GridFMO ——グリッド環境を用いた大規模量子化学計算

池上 努[†] 真木 淳[††] 高見利也[††]
田中良夫[†] 横川三津夫[†††]
関口智嗣[†] 青柳睦[††]

グリッド技術を応用して量子化学計算のアルゴリズムの 1 つであるフラグメント分子軌道（FMO）法を実装し，タンパク質の大規模電子状態計算を広域分散環境で実行するプログラム GridFMO を開発した．長期間安定な計算を支えるため，Ninf-G をベースとして高い耐故障性と柔軟な計算資源管理を実現するグリッドミドルウェアを作成した．実証実験として環太平洋地域の 10 台のクラスタ計算機を用い，各計算機上のバッチキューシステムを介して他ユーザと資源を共有しながら 70 日間にわたり GridFMO 計算を 14 回実施した．実験を通じ，耐障害性や資源管理機構の必要性を示した．

GridFMO —— Large Scale Quantum Chemistry on the Grid

Tsutomu Ikegami,[†] Jun Maki,[††] Toshiya Takami,[††]
Yoshio Tanaka,[†] Mitsuo Yokokawa,[†††] Satoshi Sekiguchi[†]
and Mutsumi Aoyagi[††]

The Grid technology was used to develop the GridFMO application to perform quantum chemical calculations in the distributed parallel environment. The Fragment Molecular Orbital (FMO) method was employed to obtain accurate electronic states of proteins. To support long-term calculations, a Grid middleware was developed with high fault-tolerancy and flexyble resource management, based on the lower middleware of Ninf-G. Uniting 10 cluster computers over the Pacific rim, 14 GridFMO calculations were conducted in a period of 70 days, while sharing them with other users via batch queue systems on each machine. The importance of the fault-tolerancy and the resource management was demonstrated through the experiment.

1. はじめに

科学技術計算の応用分野は，近年の計算機性能の向上にともなって著しく進展した．特に計算化学の分野では，第一原理計算の精度が飛躍的に高まり，実験と比肩しうるレベルで化学反応機構を調べることが可能となっている．また数千〜数万原子からなる大規模分子の電子状態計算も視野に入ってきた．こうした発展には，時々の計算機技術に合わせて考案されてきた多様な計算手法の寄与が大きい．第一原理計算の歴史を振り返ると，その黎明期には CPU 資源はまだ貴重であり，計算の中間結果は磁気ディスクやあるいは磁気テープにさえ保存し，再利用に努めた．CPU の速度が向上すると，こうした中間結果を必要に応じて再計算する直接法に置き換えられ，計算時間が大きく短縮された．ベクトル計算機や並列計算機など新たなアーキテクチャが現れると，これに適した新しいアルゴリズムが考案され，計算の精度と規模を押し広げるとともに第一原理計算の信頼性を確立していった．

今や第一原理計算は学術研究だけでなく，産業においても研究開発の重要な手段となっている．しかし従来の計算手法は，その計算量が原子数の 3 乗から 4 乗に比例して増加するため，現在の計算機性能でも巨大分子の計算は困難を極める．こうした計算を現実的な時間内で処理するため，1999 年にフラグメント分子軌道法（FMO 法）[1] が開発された．FMO 法は交換相互作用の局在性に基づいて全体の計算を小さな断片の計算に分割する手法で，全計算量の大幅な軽量化が可能である．FMO 法の登場により巨大分子の電子状態

† 産業技術総合研究所グリッド研究センター
Grid Technology Research Center, National Institute of Advanced Industrial Science and Technology (AIST)
†† 九州大学情報基盤センター
Computing and Communications Center, Kyushu University
††† 理化学研究所次世代スーパーコンピュータ開発実施本部
RIKEN Next-Generation Supercomputer R&D Center

計算は実用段階に入ったが，しかしなおその計算には大量の計算資源を必要とする[2]．

本論文では，グリッド技術を用いた FMO 法の実装である GridFMO を紹介し，大型計算機を利用しなくても比較的小規模なクラスタ計算機を互いに共有するグリッド環境があれば，大規模計算が可能であることを示す．広域分散環境では計算資源の質は一様ではなく，また第三者との資源の競合や各種の障害により，利用可能な計算資源は時間とともに変動して一定ではない．計算を高効率かつ長期間安定に維持するには，負荷分散の工夫や高い耐障害性，柔軟な資源管理の仕組みが不可欠となる．本論文では GridFMO 計算で必要とされる諸要件を整理し，それらを満足するミドルウェアの設計と実装について述べる．開発した手法の有効性を実証するため，計算対象として 16,664 原子からなる分子モータ系を選び，これを 1,090 もしくは 545 個の断片に分割して GridFMO 計算を実施した．計算は環太平洋地域に散らばる 10 台のクラスタ計算機を用い，特に占有環境を設けずに他ユーザと計算機資源を共有しながら，約 3 カ月にわたって実施した．長期間の実証実験を通じて得られた知見について，あわせて論じる．

次章で FMO 法の概要を紹介し，計算アルゴリズムと広域分散環境双方の問題点について述べるとともに，これを克服するためのグリッド技術の応用について論じる．3 章では GridFMO 計算の実施例を紹介し，4 章で得られた知見について述べた後，終章にまとめる．

2. 計 算 手 法

2.1 FMO 法

FMO 法では，性質の明かな化学結合を切断し，対象分子を M 個の断片に分割する．各断片の電子状態を，古典的な静電環境の形で他の断片の影響をとりこみながら計算する．ここで使用する静電環境は計算した電子状態から再構築されるので，一連の計算は電子状態と静電環境の整合性がとれるまで繰り返す[1]．この反復を SCC（self-consistent charge）サイクルと呼ぶ．SCC サイクルが収束し静電環境が定まると，断片の対について網羅的に電子状態計算を実施する．化学結合の切断にともなう誤差のほとんどは，この二量体の計算を通じて補正される．単量体および二量体のエネルギーから，全エネルギー E_{tot} は以下のように近似される．

$$E_{\mathrm{tot}} = \sum_{i=1}^{M} E_i + \sum_{i=2}^{M} \sum_{j=1}^{i-1} (E_{ij} - E_i - E_j) \quad (1)$$

```
1  Initial guess generation
2  synchronize workers
3  do
4    Fragment calculation
5    synchronize workers
6    update environment
7  until SCC is convergent
8  Fragment pair calculation
9  synchronize workers
10 total energy calculation
```

図 1 FMO 計算の処理の流れ
Fig. 1 Workflow of the FMO calculation.

ここで E_i は i 番目の単量体のエネルギー，E_{ij} は i-j 二量体のエネルギーである．

FMO 法では，各 SCC サイクルごとに M 本の電子状態計算が必要になる．これらは共通の静電環境の下，独立並行に計算可能である．二量体の計算も同様に独立に計算できるが，計算すべき E_{ij} の総数は M^2 に比例して増えるため，M が大きくなると破綻する．そこで空間的に十分離れた二量体については量子効果を無視し，E_{ij} を古典的な静電相互作用で近似した $\tilde{E}_{ij}$ で置き換えて，計算量を削減する[3]．

以下，FMO 計算の並列化について考える．E_i などを決定する個々の電子状態計算は並列実行が可能であるが，その実装は細粒度の並列環境が前提となっており，またその効率は並列度の上昇とともに急激に悪化する．したがって FMO 計算を分散並列環境で実行する場合，計算機資源を適切にグループ分けして各電子状態計算を割り付けるマスタ・ワーカ型の実装が適切である．処理の流れを**図 1** に示す．ワーカ数は M よりも十分に小さいと仮定し，図中，ステップ 1，4，および 8 の計算をマスタ・ワーカモデルで実行する．その際，負荷分散と環境データの転送の 2 点が問題となる．

負 荷 分 散

GridFMO の計算時間のほとんどは，図 1 中ステップ 4 と 8 に費される．ステップ 4 では E_i の計算を独立のジョブとしてワーカに割り付け，これを SCC サイクル内で 30 ～ 60 回ほど繰り返し，収束させる．各サイクルの終端ではワーカ間の同期が必要になるため，すべてのワーカがほぼ同時に処理を終えることが望ましい．しかし化学的な制約から分子の均等な断片化は困難であり，また E_i の計算時間は断片のサイズに敏感に依存する．今回採用した手法を例にとると，電子状態計算の計算量は断片に含まれる電子数の 3 ～ 4 乗

に比例して増大する．さらに，断片のサイズと利用可能なメモリ量に応じて計算アルゴリズムを省メモリ・高計算量型のものに切り替えるため，計算時間のサイズ依存性は計算量よりも顕著となる．この結果，比較的均等に分割可能な系であっても，E_i の計算時間は 10 ∼ 20 倍の範囲でばらついてしまう．この条件下で負荷分散を図るため，(1) ワーカへの断片の割付けを動的に行い，かつ (2) 大きな断片から順番に処理されるよう，スケジューリングを行う（Large Fragments First（LFF）scheduling）[4]．LFF スケジューリングは，大断片の計算で生じた負荷の不均衡を小断片の計算で埋めることでワーカ間の処理時間の均衡を図る．すなわち，各ワーカにまず大断片の計算を割り付け，先に計算を終えたワーカから順次，小断片の計算を割り付けていくことで，同期時間をおよそ小断片の計算時間内に抑える．小断片の計算時間は通常，1 SCC サイクルに要する時間の数%に満たないので，高い並列化効率を実現可能である．

ステップ 8 では E_{ij} と $\tilde{E}_{ij}$ を計算する．$\tilde{E}_{ij}$ の個数は M^2 に比例して膨大であるが，個々の計算量はきわめて小さい．このため，$\tilde{E}_{ij}$ は複数個，束ねて 1 単位のジョブとし，マスタ–ワーカ間の通信を低減する．ジョブ間の計算量にはなお 100 倍以上の差が残るが，ジョブの総数が多くかつ途中で同期の必要がないため，LFF スケジューリングで十分，負荷の均衡をとることができる．

環境データの転送

ステップ 4 と 8 の計算では各断片の及ぼす静電環境が必要となる．静電環境はステップごとに更新されるが，同一ステップ内のジョブ間では共通である．環境データはジョブごとの入出力データと比較してサイズが大きく，ジョブ実行のつど，ワーカへ転送するのでは効率が悪い．そこでステップの開始時に各ワーカに一括してデータ転送を行い，以降これを再利用しながらジョブの計算を進める．GridFMO 計算ではワーカにステートを持たせることで，データの再利用を実現している．

2.2 広域分散環境への対応

GridFMO 計算では，前節で述べたマスタ・ワーカモデルをグリッド技術を用いて広域分散環境上に実現する．今回の実験でワーカとして用いた計算機を表 2 にまとめた．このように不均質な計算環境で FMO 計算を実施する場合，計算性能のばらつき，通信性能のばらつき，および計算資源の変動の 3 点が問題となる．

計算性能のばらつき

計算性能のばらつきは負荷分散に悪影響を及ぼす．FMO 計算における LFF スケジューリングは，ワーカ間の同期時間を小断片の計算時間程度に収める[4]．計算性能にばらつきがあると，この同期時間は最も性能の劣る計算機で縛られてしまう．小断片の数が多く，かつ個々の計算時間が十分に小さければ，2 倍程度の性能差は同期時間の揺らぎに埋もれてしまい，さほど目立たない．しかし高精度計算のために分子の断片化を粗くとると，断片数 M が減少する一方で個々の計算時間は延びるため，性能のばらつきが負荷分散に悪影響を及ぼし，同期に時間がかかるようになる．

悪影響を除くには，ワーカの計算性能を均一に揃えるのが近道である．GridFMO では階層的な並列化を用いてこれを実現する．すなわち各ジョブを細粒度で並列処理し，低性能のノードは複数束ねることで計算性能を上げ，ワーカ性能の均一化を図る．

階層的並列化はさらに，LFF スケジューリングの破綻を防ぐためにも用いられる．系の化学的な性質から断片の 1 つを極端に大きくとらざるをえないケースでは，大断片の計算が律速となって LFF の仕組みが破綻する[2]．この場合，ワーカの 1 つに多数のノードを割り当てて高性能化し，大断片の計算を高性能ワーカに割り付けて計算時間の突出を抑制することができる．

通信性能のばらつき

ワーカを広域に分散して配置する場合，マスタ–ワーカ間の通信性能はワーカごとに異なり，一定ではない．通信量が小さく，通信時間が計算時間に比べて無視できる量であれば，通信性能の不均衡が負荷分散に影響を与えることはない．しかし GridFMO 計算で各ワーカに転送する環境データは，大きな系では数百 MB に達する．表 2 の B/W 値（マスタ–ワーカ間の通信速度）より，今回の実験環境ではマスタ–ワーカ間の通信速度に 200 倍近いばらつきがあり，環境データの転送に要する時間に数秒から数十分の不均衡を生じた．このため，環境データ転送後には同期をとらず，転送を完了したワーカから順番にジョブを割り付けていく必要がある．また同一計算機上に複数のワーカがある場合，これらに個別に環境データを転送するのは非効率なので，代表ワーカに対してデータ転送を行い，他のワーカに対しては代表ワーカから再転送するのが望ましい．

ジョブの入出力データはジョブが割り付けられるつど，マスタ–ワーカ間で送受信する．ジョブはワーカに動的に割り付けられ，またジョブごとに計算量が異なるので，同一計算機に複数のワーカが存在しても通信が競合する可能性は低い．GridFMO 計算では，入力データの共通部分は環境データとともに送信される

ため，ジョブごとに送信しなければならないデータ量は数百 B である．出力データは単量体の計算では断片のサイズに応じて 50 KB ～ 1.5 MB の間でばらつき，二量体の計算では 10 KB 以下に収まる．これら入出力データの通信時間は計算時間と比較してほぼ無視できる量であり，そのばらつきは負荷分散にはほとんど影響しない．

計算資源の変動

広域分散環境では，すべての計算機がつねに利用可能であることは期待できない．個々の計算機はそれぞれ独立に運用されており，メンテナンスの予定はまちまちである．また占有環境を構築せず，既存のローカルキューを介して計算機を他のユーザと共有しながら利用する場合，計算資源が利用可能な期間を前もって予測するのは困難である．このため，アプリケーションは各時点で利用可能な計算資源をつねに把握し，これをやりくりして計算を継続しなければならない．GridFMO で採用するマスタ・ワーカモデルでは，計算資源はもっぱらワーカが消費する．したがって計算資源が変動する状況下では，ワーカを静的に用意するのではなく，ジョブごとに資源を割り当ててワーカを動的に生成する，柔軟な資源管理が必要になる．

計算資源管理の仕組みは，耐障害性の実現にも利用することができる．大規模な系の GridFMO 計算には 1 回あたり数日から 1 週間を要する．計算には複数の計算機を利用するので，期間中マシントラブルや通信障害に見舞われる可能性が高くなる．こうした障害を適切に検出できれば，管理機構を経由して問題の計算資源の利用を停止し，失敗したジョブを別のワーカに割り付けることで，アプリケーションの実行を維持することができる．

2.3 グリッドミドルウェアの設計

ミドルウェアの設計は，アプリケーション側で実装する部分とミドルウェア側で担保すべき機能の分担が焦点となる．アプリケーションの開発ではロジックに集中できることが理想なので，前節で述べた広域分散環境における諸要件はミドルウェア部分で解決されることが望ましい．GridFMO の実装では，実際に利用する計算資源をアプリケーションから隠蔽することを基本方針として，広域分散処理のミドルウェア部分への隔離を図った．具体的には以前に実装したレプリカ交換モンテカルロ法の経験に基づき[5]，バッチキュー型のユーザインタフェースを基にこれを拡張した．一方，耐障害性についてはミドルウェア側は障害の検出にとどめ，失敗したジョブの再実行はアプリケーションの裁量にまかせた．これは「障害」としてジョブの論理的なエラーまで含む，より高度な耐障害性の実現には，アプリケーションに密着した処理が欠かせないとの判断に基づく．たとえば電子状態計算には非線型方程式の繰返し解法が含まれるため，必ずしも収束する保証はない．このようなケースでは，アプリケーション側でジョブの成否を判断し，失敗時には計算手法の変更など適切な迂回措置をとることで，堅固な耐障害性を確保できる．

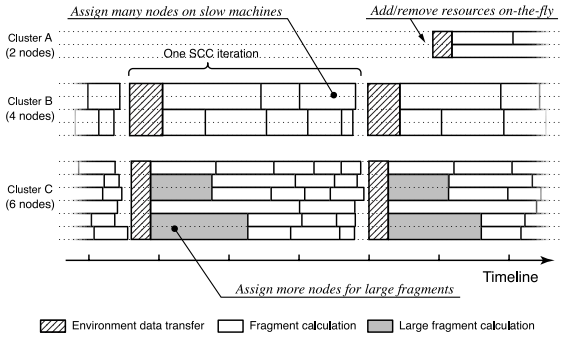


図 2 GridFMO 計算における模式的な処理の流れ
Fig. 2 A schematic timing diagram of the GridFMO calculation.

図 2 に GridFMO 計算における模式的な処理の流れを図示する．以下，アプリケーション側から見た機能をジョブの実行と環境データの転送についてそれぞれ論じ，次節でその具体的な実装について述べる．

ジョブの実行

バッチキュー型のインタフェースは，図 1 のステップ 1，4，および 8 で用いられる．各ステップの冒頭で実行すべきすべてのジョブを生成し，これを一息に待ち行列に登録する．このとき，LFF スケジューリングの破綻回避用に高性能なワーカを準備するケースでは，大断片のジョブは別に用意した待ち行列に登録する．アプリケーションは各ジョブの終了を待ち受け，終了したジョブのステータスを調べて，異常終了したものについては適宜，待ち行列に再登録する．

LFF スケジューリングが機能するためには，ジョブの実行順序が重要になる．今回はアプリケーション側でジョブを並べ替えてから待ち行列に登録し，ミドルウェアは基本的に登録順に実行することとした．加えてジョブには優先順位を付け，待ち行列への登録順序を飛び越えることも可能にした．この機能は特に，異常終了したジョブを優先して再実行するために用いた．

環境データの転送

ステップ 4 と 8 の冒頭では，その時点で有効なワーカに対し環境データを転送しなければならない．またステップ実行途中から利用可能になったワーカに対し

ては，ジョブを割り付ける前に環境データを転送する必要がある．バッチキュー型のインタフェースは，実際に使われる計算資源をアプリケーションから隠蔽するので，このような広報機構を実現できない．そこでミドルウェア側に特別な広報用待ち行列を用意してこの問題に対処した．広報用待ち行列にジョブが登録されると，これを全ワーカに最優先ジョブとして予約し，処理中のジョブの終了を待って順次実行する．広報ジョブはまたミドルウェア内にキャッシュされ，新たに加わったワーカや一時中断から復帰したワーカに対してただちに発効される．広報ジョブはその他の点では通常のジョブと同様に扱われるので，環境データの転送が完了してワーカが開放されると，他ワーカがまだ転送中であっても関係なく，通常のジョブが割り付けられていく．

2.4 グリッドミドルウェアの実装

内部構成

構築したミドルウェアはアプリケーション（Client，C）のほかに 3 つのオブジェクトから構成される．すなわち，Bookkeeper（BK），Doorkeeper（DK），および Machine（M）である．構成を**図 3** に示す．BK は計算資源の利用状況を管轄し，実際の計算資源は DK が管理する．M は個々の計算機に対応し，DK 内部に保持される．利用する計算資源のリストはユーザが用意し，BK，DK を立ち上げる際に指定する．マスタ・ワーカ型の構成では C がマスタに相当し，ワーカは BK–DK–M の連携により動的に生成される．

待ち行列にジョブが登録されると，C は BK に計算資源の割当てを要求し，BK は利用可能な資源を切り分けて C に提供する．これを受けて C は DK に利用すべき資源の指示とともにジョブの入力データを渡す．DK は指示された M に入力データを渡し，ジョブの実行を依頼する．ジョブが終了すると，出力データと終了ステータスは DK を経由して C に伝えられ，C は BK に計算資源を返却する．このとき，BK にジョブの終了ステータスがあわせて通知される．障害などによりジョブが異常終了すると，これは終了ステータスを介して C 経由で BK に伝えられる．伝えられた異常は BK 内に記録され，ジョブの失敗例が所定の回数に達すると（もしくは広報ジョブの失敗に対してはただちに）その計算資源の提供を中断する．

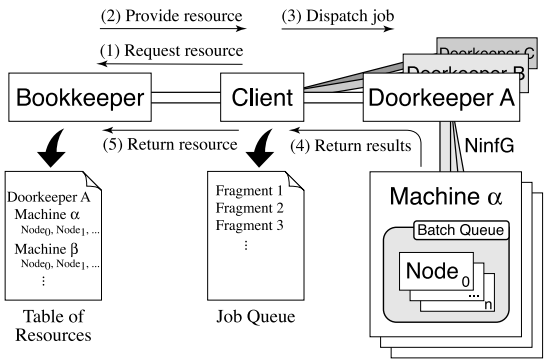


図 3 ミドルウェアの構成
Fig. 3 Structure of the middleware.

計算資源を共有するケースでは，M の使用する計算資源は一般に各計算機上にしつらえたバッチキューシステムを介して確保される．この場合，キュー内の滞留時間，すなわち M を起動してから実際に利用可能になるまでの待機時間は予測困難である．アプリケーションからこの待機時間を隠蔽するため，DK に M の監視機構を設け，M が利用可能になるのを待って BK に通知するようにした．この仕組みと障害検出機能が対となって，計算資源の動的な管理を支えている．

BK が資源を切り分ける際，同じ待ち行列内のジョブには計算性能がだいたい一定となるよう，M ごとに提供量を調節する．すなわち，ノード性能の低い M からはより多くのノードを割り当てることで性能差を目立たなくする．また大断片用待ち行列に登録されたジョブに対しては，ノード性能が高くかつ多数のノードを有する計算機を割り当てる．こうした資源の割当て量は，ベンチマーク試験の結果などを参考にユーザがあらかじめ BK に設定しておく．広報ジョブに対しては，M 内のすべてのノードをまとめて単一のワーカとして提供することができる．環境データの転送にはこの機能を用い，各 M に対して 1 組のデータを転送した後，M 内の各ノードに再配布することで転送効率を稼いでいる．

実装の詳細

BK–C–DK 間の連携は簡素な通信オブジェクトを介して行う．通信オブジェクトを切り替えることで，三者を同一プロセス内の独立したスレッドに配する構成や，独立なプロセスとして異なるマシン上に配する構成など，様々な運用形態をとることができる．

今回の実験では，DK–M 間の連携を Ninf-G 2.4.1[6] で実装したシステムを開発して用いた．Ninf-G は Globus Toolkit を用いて構築した GridRPC[7] の参照実装で，遠隔呼び出しをグリッド環境の下で安全かつ簡便に行うための基盤を提供する．本実装では DK 内に Ninf-G の実行環境を構築し，M は DK 内部に Ninf-G のハンドルアレイとして実現されている．M を起動するとハンドルアレイが作成され，GRAM 呼び出しを介して資源確保の要求がバッチキューに対し

て登録される．資源の要求量，すなわちハンドルアレイのサイズはユーザがあらかじめ指定する．起動後，M は要求がキューから実行に移され，すべてのハンドルが利用可能になるまで待機する．資源が確保されると M は利用可能なノードのリストを作成し，DK を介して BK に伝える．BK はこのリストを参考にしてジョブの実行を担当するハンドルを指定する．ジョブを細粒度に並列処理する場合，BK はジョブに複数のハンドルを割り当ててマシンリストを作成し，C, DK を介して M に渡す．M はハンドルの 1 本を用いてジョブを処理するが，そこで実行されるユーザプロセスは渡されたマシンリストを用いて排他的に並列処理することができる．

ジョブの実行中に障害が発生すると，これは Ninf-G API のエラーとして検出され，BK に伝達される．Ninf-G には強力な障害検出機能が備わっており[8]，耐障害性の確保に活用した．とりわけハートビートに代表される各種のタイムアウトを適切に設定することで，検出の困難なネットワーク障害が発生したケースも回収することができる．障害が検出されると，BK は M を単位として計算資源の利用を停止する．より細かく，ノード単位で利用を停止する手法も考えられるが[8]，これまでの経験では障害は計算機単位で発生することが多く，また実装コストが高いこともあって断念した．

使用停止となった M は，BK 経由で DK に対して再起動を指示することで，再び利用可能になる．Ninf-G を用いた実装では，M に対して改めて GRAM 呼び出しが発生し，新たな資源確保要求がバッチキューに対して登録される．しかしながらミドルウェア内部には自動再起動の仕組みは設けず，かわりに BK に対する対話的なインタフェースを用意した．すなわち障害発生時にはその原因を調査してこれを除いた後，手動で M を再起動する．自明な障害に対しては，外部に BK 監視プロセスを立てて上記の作業を自動化することもできる．今回の実験では特に，バッチキューの実行時間タイムアウトにともなう M の停止に対し，再起動を自動化した．この対話的なインタフェースはまた，新たな計算機資源を動的に追加したり，M 起動時の資源要求量を変更したりする際にも用いられ，柔軟な資源管理の一端を支えている．

3. 実験の概要

FMO 法と古典的な分子力場を比較して相互に検証するため，分子モータの一種 F_0-ATPase を対象にとり，大振幅運動に沿ってポテンシャルエネルギー曲線を計算した．曲線上の分子構造は分子力場法のプログラムパッケージ Peach[9] を用いて決定し，ロータの回転角ごとに構造最適化を行った．GridFMO 計算は曲線上の 7 点を選んで実施した．計算で必要となる断片の電子状態計算や SCC 収束条件の判定には，量子化学計算パッケージ GAMESS[10]（2004-5-19 版）をもとに，各ステップごとの機能を単離したプログラム群を開発して用いた．各電子状態計算の細粒度な並列処理には，GAMESS に実装されている並列化機構 DDI（Distributed Data Interface）[11] をそのまま利用した．DDI ライブラリのコンパイルには，すべての計算機で TCP ソケット型の設定を用いた．

全系は 16,664 原子からなり，61,784 個の電子が含まれる．これをアミノ酸残基ごとに $M = 1{,}090$ 個の断片に分割し，GridFMO 計算を実行した．典型的な計算例について，各ステップで処理するジョブ数を**表 1** にまとめた．表中，$N(E)$ はエネルギー計算のジョブ数，$T(E)$ はその計算時間を表す．E に付した添字の意味は 2.1 節を参照されたい．env と T(env) は環境データのサイズおよびその転送・展開に要した時間である．T(SCC) は SCC 1 サイクルに，T(sync) は SCC サイクル終端での同期に要した時間の平均値，T(tot) は全計算時間である．N_{wk} には計算で用いたワーカ数の最大値を示した．二量体の電子状態計算の数 $N(E_{ij})$ は構造の変化にともなって 10 前後，変化する．古典近似二量体は 579,958 組あるが，50 個ずつまとめて 1 つのジョブとした．

この標準精度の計算とは独立に，系を 2 残基ごと $M = 552$ 個の断片に分割したより高精度な計算も実施した．高精度計算ではジョブの総数が減る一方，個々のジョブの計算時間は長くなるので，負荷分散はより

表 1 F_0-ATPase の GridFMO 計算における統計量
Table 1 Parameters in a GridFMO calculation of F_0-ATPase.

Precision	Standard	High
$N(E_i)$	1,090	552
$N(E_{ij})$	13,547	5,315
$N(\tilde{E}_{ij})$	11,600	2,936
env/MB	205	338
T(env)/min	1.0 – 15.7	1.0 – 38.4
$T(E_i)$/min	0.6 – 12.2	1.4 – 32.3
T(SCC)/min	39.1	103.9
T(sync)/min	2.6	8.1
$T(E_{ij})$/min	1.9 – 62.0	15.8 – 206.8
$T(\tilde{E}_{ij})$/min	0.3 – 42.0	2.2 – 118.4
T(tot)/hrs	64.6	137.6
N_{wk}	119	86

表 2 実験に使用した計算機
Table 2 Machines used in the experiments.

Name	Location	Type of node	N_{node}	$N_{\mathrm{wk}}^{\mathrm{std}}$	$N_{\mathrm{wk}}^{\mathrm{high}}$	B/W	BM
SDSC	San Diego	Dual Xeon (2.4 GHz)/2 GB	10	10	5	0.53 MB/s	1.9
NCSA	Illinois	Dual Xeon (2.0 GHz)/2 GB	8	8	4	0.29 MB/s	2.0
ASCC	Taiwan	Xeon (2.4 GHz)/1 GB	2	1	0	0.52 MB/s	2.9
MIMOS	Malaysia	Athron (0.8 GHz)/2 GB	4	2	2	0.26 MB/s	1.7
cafe	Osaka	Dual Xeon (2.8 GHz)/2 GB	10	10	10	4.3 MB/s	1.1
tea	Osaka	Dual PentiumIII (1.4 GHz)/1 GB	20	20	10	4.1 MB/s	1.9
ume	Tsukuba	Dual PentiumIII (1.4 GHz)/1 GB	10	10	5	17 MB/s	1.7
f32	Tsukuba	Dual Xeon (3.1 GHz)/4 GB	20	20	20	38 MB/s	1.00
p32	Tsukuba	Dual Opteron (2.0 GHz)/6 GB	20	20	20	49 MB/s	0.85
m64	Tsukuba	Quad Itanium (1.3 GHz)/16 GB	10	20	10	27 MB/s	0.73

困難になる．古典近似二量体は標準精度計算と同様，50 個ずつまとめて 1 つのジョブとした．

計算には PRAGMA routine-basis testbed [12] と AIST スーパクラスタの計算資源を組み合わせ，ワーカとして利用した．これらは環太平洋地域に配された，合計 10 台のクラスタ計算機（M）から構成される．各計算機の仕様を**表 2** にまとめた．N_{node}，$N_{\mathrm{wk}}^{\mathrm{std}}$，$N_{\mathrm{wk}}^{\mathrm{high}}$ は各計算機で用いたノード数，標準精度計算でのワーカ数，高精度計算でのワーカ数のそれぞれ最大値であり，B/W にはマスタ–ワーカ間の通信速度を示した．電子状態計算の計算時間は，CPU とメモリ量だけでなく，メモリへのアクセス速度やディスク I/O の性能にも依存する．そこで計算機ごとで事前に小規模なベンチマーク試験を行い，所要時間を計測してその相対値を BM に示した．値が小さいほど高性能となる．アプリケーション本体（C），Bookkeeper（BK），Doorkeeper（DK）は p32 上の単一ノード上で独立のプロセスとして運用し，通信には Unix Socket を用いた．

標準精度の計算では単純に 1 ワーカあたり 2 CPU を割り当てた．すなわち，ASCC と MIMOS は 2 ノードで 1 ワーカ，m64 はノードあたり 2 ワーカ，他は 1 ノード 1 ワーカの構成とした．高精度計算では負荷分散を改善するため，BM 値を参考に SDSC，NCSA，MIMOS，tea，ume の各マシンで 2 ノード 1 ワーカ，他は 1 ノード 1 ワーカの構成とし，ASCC の使用は断念した．今回はジョブ数がワーカ数に比べて十分多いため，E_i，E_{ij}，$\tilde{E}_{ij}$ の計算を切り替える際，ワーカ構成は特に変更しなかった．

4. 結果と考察

4.1 計算時間

標準精度の計算は 2006 年 7 月から 8 月前半にかけて断続的に実施した．総計算時間は 24 日に及び，この間 50 万個を超えるジョブを処理した．高精度計算は 8 月後半から 9 月にかけて連続的に実施した．総計算時間は 46 日，ジョブ実行数は 20 万回であった．

標準精度・高精度の計算から，全計算時間が最短の例をそれぞれ 1 点選び，各部の計算時間を解析して表 1 にまとめた．表中，エネルギーの計算時間 $T(E)$ は入出力データの転送時間を含み，環境データの転送時間 T(env) にはさらに転送先での展開と配布に要した時間が含まれる．表 1 の計算例は，計算期間中ほぼすべての計算資源を滞りなく利用できたケースで，利用可能なワーカ数はおおよそその最大値N_{wk}に保たれた．その他の計算例では，バッチキューの混雑やメンテナンスなどの要因でワーカ数は変動し，計算時間は 1 ～ 2 割程度，余計にかかるのが通例であった．SCC サイクルは 3l–32 回で収束し，二量体の計算には SCC 全体の 1.5 ～ 2 倍の時間を要した．SCC サイクルの収束判定や全エネルギーの計算などの逐次部分が占める割合は，M の大きな標準精度の計算で全体の 2%程度，高精度計算では 1%未満に収まった．

標準精度計算，高精度計算ともに SCC サイクル終端での同期時間 T(sync) は最小フラグメントの計算時間より長くなっている．しかし E_i の計算時間の分布を詳しく調べると，ワーカ数を N_{wk} として，計算時間の短い方から N_{wk} 番目の E_i の計算時間が T(sync) とほぼ一致しており，LFF スケジューリングは正常に機能していることが分かる．このため，大断片用に資源提供量を調節する機能は用いなかった．高精度計算を標準精度と同じワーカ構成で実行したケースでは，T(sync) は 19 分になった．この場合も LFF スケジューリングは機能していたが，ワーカの計算性能が揃わないため，アプリケーションが予測したE_iの計算量と実際の計算時間が対応せず，同期時間が延びたと考えられる．ベンチマーク試験の結果をもとに計算機ごとの資源提供量を調整した結果，T(sync) は 8 分に

短縮された．

性能の異なる計算機を混合して利用しており，また計算資源の量が変動することから，FMO 計算のグリッド化にともなうオーバヘッドの算定は簡単ではない．ここでは表 1 に示した標準精度の計算例について，ワーカ数が期間中，その最大値である $N_{\mathrm{wk}} = 119$ に保たれたと仮定し，検討を進める．オリジナルの GAMESS FMO の実装を用い，p32 上に 100 ノード（200 CPU）確保して 100 ワーカ構成で標準精度の計算を実行すると，52.8 時間を要した．GridFMO では 119 ワーカ（238 CPU）で 64.6 時間かかっており，これを単純に 100 ワーカに換算すると 76.8 時間となる．ここから，GridFMO の計算時間に占めるグリッド化のオーバヘッドの割合は 30%程度と見積もられる．一方，表 2 の BM 値からテストベッドの実効性能を評価すると，p32 換算で 80 ノード相当となる．GridFMO の計算時間を p32 100 ノードに単純換算すると 51.7 時間になり，オーバヘッドは 0 となる．ワーカの計算性能を揃えた高精度計算では，全ジョブ中，ローカルサイト（すなわち表 2 中 Tsukuba に所在する計算機上）で処理されるジョブの割合は 65%で，ワーカ数の比率と大体一致する．つまり GridFMO 計算は，環境データの転送を考慮に入れても計算インテンシブな手法であることは間違いなく，オーバヘッドは小さいことが期待できる．しかし客観的な評価は困難と思われるので，ここではオーバヘッドは最悪でも 30%程度とするにとどめる．

4.2 耐障害性

GridFMO の実装は，対話的な計算資源の利用停止やバッチキューのタイムアウトにともなう停止も広く障害として検出し，中断されたジョブを漏れなく他の計算機に割り付けて再実行する．このため，使用中の計算機にメンテナンスが予定されている場合でも，その直前まで利用を続けることができる．また今回使用した計算機の中には，バッチキューのタイムアウトが 24 時間と短く設定されたものが含まれており，長時間の計算には適さないが，GridFMO 法ではこれを活用することができる．

70 日の計算期間中の障害の内訳は，停電やハードディスクの故障などの機械的な障害が約 30 回，通信障害が約 20 回，バッチキューのタイムアウトが約 80 回，メンテナンスが約 10 回であった．これらの障害の検出・対処に失敗すると，アプリケーション全体を停止してリスタートしなければならない．標準精度の計算では耐障害性の実装が不十分だったため，7 点中 4 点の計算で 1 ～ 2 回のリスタートが必要であった．高精度計算の実行時は耐障害性の向上にともない，7 点の計算を通じてリスタートは 1 回にとどまった．

障害の原因のほとんどは自明なものであったが，長期間の実験を行ってはじめて検出されたケースが 3 例あったので，参考のため記録する．

(1) 停電で停止した計算機に対し計算資源を再確保しようと Ninf-G ハンドルの作成を試みたところ，`globus_gram_client_job_request()` 関数が返ってこなかった．

(2) ディスク障害の発生した計算機に対し，計算資源の利用停止にともなって Ninf-G ハンドルの廃棄を試みたところ，`globus_gram_client_job_cancel()` 関数が返ってこなかった．

(3) 細い回線を経由して環境データを転送中，ごくまれに転送が中断したまま再開しないことがあった．

(1)，(2) はアプリケーションのリスタートが必要となる障害で，ハンドルの生成・廃棄を別スレッドに隔離することで対処した．またこれらの問題は Ninf-G 4.2 では解決済みである．

(3) は特定のサイトに高精度計算の環境データ（338 MB）を転送する際に発生し，調査した範囲では通信路の障害と推測されるが，その根本的な原因は未解明である．小型のパケットは問題なく通信できるので，Ninf-G のハートビート機構を用いても検出できない．しかしながら GridFMO は (3) に対し，その実装に由来する耐障害性を備えており，アプリケーションの実行は影響を受けない．すなわち，GridFMO では計算機（M）への環境データの転送が終わらない限りジョブは割り付けられないので，症状の出た M は実効的に使用停止となる．このため，現状では通信時間から判断して症状を検出し，Bookkeeper の対話的インタフェースを通じて発症した M を個別に停止・再起動することで対処している．上述の手続きの自動化はさほど困難ではないが，症状がまれにしか発生しないこと，および原因追及のため，いまのところ特別な対策はとっていない．Ninf-G レベルでの対策としては，I/O アクセスに汎用なタイムアウト機構の実装を検討中である．

4.3 資源共有

今回の実験は占有環境を作らずに行い，特に PRAGMA の計算資源は他の 1 グループと共有しながら計算を進めた．計算資源の共有は各計算機上のバッチキューシステムを介し，競争的に実施した．GridFMO は計算期間中，計 271 回のキュー登録を行い，登録から実行に移るまでの待機時間は平均 2.3 時間，最長 62

時間であった．待機時間はジョブの処理時間に比べて有意に長く，共有環境では待機時間の隠蔽が必須となることが分かる．

キュー登録後の待機時間は資源の要求量と密接に関連する．キューが混雑している状況では，大量の資源を要求すると長時間待機することになり，場合によってはアプリケーション実行中に間に合わない場合もある．今回の実験では，計算機資源の占有状況に応じて資源要求量を適宜調整し，計算資源の確保に努めた．調整は対話的なインタフェースを通じて手作業で行った．表 2 に N_{node} として使用ノードの最大数をあげたのは，このように使用ノード数を調整したことによる．同様の調整はメンテナンスにともなって計算機のキュー構成が変更された際にも必要となった．将来的に計算機資源の占有状況を調べる手段が提供されれば，より簡便な資源確保が実現可能と期待される．

PRAGMA の計算資源を共有した二者の間でバッチキューの利用形態はかなり異なっていた．資源要求の際，我々は多ノード長時間の要求を単発で登録（P）するのに対し，他方は単ノード短時間の要求を多重に登録（S）する．実験を通じ，健全な資源共有にはバッチキューに対して，実実行時間の上限の設定と FIFO 的な性格の付与が重要であることが分かった．実実行時間の制限は，P による計算ノードの占有を防ぐ．上限は計算の粒度に応じて定める必要があるが，アプリケーションに耐障害性が備わっていれば，キューシステムによる強制終了に十分，対応可能なことが分かった．一方，FIFO 性は P の実行を補助するために必要となる．計算資源の利用効率を重視するスケジューリングでは，S が連続的に登録されると，資源に空きが生じても後発の S がただちにこれを埋めるため，P を登録しても必要な資源量が確保できないまま，キュー内部に滞留することになる．FIFO 型のスケジューリングでは空き資源が P の実行のために予約されるので，利用効率は犠牲になるが資源共有は可能になる．将来的には計算資源の予約機構を用いることで，より良い解決策が得られると期待している．

資源を共有する場合は特に，利用終了後の計算ノードに廃残プロセスや一時ファイルを残さないよう，気を付けなければならない．しかしバッチキューシステムのタイムアウトで強制終了される場合，プロセスが抹消される過程はキューシステムに応じて様々であり，シグナル処理では対処しきれない．利用後の後始末は本来，キューシステムが担保すべき問題であるが，GridFMO では環境データ転送時に番兵プロセスを立ち上げ，非常時の後処理にあてた．

5. ま と め

FMO 法をグリッド技術を用いて実装し，大規模な量子化学計算を広域分散環境の下で効率良く実行できることを示した．実際の計算では占有環境を構築せず，各計算機上のバッチキューシステムを介して計算資源を他ユーザと共有しながら確保し，これを糾合して用いた．キューから実行に移された計算資源をつねに把握し，順次ジョブを割り付けるため，柔軟な資源管理機構を備えたミドルウェアを開発した．実装には Ninf-G を用い，その強力な障害検出機能を活用するとともに，検出した障害を集中管理することで，高い耐障害性を実現した．開発したシステムを用いて大規模な実験を実施し，グリッド技術を用いることで広域分散環境における計算を長期間，安定に継続できることを実証した．実験で使用した計算機は比較的小規模なクラスタ計算機で，なかにはバッチキューのタイムアウトが短く，長時間の計算には適さないものも含まれる．こうした計算機群をつなぎあわせ，各時点で利用可能な計算資源を糾合することで，大型計算機を利用しなくても大規模計算が可能であることを示した．

今回の実験には 10 台のクラスタ計算機を用いたが，これはどこまでスケールアップが可能だろうか．高精度計算を例にとると，計算時間の 6 割は二量体の計算に費やされた．このステップは処理すべきジョブ数が膨大なので，当面はワーカ数に比例して処理効率の上昇が見込める．一方，SCC サイクルは断片の総数 M 以上，ワーカがあっても無駄になる．現実的には LFF スケジューリングの都合から，ワーカ数は $M/3 \sim M/2$ 程度が上限と考えられる．したがって，より大量の計算資源が利用可能な局面では，ワーカ数を増やすよりも個々のワーカの並列度を上げる方が効率的だろう．ごく粗い推算ではあるが，1 ～ 2 万原子クラスの系を対象とする場合，クラスタ 50 台くらいまでは線形的な性能向上を見込むことができる．それ以上では，細粒度並列部分の効率の低下や環境データ転送時の通信の輻輳が原因となり，性能の劣化が顕著になると思われる．

互いに独立な FMO 計算を複数，実行しなければならないケースでは，複数の GridFMO アプリケーションを Bookkeeper，Doorkeeper を共有しながら同時に走らせることも考えられる．この場合，個々のアプリケーションで LFF スケジューリングが成立しなくても，暇になったワーカには他のアプリケーションのジョブが割り付けられるため，計算資源が無駄になることはない．さらに環境データを転送している間，他

のアプリケーションのジョブを処理させることで，効果的に通信時間を隠蔽することも可能である．現在の実装では，環境データの転送に用いる広報ジョブはアプリケーションにまたがって排他的に実行されるため，アプリケーション間で効率良く資源共有することはできない．今後，より限定的な排他機構を導入して改善を図る予定である．

謝辞 実験の際，実行環境のセットアップと維持には PRAGMA testbed 管理者の皆様のご理解とご協力が不可欠でした．この場を借りて深く感謝します．

参 考 文 献

1) Kitaura, K., Ikeo, E., Asada, T., Nakano, T. and Uebayasi, M.: Fragment molecular orbital method: An approximate computational method for large molecules, *Chem. Phys. Lett.*, Vol.313, pp.701–706 (1999).
2) Ikegami, T., Ishida, T., Fedorov, D.G., Kitaura, K., Inadomi, Y., Umeda, H., Yokokawa, M. and Sekiguchi, S.: Full Electron Calculation Beyond 20,000 Atoms: Ground Electronic State of Photosynthetic Proteins, *Proc. Supercomputing 2005* (2005). http://sc05.supercomputing.org
3) Nakano, T., Kaminuma, T., Sato, T., Akiyama, Y., Uebayasi, M. and Kitaura, K.: Fragment molecular orbital method: Application to polypeptides, *Chem. Phys. Lett.*, Vol.318, pp.614–618 (2000).
4) Fedorov, D.G., Olson, R.M., Kitaura, K., Gordon, M.S. and Koseki, S.: A New Hierarchical Parallelization Scheme: Generalized Distributed Data Interface (GDDI), and an Application to the Fragment Molecular Orbital Method (FMO), *J. Comp. Chem.*, Vol.25, pp.872–880 (2004).
5) 池上 努, 武宮 博, 長嶋雲兵, 田中良夫, 関口智嗣：Grid：広域分散並列処理環境での高精度分子シミュレーション，情報処理学会，Vol.44, pp.14–22 (2003).
6) Tanaka, Y., Nakada, H., Sekiguchi, S., Suzumura, T. and Matsuoka, S.: Ninf-G: A Reference Implementation of RPC-based Programming Middleware for Grid Computing, *J. Grid Comp.*, Vol.1, pp.41–51 (2003).
7) Seymour, K., Nakada, H., Matsuoka, S., Dongarra, J., Lee, C. and Casanova, H.: Overview of GridRPC: A Remote Procedure Call API for Grid Computing, *Lect. Note Comp. Sci.*, Vol.2536, pp.274–278 (2002).
8) 谷村勇輔, 池上 努, 中田秀基, 田中良夫, 関口智嗣：耐障害性を考慮した Ninf-G アプリケーションの実装と評価，情報処理学会，Vol.46, pp.18–27 (2005).
9) Komeiji, Y., Haraguchi, M. and Umpei, N.: Parallel molecular dynamics simulation of a protein, *Parallel Computing*, Vol.27, pp.977–987 (2001).
10) Schmidt, M.W., Baldridge, K.K., Boatz, J.A., Elbert, S.T., Gordon, M.S., Jensen, J.H., Koseki, S., Matsunaga, N., Nguyen, K.A., Su, S.J., Windus, T.L., Dupuis, M. and Montgomery, J.A.: General Atomic and Molecular Electronic Structure System, *J. Comp. Chem.*, Vol.14, pp.1347–1363 (1993).
11) Fletcher, G.D., Schmidt, M.W., Bode, B.M. and Gordon, M.S.: The Distributed Data Interface in GAMESS, *Comp. Phys. Comm.*, Vol.128, pp.190–200 (2000).
12) PRAGMA: *Pacific Rim Applications and Grid Middleware Assembly.* http://www.pragma-grid.net

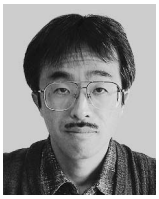
(平成 18 年 10 月 10 日受付)
(平成 19 年 2 月 13 日採録)

池上 努

昭和 42 年生．平成元年東京大学理学部化学科卒業．平成 3 年東京大学大学院理学系研究科修士課程修了，化学専攻．平成 4 年より慶應義塾大学理工学部化学科助手，岡崎国立共同研究機構分子科学研究所助手，Universidad de Puerto Rico ポスドクを経て，平成 14 年 6 月より独立行政法人産業技術総合研究所グリッド研究センター研究員．博士（理学）．計算化学，大規模数値計算の研究に従事．日本化学会，日本物理学会各会員．

真木 淳

昭和 42 年生．平成 11 年北海道大学大学院理学研究科博士後期課程単位取得退学．同年科学技術振興事業団計算科学技術研究員．平成 13 年岡崎国立共同研究機構分子科学研究所非常勤研究員．平成 17 年 4 月より九州大学情報基盤センター学術研究員．博士（理学）．「最先端・高性能汎用スーパーコンピュータの開発利用プロジェクト」においてナノアプリケーションのグリッド化に関する研究に従事．



高見 利也（正会員）

昭和 40 年生．平成 6 年京都大学大学院理学研究科博士後期課程中退．同年岡崎国立共同利用研究機構分子科学研究所電子計算機センター助手．平成 9 年総合研究大学院大学数物科学専攻助手（併任）．平成 16 年より九州大学情報基盤センター学術研究員．博士（理学）．テーマは非線形ダイナミクスと量子・古典対応，マルチフィジックス連成計算の理論等．日本物理学会会員．

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主幹研究員．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およびグリッドセキュリティに関する研究に従事．IC 1999，HPCS 2005，SACSIS 2006 論文賞．ACM 会員．

横川三津夫（正会員）

昭和 35 年生．昭和 59 年筑波大学大学院修士課程理工学研究科修了．同年日本原子力研究所入所．平成 9 年地球シミュレータ研究開発センターにてハードウェア開発に従事．平成 14 年産業技術総合研究所グリッド研究センター．平成 18 年理化学研究所次世代スーパーコンピュータ開発実施本部．平成 6 年〜7 年コーネル大学コーネル理論センター客員研究員．工学博士．大規模数値シミュレーション，並列数値アルゴリズムの研究に従事．業績賞，SC2002 ゴードン・ベル賞受賞．日本応用数理学会会員．

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．

青柳 睦

昭和 34 年生．昭和 60 年慶應義塾大学大学院理工学研究科修士課程修了．昭和 62 年名古屋大学大学院理学研究科後期博士課程単位取得退学．岡崎国立共同研究機構分子科学研究所入所．アルゴンヌ国立研究所を経て，平成 14 年 4 月より九州大学情報基盤センター．現在同大学同センター教授．博士（理学）．計算分子科学，並列処理，グリッドに関する研究に従事．日本化学会会員．