. 100 times of the failure were occurred successively in this situation.

Table 2. Summary of the experimental results

Reason	Number of occurrence	Allocation to the same computing resource		Allocation to the other computing resource	
		Success	Failure	Success	Failure
Faults in initialization	126	3	104	27	31
Faults during execution	39				
Change of QM region	27	2	0	21	4

Figure 6 shows the change of the number of QM atoms and the number of CPUs used for each QM simulation. The total number of QM atoms is drawn by a solid line which shows that the total number of QM atoms was increased from 117 to 152 in 386 time steps. Box bars show the number of CPUs used for each QM simulation. The different colors and patterns show the number of CPUs used for different QM regions. Pink box bars show the number of CPUs used for main QM simulation. During the first 20 time steps, 128 CPUs were assigned for the main QM simulation, however we encountered successive faults as described in the previous paragraph. At this point, we changed the configuration so that 64 CPUs would be assigned to the main QM simulation. We can observe that sub QM regions were created and destroyed during the simulation and the number of CPUs used for sub QM simulations was automatically decided by the simulation according to the number of QM atoms and pre-defined translate rule between the number of QM atoms and appropriate number of CPUs for simulating the QM region. Figure 6 indicates that the number of QM regions and the number of QM atoms are changed during the simulation and our implementation could adapt to such dynamic behavior by dynamic allocation and release of computing resources.

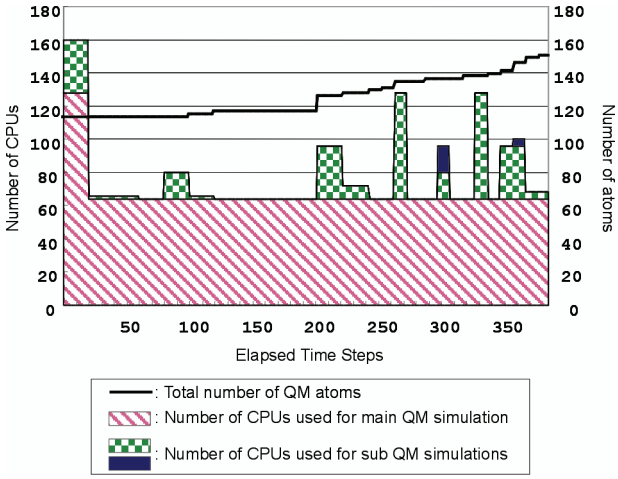

Fig. 6. The number of QM atoms and the number of CPUs used for each QM simulation

The experimental results indicate that the hybrid QM/MD simulation could (1) adapt to the dynamic behavior of the simulation and can change the number of CPUs and the number of clusters, (2) adapt to unstable Grid infrastructure and recovers from faults automatically, and (3) manage hundreds of CPUs on distributed locations, and (4) survive for several weeks without interrupting with manual operation.

5. CONCLUSION AND FUTURE WORKS

We implemented a flexible, robust and efficient Grid-enabled hybrid QM/MD simulation using GridRPC and MPI. The proposed programming model enables dynamic resource allocation and migration, automatic recovery from faults, and utilizing large number of CPUs in geographically distributed sites with high performance. The results of the experiment on the PRAGMA testbed indicated that implementation and strategy for long-run is a practical approach to making Grid a real infrastructure for large-scale scientific applications.

Future works include research and development on scheduling and load balancing of QM calculations, design and implementation of higher-level middleware which hides the complexity of implementation of fault recovery and dynamic resource allocation and migration from application developers.

Acknowledgement

We would like to give thanks to Dr. Shuji Ogata in Nagoya Institute of Technology. He was kind enough to provide his hybrid QM/MD program for our research work and to give advices on our experiments. This work has been conducted as part of ApGrid and PRAGMA activities. We wish to express our gratitude to all of the ApGrid and PRAGMA participants, especially Dr. Kenjiro Taura

(U. Tokyo), Dr. Isao Ono (Tokushima U.), Dr. Kento Aida (TITECH), and PRAGMA testbed administrators in SDSC, NCHC, NCSA, and AIST for their resource contribution for our experiments.

References

[1] I. Foster and C. Kesselman, *The Grid 2: Blueprint for a New Computing Infrastructure*, 2nd edition, Morgan Kaufmann Publishers, Inc. (2004).

[2] I. Foster and C. Kesselman, *Globus: A Metacomputing Infrastructure Toolkit*, Supercomputing Applications and High-Performance Computing **11(2)**, 115-128 (1997).

[3] H. Takemiya, K. Shudo, Y. Tanaka and S. Sekiguchi, *Constructing Grid Applications Using Standard Grid Middleware*, Grid Computing **1**, 117-131 (2003).

[4] G. Allen, T. Dramlitsch, I. Foster, N. T. Karonis, M. Ripeanu, E. Seidel and B. Toonen, *Supporting Efficient Execution in Heterogeneous Distributed Computing Environments with Cactus and Globus*, Proceedings of Supercomputing (2001).

[5] Pacific Rim Application and Grid Middleware Assembly (PRAGMA): http://www.pragma-grid.net/.

[6] K. Seymour, H. Nakada, S. Matsuoka, J. Dongarra, C. Lee and H. Casanova, *Overview of GridRPC: A Remote Procedure Call API for Grid Computing*, Proceedings of the 3rd International Workshop on Grid Computing, 274-278 (2002).

[7] MPI: http://www.mpi-forum.org/.

[8] J. H. Fendler (Ed.), *Nanoparticles and Nanostructured Films*, Wiley-VCH (1998).

[9] W. Kohn and P. Vashishta, *General density functional theory*, In Inhomogeneous Electron Gas, 79-184 (1983).

[10] S. Ogata, E. Lidorikis, F. Shimojo, A. Nakano, P. Vashishta and R. Kalia, *Hybrid finite-element/molecular-dynamics/electronic-density-functional approach to materials simulations on parallel computers*, Computer Physics Communications **138**, 143-154 (2001).

[11] H. Casanova, T. Bartol, J Stiles and F. Berman, *Distributing MCell Simulations on the Grid, Journal of Supercomputing Applications* **15(3)**, 243-257 (2001).

[12] T. Ikegami, H. Takemiya, U. Nagashiima, Y. Tanaka, and S. Sekiguchi, *Accurate Molecular Simulation on the Grid – Replica Exchange Monte Carlo Simulation for C20 Molecule*, IPSJ Transaction of Advanced Computing Systems, **44(SIG11)**, 14-22 (2003).

[13] Global Grid Forum: http://www.ggf.org/

[14] H. Nakada, S. Matsuoka, K. Seymour, J. Dongarra, C. Lee and H. Casanova, *A GridRPC Model and API for End-User Applications*, Global Grid Forum Document, GFD-R.P 52 (2005).

[15] Y. Tanaka, H. Nakada, S. Sekiguchi, T. Suzumura, and S. Matsuoka, *Ninf-G: A Reference Implemantation of RPC-based Programming Middleware for Grid Computing*, Journal of Grid Computing **1(1)**, 41-51 (2003).

[16] Y. Tanaka, H. Takemiya, H. Nakada, and S. Sekiguchi, *Design, implementation and performance evaluation of GridRPC programming middleware for a large-scale computational Grid*, 5th International Workshop on Grid Computing, 298-305 (2004).

[17] W. Gropp, E. Lusk, N. Doss and A. Skjellum, *A High-Performance, Portable Implementation of the MPI Message Passing Interface Standard*, Parallel Computing **22**, 789-828 (1996).

[18] E. S. Snow and P. M. Campbell, *AFM fabrication of sub-10-nanometer metal-oxide devices with in situ control of electrical properties*, Science **270**, 1639-1641 (1995).

[19] D. Abramson, R. Buyya and J. Giddy, *A Computational Economy for Grid Computing and its Implementation in the Nimrod/G Resource Broker*, Future Generation Computer Systems **18(8)**, 1061-1074 (2002).

[20] N. Chalakornkosol and P. Uthayopas, *Monitoring the Dynamics of Grid Environment using Grid Observer*, Poster Presentation in CCGrid 2003 (2003).

[21] Hee-Khiang Ng, Quoc-Thuan Ho, Bu-Sung Lee, Dudy Lim, Yew-Soon Ong and Wentong Cai, *Nanyang Campus Inter-organisational Grid Monitoring System*, Proceedings of Grid Asia Workshop on Grid Computing and Applications, 118-127 (2005).

DR. YOSHIO TANAKA was born in 1965, received his B.E. in 1987, his M.E. in 1989 and his Ph.D.(Eng.) degree in 1995 all in mathematics from Keio University. He was working at Real World Computing Partnership from 1996 to 1999. In 2000, he joined the Electrotechnical Laboratory. In 2001, the Electrotechnical Laboratory was re-organized as the National Institute of Advanced Industrial Science and Technology (AIST). He is currently a team leader of Grid Infraware Team at Grid Technology Research Center, AIST. His current research interests include Grid programming tools, developments and managements of Grid Testbed, and Grid security. He is in particular leading the Ninf project being developed as a reference implementation of current GridRPC GGF standard draft and the Asia Pacific Grid Partnership. He is also the chair of the Asia Pacific Grid Policy Management Authority. He has won the best paper award in the International Conference 1999. He serves an editorial position of the Transactions of Computing Systems of the Information Processing Society of Japan and is a member of sterring committee of Japan Grid Consortium. He is a member of ACM and IPSJ.

HIROSHI TAKEMIYA, born in 1959, earned a bacheler degree in 1984 and a master degree in 1986 from Tohoku University. He works at Hitachi East Japan Solutions Inc. since 1991 and is on loan to National Institute of Advanced Industrial Science and Technology, Grid Technology Research Center as a Senior Researcher since 2002.

HIDEMOTO NAKADA, born in 1967, earned a bacheler degree in 1990, a master degree in 1992, and a doctor degree in 1995, in the field of Comupter Engineering, at the University of Tokyo. He is working for National Institute of Advanced Industrial Science and Technology, Grid Technology Research Center, as a Senior Research Scientist. He had served for Tokyo Institute of Technology, Global Scientific Information and Computing Center, as a Visiting Associate Professor, from 2001 to 2005. He has been one of the co-chairs of the GridRPC-WG in Global Grid Forum.

SATOSHI SEKIGUCHI was born in 1959, received B.S. from Department of Information Science, Faculty of Science, the University of Tokyo in 1982, and M. SE. from University of Tsukuba in 1984 respectively. He joined Electrotechnical Laboratory, Agency of Industrial Science and Technology in 1984 to engage research in high performance and parallel computing widely from the computer architecture, compiler, numerical algorithm, performance evaluation as well as its applications. He served as the deputy director of Research Institute of Information Technology, AIST in 2001, and is currently the founding director of Grid Technology Research Center (GTRC), AIST since 2002. He is a member of IEEE, SIAM, IPSJ, and is a chair of the SIGHPC. He also had been serving as one of the steering committee members of the Global Grid Forum (GGF) till 2003, and now a member of GGF advisory committee. Since the dawn of grid era, he has been one of technology and community leaders, who is in particular one of the PIs of the Ninf project since 1995 being developed as a reference implementation of current GridRPC GGF standard draft, the founder of the Asia Pacific Grid partnership (ApGrid), and chairing Japan Grid Consortium (JpGrid).