動的に計算量が変化する大規模長時間実行 Grid アプリケーションの実現

武宮 博[†,††] 田中良夫[†,††]
中田秀基[†,†††] 関口智嗣[†]

(1) 数千プロセッサ規模の大規模な計算機資源を要する，(2) 数週間から数カ月に及ぶ長期実行を必要とする，(3) 計算の進展に応じて動的に計算量が変化する，という特徴を持つシミュレーションプログラムを GridRPC と MPI を組み合わせるプログラミング手法に基づき Grid 化し，分散配置された並列計算機から構成される Grid 環境上で実行するための手法を提案する．本手法を用いることにより，上記 3 点の特徴を持つ Grid アプリケーションを構築するために必要な，プログラムの柔軟性，頑健性，効率性を実現することが可能となる．本手法の有効性を検証するために，本手法に基づき Grid 化したプログラムを環太平洋 Grid 環境上で長期間継続実行させる実験を試みた．その結果，障害発生や計算量の変化に対応して実行対象計算機やその利用規模を変更しながら最長 20 日以上にわたり大規模 Grid シミュレーションを継続実行することができた．

Implementation of Large-scale Long-run Grid Applications with Varying Problem Sizes

Hiroshi Takemiya,[†,††] Yoshio Tanaka,[†,††] Hidemoto Nakada[†,†††] and Satoshi Sekiguchi[†]

We propose a sustainable adaptive Grid supercomputing paradigm to enable simulations which use thousands of computing resources over tens of continuous days with varying their problem sizes on a Grid of geographically distributed parallel supercomputers. The paradigm "grid-enables" simulation programs based on a hybrid GridRPC + MPI approach for flexibility, robustness, and efficiency. We have achieved an automated execution of an adaptive hybrid simulation over 20 days on the Pan-Pacific Grid testbed, in which the number of processors changes dynamicallly on demand and resources are allocated and migrated dynamically according to the availability of computing resources and the growth of problem sizes.

1. はじめに

1995 年に Grid の概念が提唱されて以来，Global Grid Forum（GGF）等の活動を通して Grid 基盤や Grid ミドルウェアの持つべき機能に関して広範な議論が行われ，インタフェースの標準化，実装が推し進められてきた．このような基盤部分に関する研究開発の進展にともない，大規模な計算機資源を必要とするバイオサイエンス，ナノサイエンス，原子力物理等の応用分野でも Grid に対する大きな期待が寄せられてきている．しかしながら，Grid 上で大規模アプリケーションを長時間にわたり実行するためのアプリケーションの実装方法，障害発生時あるいは利用中の計算機が利用不可となった場合の対処方法，等が確立されていないため，これらの分野において Grid 技術を適用したアプリケーションを構築した事例はまだ少数にとどまっている．また，アプリケーションの種類も Grid 上の計算機資源を利用して多数の独立なタスクを処理するもの[6),7)] や，Grid-enabled MPI[10)] を用いて Grid 上の複数の計算機資源上に静的に割り付けられたプログラムを互いに通信させることにより処理を行うもの[8),9)] が大半である．Grid アプリケーションの開発，実行を加速し，Grid を実用レベルの技術とするためには，より多くの種類のアプリケーションの Grid 上での実行可能性を検討し，開発および実行の方法論を明確化する必要がある．

† 産業技術総合研究所
National Institute of Advanced Industrial Science and Technology
†† 科学技術振興機構
CREST, Japan Science and Technology Agency
††† 東京工業大学
Tokyo Institute of Technology

我々は，Grid 上での実行に適した新たな種類のアプリケーションとして，(1) 数千プロセッサ規模の大規模な計算機資源を要する，(2) 数週間から数カ月に及ぶ長期実行を必要とする，(3) 計算の進展に応じて動的に計算量が変化する，といった特徴を持つアプリケーション（以下，大規模長時間実行アダプティブアプリケーション，Grid 化されたものを大規模長時間実行アダプティブ Grid アプリケーションと呼ぶ）に着目する．我々の想定する大規模長時間実行アダプティブ Grid アプリケーションは，地理的に分散した並列計算機上で複数の細粒度並列プログラムが互いに疎に連携しつつ，状況に応じて動的に計算規模を変更しながら総計数千プロセッサを用いて長期間の計算を行う．

このようなアプリケーションの実行には，1) アプリケーションの要求あるいは Grid を構成する計算機資源の状態に対応し実行対象計算機を変更していく柔軟性，2) 発生する障害を検知，復旧し処理を継続していく頑健性，3) 多数の計算機資源を利用してスケーラブルな処理を実現する効率性，が要求される．これまで採用されていた Grid 化手法では，これらの要求にすべて対応することは困難である．

このような特徴を持つアプリケーションとして，材料工学，バイオ工学，構造解析等の分野で行われているアダプティブな連成シミュレーションがあげられる．ここでいう “アダプティブ” とは，計算対象としている系の状況にあわせて必要な領域において詳細な計算を行う手法を意味し，計算量を抑えつつ高精度な結果を得るために広く用いられている手法である．また，連成シミュレーションは，流体計算と構造計算，あるいは量子力学に基づく計算手法と古典力学に基づく計算手法等，複数の計算原理に基づくシミュレーションを連携させてシミュレーションを行う手法である．このようなアプリケーションは計算アルゴリズムの進展，計算機能力の向上にともない新しく提唱されてきたもので，従来不可能であった高い精度でのシミュレーションを可能にすることから産業界を含め高い要求があるが，膨大な計算機資源を必要とすることからその利用が制限されていた．これらを Grid 化するための方法論を明確にすれば，Grid アプリケーションの開発，実行の加速が期待される．

我々は，長期にわたり Grid 環境上で実行される大規模アプリケーション開発のための現実的なプログラミング手法として，GridRPC と MPI を組み合わせる手法を提案した[3)]．本手法は，Grid 上に散在する並列計算機に対し，MPI によって並列化されたシミュレーションプログラムを GridRPC により動的に起動し，クライアントプログラムを介して連携させるものである．GridRPC の持つ動的な遠隔プログラム実行機能および障害検知機能と MPI の持つ効率的な並列実行機能を相補的に組み合わせることにより，上記 3 つの要件を満たすアプリケーションの実装を可能とする．本手法に基づき典型的な大規模長時間実行アプリケーションを Grid 化し，日米に散在するクラスタから構成される Grid 環境上で最大 1,793CPU を利用して実行することで，アプリケーションが実際に大規模実行可能であることを示した．さらに，プログラムの実行中に人為的に障害を発生させ，本手法が種々の障害を検知可能であることを示した．

本稿では，これまでに得られた成果をふまえ，GridRPC と MPI を組み合わせるプログラミング手法に基づき柔軟性，頑健性，効率性の要件を満たす大規模長時間実行アダプティブ Grid アプリケーションの実装および実行に関する具体的な方法論を述べる．我々の目的は，上記のような特質を持つ大規模 Grid アプリケーションの長時間実行を期待しているアプリケーションユーザに対し，実装の際の指針を示し，考慮すべき事項に関する情報を提供することにある．

また，本手法を用いて典型的な大規模長時間実行アダプティブアプリケーションの 1 つである Adaptive Hybrid QM/MD シミュレーションコードを Grid 化し，環太平洋 Grid テストベッド上で長期にわたる実験を行った．実験結果に基づき実装機能の有効性を評価するとともに，得られた知見をまとめる．

2. 大規模長時間実行アダプティブ Grid アプリケーション

1 章で述べたように，我々の想定する大規模長時間実行アダプティブ Grid アプリケーションは，地理的に分散した並列計算機上で複数の細粒度並列プログラムが互いに疎に連携しつつ，状況に応じて動的に計算規模を変更しながら総計数千プロセッサを用いて長期間の計算を行うものである．このようなアプリケーションの実行を可能とするためには，以下の 3 つの特質が要求される．

(1) 柔軟性

- 動的割付け：必要に応じて動的に計算機資源上で起動され，実行されなければならない．
- マイグレーション：動的に利用する計算機資源を変更しつつ実行を継続できなければならない．
- 計算規模の変更：問題規模の変化に対応して利用する計算機資源の規模を変更できなければならない．
- 計算機資源の追加，変更：当初想定していなかっ

た計算機を利用対象ターゲットとして新たに計算機資源を追加したり，運用管理スケジュールに応じて利用可能プロセッサ数を変更したりする等，計算機の利用に関する情報を動的に変更できなければならない．

(2) 頑健性

- 障害の検知：シミュレーション実行中に発生する種々の障害を検知できなければならない．計算機資源のシステムダウン，ネットワークの切断といったあらわな障害（以後，陽的障害）だけでなく，計算機資源が他のユーザによって利用されているために実行を長時間待たなければならないといったことも広い意味での障害（以後，陰的障害）ととらえ，対応できなければならない．
- 障害の復旧：発生した障害から自動的に復旧できなければならない．

(3) 効率性

- 計算機資源の効率的管理：地理的に分散した数千台規模の計算機資源を効率的に管理できなければならない．
- 通信，計算の効率的実行：個々の細粒度並列シミュレーションおよびそれらの連携を効率的に実現できなければならない．

3. 大規模長時間実行アダプティブアプリケーションの Grid 化

3.1 GridRPC + MPI によるプログラミング手法

我々が提案している GridRPC と MPI を組み合わせるプログラミング手法は，MPI によって並列化された遠隔実行プログラムを GridRPC を用いて動的に起動し，連携する．本手法を大規模長時間実行アダプティブアプリケーションに適用した場合に実現されるシステム構成例を**図 1** に示す．アプリケーションを構成する複数の細粒度並列シミュレーションプログラムは並列計算機上に各々動的に割り付けられ，クライアントプログラムを中心として疎に連携する．このような構成は，アプリケーションの特質に適合しているだけでなく，分散配置された並列計算機から構成される Grid のトポロジにもうまく適合している．

本手法は上記の要件を以下のように支援する．まず，GridRPC の持つ動的な遠隔実行プログラム起動機能と多様な障害検知機能を利用してプログラムの柔軟性および頑健性の実現を支援する．また，MPI の持つ遠隔実行プログラム内での効率的な並列処理機能と GridRPC の遠隔実行プログラム管理機能を組み合わせることにより効率性の実現を支援する．

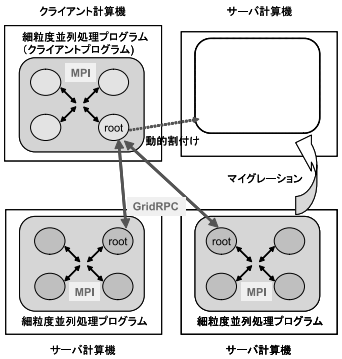

図 1 GridRPC+MPI を用いたプログラミング手法により実現されるシステム構成例

Fig. 1 Example of a system configuration realized by the programming approach which combines GridRPC with MPI.

既存のプログラミング手法，たとえば Grid-enabled MPI あるいは GridRPC のみを用いた場合，上記要件をすべて満足することは困難である．Grid-enabled MPI の場合，静的なシステム構成しかとれない，耐故障性が低い，という問題点が存在する．一方，GridRPC の場合，多数の計算機資源を単一のクライアントが管理しなければならないため，効率的な管理が困難である．本手法では GridRPC のクライアントは MPI によって生成されるプロセスグループの単位で計算機資源を管理するため，管理対象の個数を少なく抑えることができる．

3.2 Grid 化方針

2 章で述べた機能要件は，GridRPC および MPI により提供される機能だけで容易に実現されるものではない．より上位のレイヤにおいて実現される機能と統合することによってはじめて実現されるものである．Adaptive Hybrid QM/MD シミュレーションコードを対象に 2 章で述べた機能要件を実現するにあたり，我々は上位レイヤにおいてどのような機能を実現すべきかを検討した．検討に際しては，GridRPC/MPI により提供される機能，対象アプリケーションによらず共通して必要な機能，およびアプリケーション固有の知識に基づいて実装される機能に分類し，共通機能をアプリケーションと GridRPC/MPI の中間のレイヤ（スケジューラ）にまとめることにした．

3.3 Ninf-G 提供機能

まず最初に，GridRPC が提供する機能に関してまとめる．本節では，GridRPC の参照実装の 1 つである Ninf-G [2] について，柔軟性の実現および頑健性の実現を支援する機能を中心に説明する．

Ninf-G は，Grid 基盤として最も普及している

Globus 上に構築された GridRPC システムで，大規模な Grid アプリケーションを容易に開発するための機能提供を目的としている．また，複数サイトにまたがる計算機の接続等，Grid 特有の問題にも対応しており，クライアントとサーバの通信路を確立する際には，サーバからクライアントに対して接続することにより，クライアント計算機の存在するサイトさえ外部からの接続を許可すれば複数サイトの計算機を用いた計算が可能となっている．計算ノードにプライベート IP アドレスが割り当てられている場合でも NAT 等の機能を用いれば外部に対して接続可能であるし，Firewall の設定においても外部への接続が許可されていれば問題ない．

Ninf-G では，柔軟性実現および頑健性実現を以下のように支援している．

(1) 柔軟性の実現支援

- 遠隔実行プログラムの動的起動，終了

リモート計算機上に存在する遠隔実行プログラムはリモートオブジェクトあるいは遠隔実行関数として抽象化し，必要に応じて動的に起動，終了する．

- 属性の動的追加，変更

シミュレーション実行途中での動的な計算機資源の追加，変更を支援するために，計算機名，利用バッチキュー名，利用プロセッサ数，あるいは後述する各種タイムアウト値等，計算機に関する種々の属性を動的に追加，変更可能としている．これらの属性は，利用者が configuration file と呼ばれるファイルに記述することにより設定される．この configuration file の再読み込みを行う `grpc_config_file_read_np()` 関数，あるいは遠隔実行プログラム起動時に属性を動的に設定する `grpc_handle_attr_set_np()` 関数を提供することで，動的な計算機資源の追加，変更を支援する．

(2) 頑健性の実現支援

Ninf-G は RPC 実行における障害検知のために 2 種類の機構を提供している．

- 陽的障害検知

陽的な障害検知とは，クライアントプログラムと遠隔実行プログラム間のネットワーク接続の切断に基づく障害検知を意味する．遠隔実行プログラムが起動されると，クライアントプログラムとの間で接続が確立され，それを維持しながら RPC を実現する．ネットワークあるいは計算機の障害が発生すると，この接続が切断され，それにより Ninf-G は遠隔実行プログラムの異常を検知する．

- 陰的障害検知

陰的な障害検知とは，RPC の各動作（初期化，実行，終了等）に関する時間制限（タイムアウト）に基づく障害を意味する．陰的な障害の検知のために，Ninf-G は利用者が configuration file に設定した種々のタイムアウト時間を超えて処理が終了しない場合にエラーを返す．具体的には，起動時間タイムアウト，ハートビートタイムアウト，実行時間タイムアウトの 3 種類のタイムアウト時間が設定可能となっている．起動時間タイムアウトは，クライアントプログラムが遠隔実行プログラムの起動要求を行った後，サーバ計算機上でバッチシステムを介して実際に起動が行われるまでの時間に関する制限である．ハートビートタイムアウトを設定すると，遠隔実行プログラムは一定間隔でハートビートメッセージをクライアントプログラムに送付する．利用者の設定した制限時間を超えてハートビートメッセージを受信できない場合，クライアントプログラムは遠隔実行プログラムに障害が発生したと見なしてエラーを返す．実行時間タイムアウトに関してはさらに細かな設定が可能となっており，利用者は計算機の継続実行時間制限（`job_maxTime`, `job_maxCpuTime`, `job_maxWallTime`），および 1 回の RPC 実行時間制限（`session_timeout`）を個別に設定することができる．利用するグリッド環境やアプリケーションの性質に応じ，タイムアウト値は利用者により適切な値に設定される必要がある．Ninf-G が提供するタイムアウト検知機能を利用することにより，陰な障害の検知およびそれからの復旧機能をアプリケーションに実装し，頑健なアプリケーションを実現することができる．タイムアウト機能を利用するための configuration file 記述例を**図 2** に示す．

```
<FUNCTION_INFO>
   hostname cluster_A
   session_timeout  18000
   funcname QMMD/QM
   path /scratchb2/7/takemiya/Hybrid_QMMD/_stub_QM
</FUNCTION_INFO>

<SERVER>
   hostname cluster_A
   workDirectory
/scratchb2/7/takemiya/Hybrid_QMMD_SIMOX_Current
   heartbeat  60
   heartbeat_timeoutCount  20
   argument_transfer wait
   job_startTimeout 900
   job_stopTimeout 300
   redirect_outerr true
   job_queue res_5
   jobmanager jobmanager-lemieux-pbs
   job_maxWallTime 90
</SERVER>
```

図 2 configuration file 記述例
Fig. 2 Example of a configuration file.

3.4 スケジューラの設計

スケジューラで実現されるべき機能として，(1) 利用対象となる計算機資源の状態を管理し，利用可能な計算機資源の中から実行対象となる計算機を選択する計算機選択機能，および (2) 障害発生等の理由から中断したシミュレーションを継続するために必要なデータを遠隔実行プログラムに対して与え，遠隔実行プログラムの状態を回復する状態回復機能，の 2 種類を考慮した．これらの機能と Ninf-G により提供された機能を組み合わせることによって柔軟性，頑健性に関する機能要件が実現される．たとえば，障害の復旧は，計算機選択機能を利用して代替計算機資源を選択し，Ninf-G の機能を用いて動的に遠隔実行プログラムを起動するとともに，状態回復機能を用いてシミュレーション途中の状態を回復することで実現される．

両機能の設計に際しては，以下の項目を検討した．

- 大規模計算機の利用に関しては，利用可能バッチクラス，CPU 数，ジョブ投入本数，計算時間，利用可能期間等，一般に様々な制約がともなう．スケジューラは，これらの制約を満足しながら計算機を選択しなければならない．しかし，アプリケーションに対してはこれら細かな制約を可能な限り隠蔽する．
- さらに，これらの制約は動的に変化する可能性がある（たとえば，計算実行中に計算機のメンテナンスが発生し，利用可能期間が変化する等）．スケジューラも制約の変化に対応できるようにする．
- 計算機の選択手法はアプリケーションや計算環境の特徴に依存して種々のものが考えられるが，今回の実験では以下の 2 つの手法を採用する．1 つは，割り付けられた計算機をなるべく継続利用する方法（継続利用重視手法）である．障害が発生したときも再利用を試みる．この手法は，大規模な計算機資源を利用する場合，代替計算機資源を動的に確保することは困難であり，仮に存在しても大規模シミュレーションのマイグレーションのコストが大きいということを想定している．もう 1 つの選択方法は，すべての利用可能な計算機に対し過去の実行履歴を保持しておき，最も障害発生の少ない計算機を優先して選択する（安定性重視手法）．この手法は計算機の安定性を重視した手法である．
- 状態の回復が必要となる契機は，障害発生時，マイグレーション時，および問題規模が変化したときである．それらの契機が発生した際には，スケジューラ内で自動的に状態回復が行われるようにし，アプリケーションが意識する必要のないようにする．
- 一般に，時間進展型の連成シミュレーションコードは，初期パラメータを読み込む初期化処理および実際のシミュレーション処理から構成され，シミュレーション処理は時間進展ループ内で繰り返し呼ばれる構造となっている．このような特性に着目し，アプリケーションから渡される初期パラメータデータおよびシミュレーション処理用入力データを保持しておき，状態回復に利用することにする．

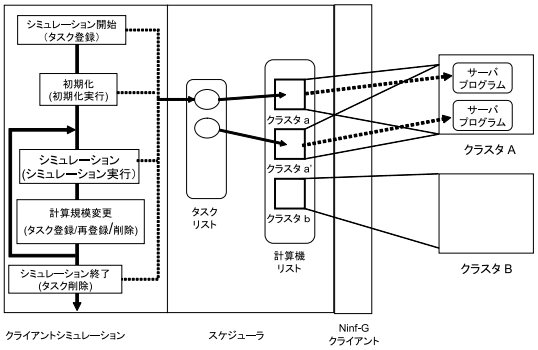

図 3 スケジューラの構成
Fig. 3 Structure of a scheduler.

3.5 スケジューラの実装

前節の検討結果に基づき，スケジューラを以下のように実装した（**図 3** 参照）．スケジューラは，利用者から提供される計算機情報とアプリケーションから提供されるタスク情報を，計算機リストおよびタスクリストとして管理することにより計算機選択機能および状態回復機能を実現する．計算機リストに登録される計算機は，その計算機のアドレス等の属性および利用に関する制約情報を持つ実体として定義される．利用者は自分が利用する予定の計算機情報をシミュレーション実行前に設定しておく．スケジューラは起動時にそのファイルを読み込むことで計算機リストに利用対象計算機を登録する．利用に関する複雑な制約の指定を容易にするために，利用対象計算機を仮想的に複数の計算機として設定可能としている．各仮想計算機は指定された CPU 数と単一のジョブキューを持つものと見なされる．これにより，複数のバッチキューを持つ計算機を複数の仮想計算機として利用したり，大規模計算機を多数の小規模計算機として利用したりすることが可能となる．**図 4** に制約情報設定ファイルの記述例を示す．

タスクリストには，実行されるシミュレーションが登録される．シミュレーションは実行に必要な CPU 数のほか，障害等で中断したシミュレーションを継続するために必要な初期パラメータおよびシミュレーション実行時の入力データを保持する実体として定義され

```
<HOST>
NAME cluster_A
ADDR aaa.asc.ac.jp
FROM 2005/10/7/9/0/0
TO   2005/11/3/12/0/0
MAX_AVAIL 604800
CPU_MAX 64
</HOST>
```

図 4 制約情報設定ファイル記述例．利用可能期間，最長継続利用時間，最大利用可能 CPU 数が記述されている
Fig. 4 Example of a host file. Restriction on the access period, maximum usage time, and maximum number of CPUs are described.

る．保存された初期パラメータおよび入力データは，後述のように障害等が発生した場合に遠隔実行プログラムの状態回復に利用される．

計算機の選択およびシミュレーションの状態回復は以下のように実現される．計算機の選択では，スケジューラはタスクリストに登録されたシミュレーションの実行に必要な CPU 数と計算機リストに登録された計算機の持つ利用に関する制約情報を検索し，その時点で利用可能かつシミュレーション実行に十分な CPU 数を持つ計算機を候補として選出する．最終的な利用対象計算機は，継続利用重視手法あるいは安定性重視手法に基づいて候補の中から決定される．

状態回復では，タスクリストに登録されたシミュレーションが保持する初期化パラメータおよびシミュレーションの入力データを利用して Ninf-G により初期化処理およびシミュレーションの実行を順次行う．

本スケジューラを用いて構築した大規模長時間実行アダプティブ Grid アプリケーションの処理の流れを示す．図 3 に示すように，アプリケーションは時間進展型連成シミュレーションコードの特性にあわせて，1) シミュレーション開始時，2) 初期処理時，3) シミュレーション実行時，4) 計算規模変更時，5) シミュレーション終了時，にスケジューラおよび Ninf-G と連携して以下のような処理を行う．

シミュレーション開始時に，アプリケーションは実行すべきシミュレーションをスケジューラに登録する．登録に際し，アプリケーションは実行に必要な CPU 数を指定する．スケジューラはシミュレーションをタスクリストに登録するとともに，起動時に読み込んだ計算機情報をもとに作成した計算機リストを検索し，適した計算機を選択する．計算機が決定すると，Ninf-G を利用して遠隔実行プログラムをリモートオブジェクトとして実行対象計算機上に動的に起動する．

初期処理時およびシミュレーション実行時には，アプリケーションは必要なデータをスケジューラに渡して処理を依頼する．スケジューラは渡されたデータをタスク内に保存するとともに，Ninf-G を用いて遠隔実行プログラムによる処理を実現する．

計算規模の変化にともない必要な CPU 数が変化した場合や新たなタスクが生成された場合，アプリケーションはスケジューラに CPU 数の変更や新たなタスクの追加/削除を依頼して新しい初期化データを渡す．スケジューラは新たに計算機を選択し，遠隔実行プログラムを起動するとともに初期化処理を行う．

最後にシミュレーションを終了する際には，アプリケーションはスケジューラにシミュレーションの削除を依頼する．スケジューラはタスクリストから対象となるシミュレーションを削除するとともに，Ninf-G を用いて遠隔実行プログラムの終了を行う．

スケジューラが主体となって行う処理としては，障害の復旧および利用に関する制約の変化への対処があげられる．Ninf-G の機能を利用して障害を検知すると，スケジューラは新たな計算機を選択し，Ninf-G を用いて遠隔実行プログラムを起動するとともに，状態回復機能を用いて中断したシミュレーションの状態を回復する．

アプリケーションの実行中に利用に関する制約が変化すると，利用者は計算機情報設定ファイルを随時変更する．スケジューラはシミュレーション実行前に毎回設定ファイルを再読み込みし，実行対象計算機の利用可能性を確認する．制約の変化により利用が不可能になった場合には，新たな計算機を選択し，障害の復旧時と同一の手順で中断したシミュレーションを継続する．

4. Adaptive Hybrid QM/MD シミュレーションコードの Grid 化

本章では，大規模長時間実行アダプティブアプリケーションの例として用いる Adaptive Hybrid QM/MD シミュレーションコードの概要を説明した後，GridRPC と MPI を組み合わせるプログラミング手法を用いた Adaptive Hybrid QM/MD シミュレーションコードの Grid 化に関する詳細を述べる．

4.1 Adaptive Hybrid QM/MD シミュレーションコード

Adaptive Hybrid QM/MD シミュレーションコードは，材料工学分野における次世代電子デバイスやマイクロマシンの設計において，亀裂発生等ナノスケール特性を解析するために開発されたプログラムである．本コードは，Density Functional Theory（DFT）に基づく詳細な QM（Quantum Mechanics）シミュレーションと経験的原子間ポテンシャルを用いた古典 MD

(Molecular Dynamics) シミュレーションを組み合わせる．高精度ではあるが長時間の計算を要する QM シミュレーションを用いて，亀裂発生の可能性のある領域等，詳細な解析が必要な領域（以後，QM 領域，QM 領域に含まれる原子を QM 原子と呼ぶ）の原子の振舞いを計算し，MD シミュレーションの結果を補正することにより，現実的な時間内に大規模系の挙動を精度良く解析可能としている．

MD，QM 両シミュレーションは各々細粒度の並列処理により計算を行う．特に QM シミュレーションの計算量は膨大で，典型的な系に対し数百〜数千台規模のプロセッサを用いて 1 タイムステップの計算に 10^3 秒程度，シミュレーションの完了には数週間から数カ月を要する．QM シミュレーションと MD シミュレーション間では QM 原子の位置，速度，力といったデータの交換が必要であるが，QM 原子の数を N とすれば，計算量が〜O(N^3) であるのに対して通信量は〜O(N) で抑えられることから，両者の連携は疎である．また，本シミュレーションでは，系の挙動に従い計算実行中に QM 領域の再定義が行われ，領域の数や QM 原子の数が変化する．その結果，動的に計算量が変化する．したがって，Adaptive Hybrid QM/MD シミュレーションコードは，大規模長時間実行アダプティブシミュレーションの特徴を満足している．

図 5 に本シミュレーションのアルゴリズムを示す．系を構成する原子の位置と速度が与えられると，MD シミュレーションにより系全体の振舞いが計算される．次に，QM 原子の振舞いが QM，MD 両シミュレーションにより独立に計算され，系全体に関する結果を補正する．これらの計算を規定回数繰り返して系の状態が変化すると，QM 領域の再定義が行われる．

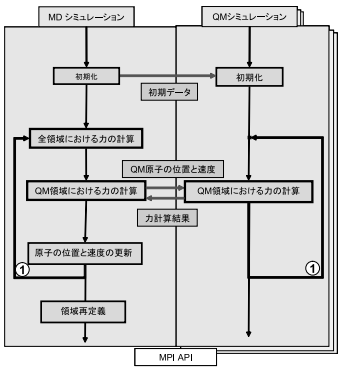

図 5 Adaptive Hybrid QM/MD シミュレーションコードの処理の流れ．図中 ① はアプリケーションの時間進展ループを表す
Fig. 5 Flow chart of the Adaptive Hybrid QM/MD simulation code. ① denotes a time evolution loop of the application.

再定義は QM 領域の拡張および分割の 2 段階で行われる．QM 領域の拡張では，領域の近傍に存在する原子の位置を調べ，領域内の原子に十分近接している場合は領域に加える．これは，時間進展にともなう系の状態変化による計算精度の悪化を防ぐためである．一方，QM 領域の分割では，各領域の分割可能性を調べる．上述のように QM シミュレーションの計算量は〜O(N^3) であるため，領域を分割して個々の領域に含まれる原子の数を少なくすれば総計算量が減少し，計算時間を節約できるからである．この再定義が終了すると，各 QM 領域に含まれる原子の数に基づき必要とされる計算機の CPU 数が決定される．原子の数と必要とされる CPU 数の換算は，ユーザがあらかじめ設定しておいた原子の数と CPU 数の対応表に基づき行われる．

原プログラムは MPI を用いて並列化されていたため，CPU 数を動的に変更することができなかった．したがって，時間進展ループの実行回数を制限し，CPU 数を変更する必要が発生すると計算を終了し，手動で再起動を行う構造となっている．

4.2 Adaptive Hybrid QM/MD シミュレーションコードの Grid 化

Adaptive Hybrid QM/MD シミュレーションコードの Grid 化に際しては，まず QM シミュレーションと MD シミュレーションの処理を分離し，QM シミュレーションにおける初期化処理とシミュレーション処理を GridRPC の遠隔実行プログラムとして実装した．また，MD シミュレーションの処理をクライアントプログラムとして実装する際には，前章で述べたスケジューラを利用した．また，動的な計算量の変化に対応するために，原コードでは時間進展ループ終了時に行っていた QM 領域の再定義処理を時間進展ループ内で行うようにした．

Grid 化された Adaptive Hybrid QM/MD シミュレーションコードの処理の流れを**図 6** に示す．MD シミュレーションプログラムが起動されると，各 QM シミュレーションをスケジューラに登録し実行に必要なプロセッサ数を通知する．スケジューラは計算機を選択した後，対象計算機上に QM シミュレーションを起動する．MD シミュレーションプログラムは，初期化およびシミュレーション実行用スケジューラ API を介して QM シミュレーションプログラムに入力データを転送し，結果を受信する．結果に基づいて系の状態を更新した後，領域の再定義が行われる．領域数や領域の大きさに変更があった場合，スケジューラに必要な CPU 数を通知して実行対象計算機の変更を依頼す

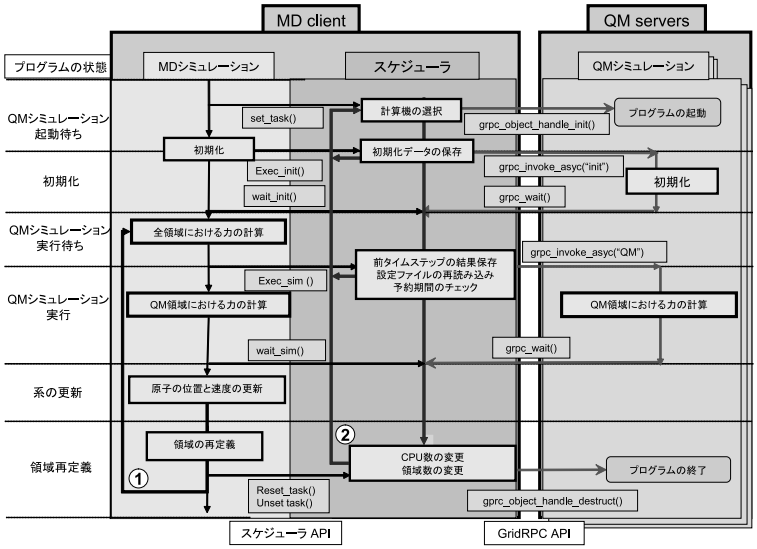

図 6 Grid 化 Adaptive Hybrid QM/MD コードの処理の流れ．
図中 ① はアプリケーションの時間進展ループ，② は障害発生時の状態遷移を表す
Fig. 6 Flow chart of the grid-enabled Adaptive Hybrid QM/MD code. ① denotes the time evolution loop of the application, and ② denotes the flow of fault recovery.

る．障害発生時の復旧はスケジューラ内で自動的に行われ，MD シミュレーションプログラムは意識しない．

本コードの Grid 化に際してアプリケーションレベルで留意した点は，中断したシミュレーションの継続に必要なデータの転送量削減である．障害発生等の理由により実行対象計算機が変更される場合に備えて，QM シミュレーションの継続実行に必要なデータを毎回クライアント計算機上に転送しておく必要がある．しかし，QM シミュレーション継続に必要なデータの量は膨大で，典型的な系のシミュレーションにおいて 1 GB 程度のデータを毎回転送しなければならない．このような大量のデータ転送はシミュレーションの実行性能を大きく低下させる．上記データの大部分は QM シミュレーションにおける Self Consistent Field 計算（SCF 計算）の初期値推定に用いられるため，実行計算機が変更された場合には乱数をもとに SCF 計算の初期値を推定することとし，上記データを不要とすることで転送量を低減させることにした．その結果，転送量は数 MB 程度に抑えられるようになった．ただし，乱数に基づき SCF 計算の初期値を推定すると経験上 SCF 計算に時間を要してしまう．そのため，SCF 計算の初期値推定に必要なデータをいったん実行対象計算機上のディスクに保存しておき，実行計算機が変更されない限りそのデータを再読み込みすることにより推定を行うようにし，乱数に基づく SCF 計算の初期値推定は実行計算機の変更時にのみ実行されるようにした．

5. 長時間実行実験

5.1 実験概要

環太平洋に分散配置されたクラスタを利用して 2 種類の実験を行った．実験の違いは計算機資源の選択手法であり，(1) 継続利用重視手法，(2) 安定性重視手法を各々採用した．

(1) の実験では，SIMOX（Separation by Implantion of Oxygen）と呼ばれる IC チップ製造技術のシミュレーションを行った．SIMOX は，高速に加速された酸素原子をシリコン基盤に入射し SIO_2 絶縁層を形成する技術である．シミュレーションでは，シリコン基板上に入射位置の異なる 5 つの酸素原子を配置し入射させ，酸素原子と基盤内のシリコン原子の挙動を調べる．本シミュレーションでは，個々の酸素原子およびその周辺のシリコン原子が QM 領域として動的に定義され，酸素原子がシリコン基盤の内部に侵入するにつれて QM 領域に含まれる原子の数は徐々に増加する．また，シミュレーション開始直後の QM 領域数は 5 つであるが，酸素原子とシリコン原子の相互作用の結果によっては，1 つの QM 領域が 2 つに分割されたり，複数の QM 領域が 1 つに統合されたりする．

(2) の実験では，原子レベルの摩擦シミュレーショ

ンを行った．本シミュレーションは，ナノスケールのシリコン探針をシリコン基盤に接触させつつ移動させた場合の接触領域の酸化過程を調べるものである．接触領域には水分子や OH 分子を配置し，その周囲の領域を動的に QM 領域として定義し計算を実行する．シリコン探針が時間とともに基板上を移動するため，本シミュレーションでも QM 領域の大きさが動的に変化する．また，シリコン探針の形状の変化により QM 領域の数も動的に変化する．

5.2 実験結果

5.2.1 SIMOX シミュレーション実験

SIMOX シミュレーションの実行においては，国内および TeraGrid [4] に設置された計 8 台のクラスタを事前に予約した（**表 1** 参照）．これらのクラスタを 128 プロセッサごとに分割し，計 20 台の仮想的なクラスタと見なし，産業技術総合研究所の P32 クラスタのみを利用する予備的計算フェーズ 9 日間を含め 19 日間にわたってシミュレーションを行った．

本計算に先がけて利用計算機のスケジュールを行った．スケジュールの際には，動的な QM 領域数の増加や障害発生にともなう利用計算機数の変化に備えて，5 台の（仮想）クラスタに加え 1 台の予備クラスタをスケジュールし，領域数の増加に備えることにした．スケジュールは大きく 5 つのフェーズに分類される．

フェーズ 0（9 日間）：AIST Super Cluster（P32, M64 および F32 クラスタ，以下 ASC）のうち，P32 のみを用いる（予備的計算フェーズ）．

表 1 SIMOX シミュレーション実験における利用計算機．サイト名は，AIST：産業技術総合研究所，NCSA：National Center for Supercomputing Applications，PSC：Pittsburgh Supercomputing Center，USC：University of Southern California，U-Tokyo：東京大学，TITECH：東京工業大学，を表す

Table 1 List of clusters used in the SIMOX simulation experiment. Abbreviations in the site name are; AIST: National Institute of Advanced Industrial Science and Technology, NCSA: National Center for Supercomputing Applications, PSC: Pittsburgh Supercomputing Center, USC: University of Southern California, U-Tokyo: The University of Tokyo, TITECH: Tokyo Institute of Technology.

計算機名	サイト名	利用 CPU 数
P32	AIST	128 × 6
M64	AIST	128 × 2
F32	AIST	128 × 2
NCSA	NCSA	128 × 1
TCS	PSC	128 × 2
USC	USC	128 × 4
ISTBS	U-Tokyo	128 × 1
Presto	TITECH	128 × 2

フェーズ 1（2 日間）：ASC の 3 台のクラスタを用いる．

フェーズ 2（3 日間）：ASC および NCSA と PSC に各々設置された TeraGrid クラスタ 2 台を用いる．

フェーズ 3（3 日間）：ASC および USC のクラスタを用いる．

フェーズ 4（2 日間）：ASC，U-Tokyo および TITECH のクラスタを用いる．

実験では，19 日間で計 270 ステップの計算が行われた．また，10 日間の本計算において最長 5 日間にわたって継続実行することができた．実験結果に関して柔軟性，頑健性，効率性の観点からまとめる．

(1) 柔軟性

図 7 に QM 原子数の変化とそれにともなう利用 CPU 総数の変化を示す．実験開始時に QM 領域数は 5，QM 原子の総数は 62 個，利用 CPU の総数は 10 であった．シミュレーションが進むにつれて徐々に QM 原子の総数は増加し，最終的には 341 個になっている．それにともない利用 CPU の総数も増加し，実験フェーズ 0 の途中から 100CPU を超え，最終的に最大 702 まで増加している．

図中に見られる一時的な利用 CPU 数の増加は，QM 領域数が一時的に 5 から 6 に増加したことを表している．このような利用 CPU 数の増加にともない，実験期間中に計 244 回の動的割付けが発生している（**表 2** 参

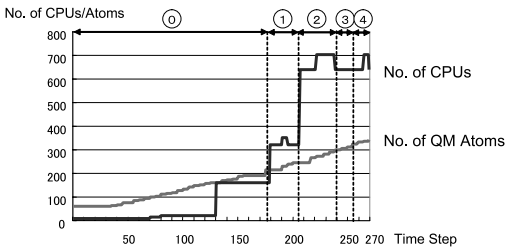

図 7 QM 原子の総数と利用 CPU 総数の変化．○付き数字は実験フェーズを表す

Fig. 7 Time evolution of the number of QM atoms and the number of CPUs used in the experiment. The numbers enclosed with a circle denote the experiment phases.

表 2 SIMOX シミュレーションにおける動的割付け結果

Table 2 Results of the dynamic allocations in the SIMOX simulation.

原因	発生回数	同一計算機割付け 成功	同一計算機割付け 失敗	別計算機割付け 成功	別計算機割付け 失敗
起動時障害	712	34	706	12	18
実行時障害	58				
領域変更	244	13	7	195	29

照）．これらの結果は，GridRPC と MPI を組み合わせるプログラミング手法に基づき実装したアプリケーションが動的な計算量の増加に対応し，利用 CPU 数を自動的に調節できることを示している．

(2) 頑健性

実験期間中，770 回の障害発生が観測された．これらのうち，遠隔実行プログラムの起動に失敗したケースが 712 回と大半を占める．失敗の主な原因は，Globus の job manager が起動されていなかった，MPI プロセス間の通信用メモリ資源の確保ができなかった，バッチキューが active になっていなかった，ディスク容量制限を超えてしまい起動に必要なファイルの生成ができなかった等である．起動に成功すると，比較的安定してシミュレーションが実行された．

障害が発生した場合，同一計算機への再割付けを試みても成功する確率は低い．表 2 から分かるように，障害発生時に同一計算機への割付けを試みて成功する確率は 4.6%（成功 34 回，失敗 706 回）程度であり，同一計算機に対して規定回数の失敗を繰り返した後，最終的には予備クラスタに割り付けられた．今回の実験では，規定回数失敗するまで同一計算機に対して再起動を試みる手法を採用したが，規定回数を多く設定しても有効ではないことが分かった．

実験中発生した障害において，Ninf-G の提供するタイムアウト機構により 110 回の障害が検知された．また，予約期間超過に関するタイムアウトに関しても，期間の超過を検出しその時点で予約されている別の計算機へと実行対象が変更されることが確認された．

タイムアウトに関する問題点としては，1 回の QM 計算に関する計算時間のタイムアウト（session timeout）の有効性があげられる．典型的な QM 計算は，1 回の計算に数時間を要し，かつ収束計算であるため計算時間が大きく変動する．このような特質を持つ計算に対してタイムアウトを設定する場合，タイムアウトの値として非常に大きな値を設定せざるをえない．しかしながらタイムアウトにより検知される障害は，計算の初期の段階で発生する可能性もある．実際，今回検知されたセッションタイムアウトは，計算初期にディスク容量制限を超えてしまい，計算が止まってしまったことが原因であった．また，この事例では Ninf-G の提供する heart beat 機能も有効に機能しなかった．通常，ディスク容量制限を超えるとアプリケーションが書き込みに失敗し異常終了することでエラーを検知できるが，今回の場合は何らかのタイミングでアプリケーションがブロックしたままとなってしまいエラーが検知されなかった．また，heart beat 機能はアプリケーションとは異なるスレッドで実行されており，そのために heart beat 送信が中断しなかったものと考えられる．詳細は現在究明中である．

(3) 効率性

今回の実験では最大 702CPU を用いたが，計算時間の大半は QM シミュレーションの実行で占められ，MD，QM シミュレーション間の通信時間や QM シミュレーションの起動コストは数%にすぎなかった．このように少ないコストで多数の計算機資源を管理できた要因は 2 つ存在する．1 つは，QM シミュレーションにおいて乱数に基づく SCF 計算の初期値推定手法を採用することにより，MD，QM 両シミュレーション間のデータ転送量を低減したというアプリケーションレベルの工夫である．もう 1 つは，多数の計算資源を GridRPC のクライアントが直接管理するのではなく，それらに対し MPI プログラムを割り付けることにより MPI プロセスグループという単位で管理するという我々のプログラミング手法の特性によるものである．これにより管理対象はたかだか数個程度に抑えられる．今回の実験では，利用可能な計算機資源の制約により数百プロセッサの利用にとどまったが，このことは，数千プロセッサを要する場合でも我々の手法により効率的な管理が可能であることを示唆している．

5.2.2 摩擦シミュレーション実験

摩擦シミュレーション実験では，国内および PRAGMA[5] テストベッドに設置された計 12 台の計算機を対象として，52 日間にわたってシミュレーションを実行させた．

本シミュレーションでは 52 日間で計 386 ステップの計算が行われ，最長 22 日間の継続実行を実現することができた．実験結果詳細を**表 3** に示す．実験結果に関しては，(1) の実験と異なる実行シナリオおよび計算機資源選択手法を採用したことから，主にそれらの影響についてまとめる．

実験期間中には 192 回の動的割付けが発生した．そのうち 165 回は障害の発生が契機となって行われたものであり，SIMOX シミュレーション同様，同一計算機への再割付けの失敗回数が非常に多い（104 回）．過

表 3 Friction シミュレーションにおける動的割付け結果
Table 3 Results of the dynamic allocations in the Friction simulation.

原因	発生回数	同一計算機割付け 成功	同一計算機割付け 失敗	別計算機割付け 成功	別計算機割付け 失敗
起動時障害	126	3	104	27	31
実行時障害	39				
領域変更	27	2	0	21	4

去の実行履歴から障害発生件数の少ない計算機を選択するという計算機選択戦略に反する結果となったのは，上記 104 回の失敗のうち 100 回が最も多数の CPU（128CPU）を要する QM 計算の割付けに失敗して連続して発生したためである．障害発生時，128CPU を有する計算機は 1 台しか利用可能でなかったため，他の計算機への回避が不可能であった．このように他の計算機を選択する余地がない場合を除き，本実験では 52 日間に 2 回の手動による起動を行うだけで実行することができた．

6. 考　　察

今回の実験を通じて得られた知見および考察をまとめる．

(1) より細かな非均質性への対応の必要性

グリッドミドルウェアはハードウェアやオペレーティングシステム等の非均質性を考慮して設計されているが，今回の実験において，より詳細なシステム設定，環境における非均質性への対応が必要であることが明らかになった．

- 外部との通信路の確立手段について

3.3 節で述べたように，Ninf-G は制約の多い複数サイトの計算機接続に関して考慮した設計となっている．しかし，PSC のクラスタにおいては，外部に対してネットワーク接続を行う場合，アプリケーションゲートウェイ（AGW）と呼ばれるノードを介した通信を行う必要があり，Ninf-G で PSC のクラスタにサーバプロセスを起動する際には Globus の GRAM 呼び出しを行うコマンド列に `(agw_count=#)` という Resource Specification Language（RSL）属性を渡してやる必要があった（`#` は確保したい Gigabit Ethernet インタフェースの数）．Ninf-G は Globus の RSL 仕様で規定されている属性を渡す機能は提供しているが，この `agw_count` という属性は PSC が独自に規定・利用している属性であるため，Ninf-G がこの属性を渡すことはできなかった．今回の実験においては外部との通信を行うノードは rank0 のプロセスが起動されるノードだけであったため，AGW を介さずに外部との直接の通信が可能な特殊なノードが rank0 に割り当てられるような特殊な設定をしてもらうことにより対応した．この経験に基づき，最新の Ninf-G には任意の RSL 属性を扱う機能が実装されている．

また，USC においては Firewall の設定が非常に厳格であり，外部に対する接続も許可されなかったため，今回の実験では，すべての通信をこちらのクライアントホストに中継する特殊な Proxy ノードを提供いただいた．この場合，起動するサーバプロセスに対して実際のクライアントホストではなくこの Proxy ノードを接続先のクライアントホストとして通知する必要がある．Ninf-G はサーバに対して接続先のクライアントホスト名を明示的に指定する機能を提供しているが，複数のサーバを利用する場合に各サーバごとに異なるクライアントホストを指定することができなかったため，この機能を利用することができなかった（他のサーバに対してもこの Proxy ノードがクライアントホストとして見えてしまうため）．今回の実験においては，計算ノードの hosts ファイルに実際のクライアントホスト名と Proxy ノードのアドレスを対応づけるエントリを追加することによりこの問題を回避した．最新の Ninf-G にはサーバごとにクライアントホスト名の指定を可能とする機能が実装されているが，このような厳格な Firewall で守られた計算機の利用については，Ninf-G 自体に通信を中継する機能を実装するといった本格的な対応が必要である．

- その他細かな非均質性について

バッチシステムの最大ジョブ実行時間，大きなファイルを生成してよいディスクパーティション，ディスククオータ，等の細かな項目についてサイトごとに相違があり，各クラスタごとにファイルの生成先やジョブを起動する際にバッチシステムに渡すオプション等を変更・指定する必要があった．エンドユーザがこれらの項目をいちいち指定して利用することは現実的ではなく，実行環境を仮想化することによってこれらの相違点を吸収して RPC を行うといった新しい技術の開発が必要である．

(2) グリッドオペレーションの自動化の必要性

複数のサイトにまたがる実験を行う際，TeraGrid は Web インタフェースを通じてクラスタノードの予約を申請し，申請が承認された場合に各サイトの管理者がバッチシステムの設定を変更して占有利用に対応している．しかし，今回の実験に際しては予約しているはずなのに実際には予約時刻がきてもキューがアクティブにならないという現象が NCSA のサイトで数回，PSC のサイトでも 1 回確認された．いずれも管理者の作業ミスによるものであったが，設定を手作業で行う以上ミスが起こる可能性は残る．今後複数サイトでのアプリケーションの実行をより使いやすく確実なものとするためには，予約システムの自動化，つまり複数サイトにまたがる計算機の事前予約を実現するスケジューラの実現が急務である．また，TeraGrid は我々ユーザからの要求受付け窓口を設置しており，ここにメールを送ると内容に応じて適宜各サイトの管理

者に伝達される仕組みになっている．また，NCSA のように 24 時間サポートをしているサイトにおいても，たとえば 8 時間勤務を 3 交代で行うといった運営をしているため，要求を受け取った担当者と実際に作業を行う担当者が異なる場合があり，管理者間で情報をきちんと伝える必要がある．しかし今回はこの情報の伝達に不備があり，我々が何回かメールで確認したにもかかわらず対応されなかったという問題が何回かあった．グリッド基盤を円滑に管理・運営するためには，このような情報の伝達が確実に行われるような仕組みを確立する必要がある．

(3) スケジューリング自動化の必要性

今回のスケジューラ実装においては，計算機利用に関する制約の変更を利用者がファイルによって設定し，スケジューラに伝えることでスケジューリングを行う方式を採用した．しかしながら，長期にわたり実行対象計算機を変更しながらシミュレーションを継続していく大規模長時間実行アダプティブ Grid アプリケーションの実行において，利用者が随時計算機利用に関する制約をファイルに記述することは利用者にとって負担が大きい．特に今回の実験のように広域にわたって分散する計算機を利用する場合は，時差の問題で昼夜を分かたぬ対応が必要となった．このような利用者の負担を軽減するためには，スケジューリングの自動化が望まれる．現在，Grid 上に分散配置された計算機を対象としたスケジューリングシステムを Grid ミドルウェアとして提供するための研究が行われている[11),12)]．このようなスケジューリングシステムや Ninf-G のような Grid ミドルウェアを統合することによってスケジューリングの自動化を図ることは，今後の課題である．

7. ま と め

動的に計算規模が変化するという特性を持つ大規模長時間実行アプリケーションを GridRPC と MPI を組み合わせるプログラミング手法に基づき Grid 化し，実行するための方法論について論じた．実装に際しては，柔軟性，頑健性，効率性という 3 種類の要求を満足する必要があることを示し，それらを満足するために実装すべき機能に関して考察した．また，典型的な大規模長時間実行アダプティブシミュレーションの例である Adaptive Hybrid QM/MD シミュレーションコードを Grid 化し，2 種類の異なる計算機選択手法に基づき環太平洋 Grid 環境上で長時間継続計算実験を行うことで，我々のアプローチの有効性を検証した．その結果，動的な計算規模の変化に対応し，実行対象計算機を変更しながら利用 CPU 数を自動的に調節できることを確認した．実験中には多数の障害が発生したが，それらを自動的に検知，復旧してシミュレーションを継続できた．さらに，数百プロセッサ規模の計算機資源を効率的に管理できることも確認した．また，摩擦シミュレーション実験では最長 22 日間の継続実行を実現し，過去の履歴に基づき安定した計算実績を持つ計算機を優先して利用する計算機選択戦略が有効であることが分かった．

しかしながら，実用レベルで大規模長時間実行 Grid シミュレーションを実施するには解決すべき問題が数多く残っている．たとえば，単に再起動を行うという単純な戦略ではいったん発生した障害の復旧は困難であり，現段階ではユーザの介入による問題点の解決が必要である．より洗練された障害復旧の機構を考案することも重要であるが，より現実的な Grid 環境の運用管理形態を検討していくことがさらに重要である．

5 章で述べたように，今回の実験において発生した障害の原因は，運用管理技術および組織間の情報伝達技術の未熟さに起因するものが多い．また Grid 環境に特有な問題として，計算機が広域に分散しているために，あるサイトの計算機に障害が発生した場合，原因が分かっているにもかかわらず時差の関係でサイト管理者と連絡がとれず，解決が遅れるということも発生した．このような場合，他の計算機に多重に計算を割り付けることで実行を継続するということが有効であることは分かったが，根本的な解決法ではない．広域な Grid 環境をどのように運用管理すればよいかを検討し，明確化することが急務である．そのために，今回のような実験を積み重ね，発生する障害に関する情報を蓄積していくことが重要である．

謝辞 本研究に際し，Hybrid QM/MD コードを提供いただいた名古屋工業大学尾形修司助教授に感謝いたします．また，本研究の遂行にあたり貴重なご助言をいただいた南カリフォルニア大学中野愛一郎助教授に感謝いたします．

また，SC2005 における実験に協力し計算資源を提供いただいた TeraGrid Exective Commitee メンバ諸氏，Pittsburgh Supercomputing Center，National Center for Supercomputing Applications，University of Southern California，東京大学田浦研究室，東京工業大学松岡研究室に感謝いたします．

参 考 文 献

1) Ogata, S., Shimojo, F., Kalia, R.K., Nakano, A. and Vashishta, P.: Hybrid Quantum

Mechanical/Molecular Dynamics Simulations on Parallel Computers: Density Functional Theory on Real-space Multigrids, *Computer Physics Communications*, Vol.149, pp.30–38 (2002).
2) 武宮 博, 田中良夫, 中田秀基, 関口智嗣：Ninf-G2：大規模 Grid 環境での利用に即した高機能, 高性能 GridRPC システムの実装と評価, 情報処理学会論文誌：コンピューティングシステム, Vol.45, No.SIG11(ACS 7), pp.144–159 (2004).
3) 武宮 博, 田中良夫, 中田秀基, 関口智嗣：MPI と GridRPC を利用した大規模 Grid アプリケーションの開発と実行：Hybrid QM/MD シミュレーション, *SACSIS 2005*, IPSJ Symposium Series Vol.2005, No.5, pp.153–160 (2005).
4) http://www.teragrid.org
5) http://www.pragma-grid.net
6) Casanova, H., Bartol, T., Stiles, J. and Berman, F.: Distributing Mcell Simulations on the Grid, *Journal of Supercomputing Applications*, Vol.15, pp.243–257 (2001).
7) Takemiya, H., Shudo, K., Tanaka, Y. and Sekiguchi, S.: Constructing Grid Applications Using Standard Grid Middleware, *Grid Computing*, Vol.1, pp.117–131 (2003).
8) Kikuchi, H., Kalia, R.K., Nakano, A., Vashishta, P., Iyetomi, H., Ogata, S., Kouno, T., Shimojo, F., Tsuruta, K. and Saini, S.: Collaborative simulation Grid: multiscale quantum-mechanical/classical atomistic simulations on distributed PC clusters in the US and Japan, *Proc. SC2002*, IEEE, Los Alamitos (2002).
9) Kimura, T. and Takemiya, H.: Distributed Parallel Computing for Fluid-Structure Coupled Simulations on a Heterogeneous Parallel Computer Cluster, *Journal of Supercomputing Applications*, Vol.13, No.4, pp.320–333 (1999).
10) Karonis, N., Toonen, B. and Foster, I.: MPICH-G2: A Grid-Enabled Implementation of the Message Passing Interface, *Journal of Parallel and Distributed Computing*, Vol.63, No.5, pp.551–563 (2003).
11) http://hpcsoftware.teragrid.org/Software/user/show_asset.php?asset_id=189&view=TeraGrid
12) http://www.g-lambda.net

(平成 18 年 5 月 8 日受付)
(平成 18 年 9 月 8 日採録)

武宮 博（正会員）

日立東日本ソリューションズ（株）ネットワークソリューション本部副技師長．昭和 61 年東北大学大学院理学研究科天文学博士前期課程修了．平成元年同博士後期課程中退．同年日立東日本ソリューションズ（株）入社．平成 14 年より産業技術総合研究所グリッド研究センターへ派遣．

田中 良夫（正会員）

昭和 40 年生．平成 7 年慶應義塾大学大学院理工学研究科後期博士課程単位取得退学．平成 8 年技術研究組合新情報処理開発機構入所．平成 12 年通産省電子技術総合研究所入所．平成 13 年 4 月より独立行政法人産業技術総合研究所．現在同所グリッド研究センター基盤ソフトチーム長．博士（工学）．グリッドにおけるプログラミングミドルウェア，計算ポータル，およびテストベッド構築に関する研究に従事．IC'99 論文賞．ACM 会員．

中田 秀基（正会員）

昭和 42 年生．平成 2 年東京大学工学部精密機械工学科卒業．平成 7 年同大学大学院工学系研究科情報工学専攻博士課程修了．博士（工学）．同年電子技術総合研究所研究官．平成 13 年独立行政法人産業技術総合研究所に改組．現在同所グリッド研究センター主任研究官．平成 13 年より東京工業大学客員助教授を兼務．グローバルコンピューティング，並列実行環境に関する研究に従事．

関口 智嗣（正会員）

昭和 34 年生．昭和 57 年東京大学理学部情報科学科卒業．昭和 59 年筑波大学大学院理工学研究科修了．同年電子技術総合研究所入所．情報処理アーキテクチャ部主任研究官．以来，データ駆動型スーパーコンピュータ SIGMA-1 の開発等の研究に従事．平成 13 年独立行政法人産業技術総合研究所に改組．平成 14 年 1 月より同所グリッド研究センターセンター長．並列数値アルゴリズム，計算機性能評価技術，グリッドコンピューティングに興味を持つ．市村賞受賞．日本応用数理学会，ソフトウェア科学会，SIAM，IEEE 各会員．