CONFLEX-G：OmniRPCによるグリッド環境上での分子立体配座探索プログラムの実装と性能評価

中島佳宏[†] 佐藤三久[††] 後藤仁志[†††]
朴泰祐[††] 高橋大介[††]

グリッド環境での並列プログラミングのための Grid RPC システムである OmniRPC を使用し分子配座空間探索プログラム CONFLEX のグリッド環境への実装と性能評価を行った．CONFLEX の分子配座探索において処理時間の大部分を占める試行構造最適化をグリッド環境の計算資源に割り当て，全体のスループットの向上を可能とした．OmniRPC を使用した CONFLEX-G と MPI 版の CONFLEX をローカルな PC クラスタで性能評価を行い，ほぼ同じ性能が得られることを確認した．また，広域ネットワーク環境で複数のクラスタを使用し性能評価を行い，分子の原子数が大きく，1 つの試行構造最適化の処理時間が長い場合において性能向上を確認した．1BL1 分子の配座探索において CONFLEX-G は 56 倍高速に処理することができた．さらに，試行構造最適化にかかる時間のばらつきが高速化を妨げる原因になっているということも分かった．

Implementation and Performance Evaluation of CONFLEX-G: A Grid Enabled Conformational Space Search Program by OmniRPC

Yoshihiro Nakajima,[†] Mitsuhisa Sato,[††] Hitoshi Goto,[†††]
Taisuke Boku[††] and Daisuke Takahashi[††]

CONFLEX-G is the grid-enabled version of a molecular conformational space search program, CONFLEX. We have implemented CONFLEX-G by using a grid RPC system named OmniRPC. In this paper, we report its performance in our grid testbed of several clusters geographically distributed. To explore the conformation of large bio-molecules, CONFLEX-G generates trial structures of the molecules, and allocates jobs to optimize a trial structure by molecular mechanics in the grid. Comparing to the CONFLEX MPI version, CONFLEX-G can achieve comparable performance to the MPI version, and exploit more computing resources by allowing the use of a cluster of multiple clusters in the grid. The experimental result shows that CONFLEX-G achieved 56.5 times speedup in case of 1BL1 molecule where the molecule consists of large number of atoms and each trial structure optimization takes long time. It is also found that the load imbalance of the optimization time of the trial structure may cause performance degradation.

1. はじめに

インターネットをはじめとする広域ネットワークの進歩により，ネットワーク上の計算資源やデータ共有，並列コンピューティングの支援を可能とするグリッド技術が注目されている．それにともない，そのグリッド技術を用いて大規模・複雑な問題を扱うアプリケーションが開発されてきている．その代表的な例としては，広域ネットワーク環境で複数の計算機・クラスタを利用して大規模・複雑な問題を解く Computational Grid や散在する遊休計算資源を利用し大規模な問題を解く seti@home[14]，folding@home[9] があげられる．

我々は，分子の配座空間探索プログラム CONFLEX を開発している[5),6)]．CONFLEX は Molecular Mechanics を使用して構造最適化をしているため，MO（分子軌道）計算を用いるものよりも短時間で配座探索を行えるが，生体高分子を扱う場合には，依然として時間がかかることが予想される．さらに，これま

† 筑波大学大学院システム情報工学研究科
Graduate School of Systems and Information Engineering, University of Tsukuba
†† 筑波大学電子・情報工学系
Institute of Information Sciences and Electronics, University of Tsukuba
††† 豊橋技術科学大学知識情報工学系
Knowledge-based Information Engineering, Toyohashi University of Technology

で CONFLEX は，MPI を用いて配座探索の全処理の 90%以上を占める構造最適化を並列に計算していが，この方法だけでは十分な高速化は得られない．特に HIV プロテアーゼなどの生体高分子を対象とする場合には，莫大な数でしかも巨大な試行構造を処理しなくてはならず，1 つのクラスタを利用するだけでは十分ではない．

さらに，CONFLEX で行っている構造最適化の処理は，計算インテンシブな処理であり，グリッドの広域ネットワーク環境において多少のオーバヘッドがあったとしても，大規模な計算資源であるグリッド環境を利用することにより恩恵が得られると考えられる．そこで，我々は，OmniRPC を用いて，CONFLEX のグリッド並列化を行った．

本論文では，配座探索プログラム CONFLEX を生体分子の配座解析に適応するために，OmniRPC を用いた分子配座空間探索プログラム CONFLEX のグリッド環境への実装と，グリッド環境上でのその性能評価について述べる．

我々は，グリッド環境において複数の計算資源を用いて，簡単に並列分散プログラミングを行うための Grid RPC システムである OmniRPC[10),13),15)] を開発している．

現在，Globus Toolkit[2)] が標準的なグリッド環境ミドルウエアとして多く用いられている．Globus Toolkit は，遠隔の計算機環境においてジョブの起動のコマンドインタフェースを提供している．ユーザは，この提供されたインタフェースを使用し，簡単なジョブスクリプトを記述することができる．ほかにグリッド向けの並列プログラミング環境として，Globus Toolkit の MPI プログラミング環境 MPICH-G2[7)] があげられる．しかし，MPI を用いてパラメータサーチのような比較的簡単なプログラムを記述することは，開発者にとってプログラミングコストが大きい．また，MPICH-G2 では，すべての計算ノードが固定的に設置され，複数の計算機を占有できる環境が必要である．そのため，計算機資源の状況が変化するグリッド環境での利用は難しい場合がある．

並列プログラミングシステムの開発においては，開発者が簡便に，しかもグリッド環境上の計算機資源の割当てを意識しなくてもプログラムを開発できる方が望ましい．そのため，遠隔計算機資源でのプログラム実行を，遠隔手続き呼び出し（RPC: Remote Procedure Call）を用いて行う機構が提案されている．そこで現在，Global Grid Forum[3)] の GridRPC WG[4)] では，グリッド環境で RPC 呼び出しを用いて計算機を利用するプログラミングインタフェース GridRPC[8)] の標準化作業が行われている．OmniRPC は，Grid 向けの RPC で，Ninf システムから派生し，基本的な API として Ninf[11)] システムを踏襲している．ほかに，Grid 環境向け RPC としては，Ninf-G[12)] や NetSolve[1)]，Jojo[16)] があげられる．

2 章において，これまで開発されてきた分子立体配座探索プログラムである CONFLEX の概要を述べ，3 章では，分子立体配座プログラム CONFLEX-G のグリッド化の方法について述べる．4 章では，構築したグリッド環境ついて述べ，広域ネットワーク環境での CONFLEX-G の性能評価を述べる．5 章で今後の課題をあげる．

2. 分子立体配座探索プログラム CONFLEX

タンパク質の機能と立体構造との関係を明らかにすることは，特殊な構造を持つタンパク質に起因する病気の治療や，特定のタンパク質の機能を持つ新薬の開発に関わる重要なことである．また，タンパク質のフォールディング過程における分子の立体構造の変化，すなわち配座の遷移プロセスを予測することは，タンパク質の構造や機能を理解するうえで重要な課題である．しかし配座間の相互変換に関する遷移状態を観測することは困難であり，その配座反応経路を追跡するのは容易ではない．

今までの分子計算プログラムでは，ユーザが入力した初期構造に依存した，局所的な最適化構造しか求められず，また，最安定構造でさえ，特定することが困難であった．さらに，フレキシブルな分子の性質や挙動に関して，限られた情報しか得ることができなかった．これらの問題を解決するための配座探索プログラムが CONFLEX である．

この CONFLEX を用いることにより，ユーザは対象とする分子がとりうる立体配座を自動的に発生することができ，さらに，化学的に重要な配座の最適化構造を効率的にもれなく見つけ出すことができる．

CONFLEX において，配座探索計算における構造最適化は，Molecular Mechanics を利用して行う．これにより，様々な分子構造に対応した計算が可能となる．また，計算中頻繁に生成されるすでに見つかった構造，鏡像異性体，幾何異性体，鞍点構造，非常に不安定な構造などは自動的に省かれるようになっている．

本章では，CONFLEX の概要について説明する．まず CONFLEX の探索アルゴリズムを述べ，次に MPI を使った並列化の方法について述べる．

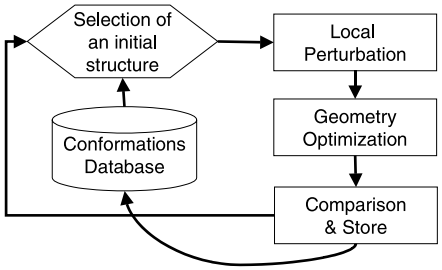

図 1 CONFLEX の処理
Fig. 1 The progress of CONFLEX.

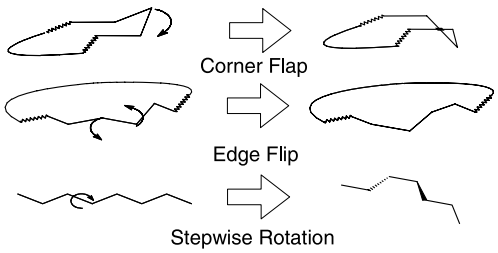

図 2 CONFLEX の試行構造作成の戦略
Fig. 2 These strategies are used to involve the local perturbations in CONFLEX.

2.1 CONFLEX の探索アルゴリズム

図 1 に CONFLEX の配座探索アルゴリズムを示す．

(1) すでに保存されている構造の中からの初期構造の選択（ただし最初だけ入力構造を初期構造として選択する）
(2) 選択された初期構造から適当な構造の生成
(3) 構造最適化
(4) すでに得られている構造と比較し，新しい配座をデータベースに保存

CONFLEX の終了条件を満たすまで以上の手順を繰り返す．

CONFLEX での最大の特徴は構造の生成の処理にある．なかでも環状部分に採用している Corner Flap および Edge Flip により，優れた探索を行うことができる．**図 2** に，その生成方法を示す．これらの操作は，初期構造を中心とした局所的な探索である．以下でその説明を行う．

- Corner Flap
  初期構造の環の構成原子から 1 つを選び，環の平均平面に対して反対側に移動させ新たな構造を生成する．
- Edge Flip
  初期構造の環の構成原子から隣接する 2 つを選び，環の平均平面に対してそれぞれ反対側に移動させることによって，ねじる操作を行う．さらに 2 つの原子を環の内側に動かし，へこませる操作も行う．
- Stepwise Rotation
  側鎖に対して，単純な回転操作を行い新しい出発構造を発生させる．

以上の 3 つの操作は，初期構造を中心とした局所的な探索である．

CONFLEX では，さらに見つかった配座異性体の中でエネルギーの低い配座から順に初期構造を選択する．この初期構造から Corner Flap などによって試行構造が生成され，それぞれの試行構造に対して構造最適化が行われる．またこの過程で，よりエネルギーの低い配座異性体が見つかれば，次にその配座を初期構造として選択し，配座探索が行われる．これにより，立体配座探索空間における構造のエネルギー状態は低くなる．さらに，局所的な安定配座になるのを防ぐため，最安定配座が見つかれば，探索領域を制限内で高いエネルギー状態の領域に広げる．これまでの経験則により，化学的に重要な領域を探索しつくすには，探索制限はエネルギー状態が最低のところから 7〜8 kcal/mol の辺が良いということが分かっている．以上の CONFLEX の探索のスキーム全体が，貯水池やダムに水が流れ込み，周りに水が満たされていく様子に似ているので，我々は，「Reservoir-Filling（貯水池注水）アルゴリズム」と呼んでいる．

2.2 CONFLEX の並列化

生体高分子をターゲットとして CONFLEX を適用するには，配座探索効率の向上が必要である．ただし，CONFLEX が配座空間の低エネルギー領域を効率的に網羅するかどうかは，CONFLEX 独自の戦略的探索手法に依存している．しかし，単純に時間がかかっている処理は，試行構造の構造最適化であり，この部分が，全探索時間の 90%以上を占めている．

したがって，この構造最適化の処理を高速化することにより，巨大な高分子に対しても配座探索を行えるようになると考えられる．そこで我々は，試行構造をマスタノードにプールしてから，空いているワーカノードに 1 つずつ試行構造を渡し，構造最適化の処理を行うマスタ/ワーカ型の並列化アプローチを採用し，MPI を用いて CONFLEX の実装を行った．そして，単一の PC クラスタにおいて比較的小さな分子の配座探索を行い，性能評価が行われた．そこで，並列化により高速化が達成できていることが確認されている[6]．

3. CONFLEX-G：CONFLEX のグリッド並列化

我々は，グリッド環境上で実行可能にした分子配座

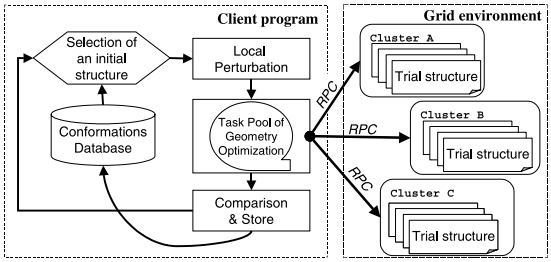

図 3 CONFLEX-G の処理
Fig. 3 The progress of CONFLEX-G.

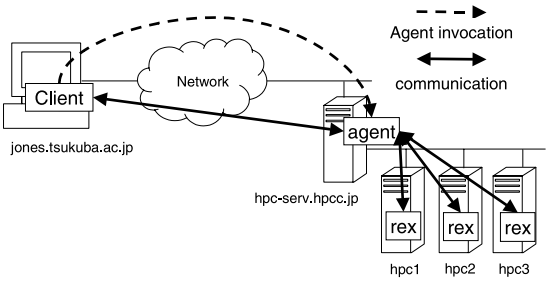

図 4 OmniRPC の動作
Fig. 4 The overview of OmniRPC system for the remote cluster with private IP address.

空間探索プログラムを CONFLEX-G と呼ぶ．これまで CONFLEX は MPI を用いて，ローカルなクラスタ上で実行されてきたが，今回は CONFLEX をグリッド環境上で展開・実行できるようにした．MPI 版 CONFLEX のアプローチと同じように，試行構造最適化の処理を OmniRPC を使ってリモートホストで構造最適化の計算を行うように，我々は CONFLEX のプログラムの変更を行った．CONFLEX-G の処理の流れを**図 3** に示す．

まず，CONFLEX-G で使用した OmniRPC について概要を述べ，CONFLEX-G の実装について述べる．

3.1 OmniRPC の概要

OmniRPC は，クラスタ環境から広域ネットワークで構成されたグリッド環境までシームレスな並列プログラミングを可能にする Grid RPC システムである．また，OmniRPC は，マスタ/ワーカ型の並列プログラミングをプログラミングモデルとしてサポートしている．

OmniRPC がターゲットとするグリッド環境は，複数の計算機クラスタがグリッド上に接続されており，それらのクラスタを相互利用する環境である．さらに，OmniRPC では，クラスタを構成するネットワークとしてプライベートアドレスを用いて構成されたクラスタについてもサポートしている．

OmniRPC の API は，基本的に Ninf の API を踏襲しており，さらに，リモート側の状態を保持する persistency をサポートしている．この persistency をサポートした API を利用することにより，ユーザは効率的なプログラミングが可能となる．また，ユーザは，非同期呼び出しの API を用いることにより並列プログラミングを行うことができる．

OmniRPC では，1 つのリモート実行プログラムは複数の関数を含むことができる．このため，ユーザは，インタフェースを記述する IDL 記述において，複数のインタフェースをまとめてモジュールとして定義することができる．また，グリッド環境において典型的な並列アプリケーションであるパラメータ検索などのアプリケーションを効率的にサポートするために，OmniRPC では，リモート実行モジュールの起動時に実行される初期化手続きを定義することによって，実行モジュールを起動時に自動で初期化する機能を有している．我々は，この機能を自動初期化実行モジュール機能と呼ぶ．この機能を利用することにより，リモート実行プログラムの初期化のための大量のデータ転送や計算を，2 回目の RPC 以降省くことができる．さらに，リモートプログラムは一度初期化したデータを再利用することができるようになるため，処理の効率化と余分なデータ転送を防ぐことができる．

OmniRPC は，エージェントを使用してクライアントプログラムと複数のリモート実行プログラムとの通信を多重化し 1 つのコネクションで行うことができる．この通信の多重化を利用することにより，ユーザは，プライベートアドレスで構築されたクラスタや 1000 台規模のリモートホストを利用することができる．

Grid 環境として，OmniRPC は，Globus Toolkit のほかに，SSH（Secure Shell）による認証も使用できる．さらに，SSH が用いられている環境においては，ポートが制限されていたり，使用するネットワークの途中に firewall が存在したりすることが多い．そのような環境のために，エージェントとクライアントプログラムの通信は SSH の port forwarding 機能を用いて中継させるため，ユーザは，firewall のある環境においても，リモート計算機資源を利用することができる．

OmniRPC の動作イメージを**図 4** に示す．この図では，通信の多重化が使用され，クラスタの計算ノードに向けて RPC 呼び出しが行われている．OmniRPC において，クライアントプログラムが起動されたときに，omrpc-agent は OmniRPC のホストファイルで指定されたホスト上で起動される．次に，クライアントプログラムの RPC 呼び出しに対して，omrpc-agent

```
<?xml version="1.0" ?>
<OmniRpcConfig>
  <Host name="hpc-serv.hpcc.jp" >
    <Agent invoker="ssh" mxio="on"/>
    <JobScheduler type="rr" maxjob="3"/>
  </Host>
</OmniRpcConfig>
```

図 5 OmniRPC のホストファイルの例
Fig. 5 An example of OmniRPC hostfile.

が計算ノードにあるリモート実行プログラムを起動する．

また，OmniRPC のホストファイルにおいて，omrpc-agent を起動させるホスト，一度に起動できる最大プロセス数を指定する．このホストファイルを実行環境に合わせて設定することにより，ユーザは，並列プログラムの開発をそれぞれのクラスタで行い，プログラムの変更なしにグリッド環境で実行することができる．OmniRPC のホストファイルの例を**図 5** に示す．この設定では，認証に SSH を用い，omrpc-agent を hpc-serv.hpcc.jp に起動させ，通信の多重化を行っている．さらに，ジョブスケジューリングとして Round-Robin を使い，一度に起動できる最大ジョブ数は 3 となっている．

3.2 CONFLEX-G の実装

CONFLEX-G の実行の概要を図 3 に示す．MPI 版 CONFLEX において，それぞれのクラスタの計算ノードで試行構造最適化をさせていた処理を，CONFLEX-G では RPC 呼び出しを用いて，グリッド上にあるクラスタの計算ノードに割り当て処理するように変更する．

CONFLEX-G でのワーカプログラムで行われる処理は 2 つある．1 つはワーカプログラムの初期化，もう 1 つは試行構造の構造最適化の部分である．

まず，ワーカプログラムの初期化には，OmniRPC が提供している自動初期化実行モジュール機能を使用する．自動初期化実行モジュール機能は，モジュールのリモート実行プログラムが起動されたときに自動的に初期化の関数を呼び出す機能である．ここで，一般的な RPC システムでは，ワーカプログラム側での persistent をサポートしていないため，RPC 呼び出しごとにプログラムの初期化を行わなくてはならない．

しかし，OmniRPC では，モジュール内に Initialize リモート関数が定義されている場合において，その他の RPC 呼び出しに対して新たなリモート実行プログラムが割り当てられ起動されるときに，Initialize 関数が自動的に呼び出される．そのため，この Initialize 関数に共通のデータを設定することで，それ以降の

```
OmniRpcInit()
OmniRpcModuleInit("conflex_search",..);
...
while( <new conformers> ) {
 foreach( <trial structures> )
  OmniRpcCallAsync("conflex_search_worker",..);
 OmniRpcWaitAll();
 ...
}
...
```

図 6 CONFLEX-G のクライアントプログラムの概略
Fig. 6 The outline of client program code in CONFLEX-G.

RPC 呼び出しにおいて，ワーカプログラムはデータを再利用することができ，処理を効率化できる．また，このセットアップするコストが大きいほど，その効果は大きくなる．

そこで，ワーカプログラムの Initialize リモート関数で，エネルギー評価のパラメータなどのデータをクライアントプログラムから受信しプログラムの初期化を行うことにした．また，グリッド環境での実行を考慮して，計算ノードにクライアント実行プログラムがあれば利用できるよう，CONFLEX-G では，初期化時にそのパラメータファイルの転送を行いワーカの初期化を行うようにした．

ワーカプログラムの試行構造の構造最適化の処理については，クライアントプログラムから，試行構造の原子配置，内部でのエネルギー状態のデータを受信し，その試行構造の構造最適化を行う．そして，その結果をクライアントプログラムに返すようにした．

このワーカプログラムでの構造最適化の計算部分はパラメータ独立で計算することができるため，クライアントプログラムにおいて，この部分を非同期遠隔手続き呼び出しを行い並列に処理するようにした．クライアントプログラムから構造最適化のルーチンを呼び出すためには，非同期手続き呼び出しの API `OmniRpcCallAsync` を使い，すべての遠隔関数呼び出しが終了するまで待つ API `OmniRpcWaitAll` を使って同期をとることにした．RPC 呼び出しを用いて試行構造最適化をクラスタの計算ノードに割り当てる．**図 6** にクライアントプログラムのコードの概略を示す．

ここで，OmniRPC には負荷分散機能があるため，アイドル状態になったワーカに順次割り当てられることになる．

4. CONFLEX-G の性能評価

4.1 実験環境

筑波大学，豊橋技術科学大学，産業技術総合研究所

表 1 グリッドテストベットの計算機環境
Table 1 Machine configurations in our grid testbed.

Site	Cluster Name	Machine	Network	Agent-Invoker	Node 数	C PU 数
筑波大学	Dennis	Dual Xeon 2.4 GHz	1 Gb Ethernet	Globus,SSH	14	28
	Alice	Dual Athlon 1800+	100 Mb Ethernet	Globus,SSH	18	36
豊橋技術科学大学	Toyo	Dual Athlon 2600+	100 Mb Ethernet	SSH	8	16
産業技術総合研究所	Ume	Dual Pentium3 1.4 GHz	1 Gb Ethernet	Globus, SSH	32	64

表 2 Dennis クラスタのマスタノードと各クラスタ間のネットワーク性能
Table 2 Network performance between master node of the Dennis cluster and master node of each PC cluster.

Cluster	Round-Trip Time (ms)	Throughput (Mbps)
Dennis	0.23	879.31
Alice	0.18	94.12
Toyo	11.27	1.53
Ume	1.07	373.33

（AIST）の 3 つのサイトにあるクラスタを使用し実験を行った．実験環境を**表 1** に示す．筑波大学と AIST 間は 1 Gbit の Tsukuba-WAN，筑波大学と豊橋技術大学間は SINET でそれぞれ結ばれている．つまり，Dennis クラスタと Ume クラスタは 1 Gbit，Dennis クラスタと Toyo クラスタは Tsukuba-WAN を経由し SINET，Dennis クラスタと Alice クラスタは 100Base Ether のネットワークで結ばれている．まず Dennis クラスタのマスタノードと各クラスタ間のネットワークの性能を明らかにするために *netperf*，*ping* を使用し測定を行った．その結果を**表 2** に示す．

CONFLEX-G の実験において，クライアントプログラムは筑波大学にある Dennis クラスタのマスタノードから実行している．ただし，RPC 呼び出しのためのデータ転送は，クライアントプログラムが動作している Dennis クラスタのマスタノードと，ワーカプログラムが動作しているクラスタの計算ノードで行われ，クラスタの計算ノード間での通信は行われない．また，認証方式に SSH を用いて，ジョブスケジューラは OmniRPC に内蔵されている Round-Robin スケジューラを使用している．ただし，SSH の port forwarding による中継は使用していない．さらに，クラスタにある計算ノードの 1 CPU に対して，1 つのリモート実行プログラムを割り当て実行させている．つまり，起動するワーカ数と使用する CPU 数は一致する．

CONFLEX の Version 402 revision r を参考にして，CONFLEX-G を作成した．また，プログラムは Intel Fortran Compiler 7.0，gcc2.95 を使用し作成している．比較対象の MPI 版 CONFLEX では，

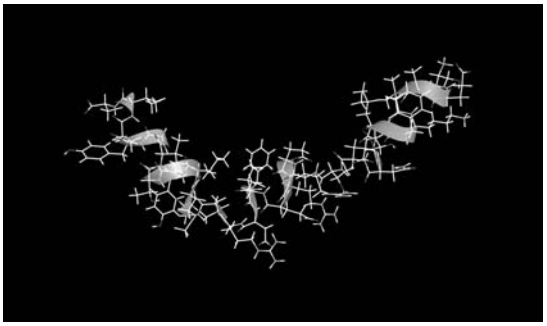

図 7 CONFLEX-G で配座探索を行った分子（1BL1）
Fig. 7 NMR structure of PTH receptor N-Terminus fragment.

表 3 性能評価で使用した分子の概要
Table 3 The characteristics of sample molecules for experiments.

Molecular code	# of atoms	# of all trial structures	trial structures /loop
AlaX04	181	360	45
AlaX16	191	480	160
1L2Y	315	331	331
1BL1	519	519	519

表 4 分子の構造最適化に必要なデータ転送量
Table 4 Data transmission for optimizing trial molecular structures in each molecular code.

Molecular code	Data transfer to initialize a worker	Data transfer /trial structure
AlaX04	2,033 KB	17.00 KB
AlaX16	2,063 KB	18.14 KB
1L2Y	2,099 KB	29.58 KB
1BL1	2,150 KB	48.67 KB

MPICH version 1.2.5 を使用した．

CONFLEX-G のテストデータとして，アラニン 4 量体（AlaX04），同 16 量体（AlaX16），TRP-CAGE モチーフを取る合成ミニタンパク質（1L2Y），**図 7** に示した甲状腺ホルモンレセプタ（1BL1）の 4 つのペプチドおよび小タンパク質を使用した．使用した分子サンプルの詳細を**表 3** に示す．この表の trial structures/loop は，各繰返しステップにおいて生成される試行構造数で，並列に処理される値であり，並列度を意味する．また，今回の実験では，CONFLEX 構造最適化のエネルギー評価として MM2 力場を用いた．**表 4** に，

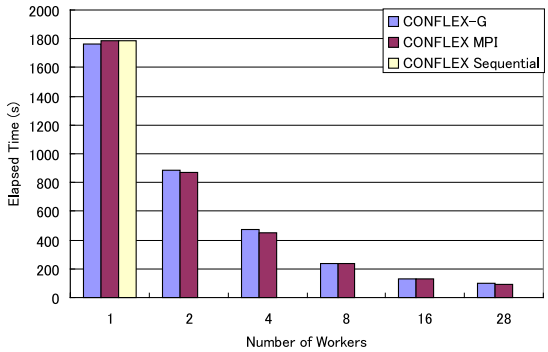

図 8 Dennis クラスタにおいてサンプル分子 AlaX04 を CONFLEX-G と MPI 版 CONFLEX で配座探索した実行時間

Fig. 8 The performance of CONFLEX-G and CONFLEX MPI version and Original CONFLEX in the Dennis cluster.

図 9 Dennis クラスタの 1 ワーカの性能を 1 としたときにグリッド環境で実行した際の Speedup

Fig. 9 The speed up ratio, which was based on the elapsed time of CONFLEX-G by using 1 worker in Dennis cluster, in our grid testbed.

CONFLEX-G がワーカプログラムの初期化に必要なデータ転送量，1 試行構造を最適化する際に必要なデータ転送量を示す．ここで，1 試行構造を最適化する際に必要なデータ転送量は MPI 版 CONFLEX と CONFLEX-G は同じである．

4.2 CONFLEX-G と MPI 版 CONFLEX との比較

ローカルな PC クラスタにおいて，OmniRPC を使って実装した CONFLEX-G と，従来の MPI 版 CONFLEX，また並列化を行っていない CONFLEX との比較を行う．筑波大学にある Dennis クラスタを使用し，ワーカ数を変更することによってどのように性能が変化するか調べる．この比較実験では，AlaX04 をテスト分子として使用した．実行結果を**図 8** に示す．

実験の結果より，CONFLEX-G は MPI 版 CONFLEX と同じようにワーカ数が増えるに従って実行時間を短縮できることが分かった．ちなみに，CONFLEX-G では，28 ワーカを使用した際に，サンプル分子 AlaX04 では 18.00 倍の高速化が達成できている．

CONFLEX-G は，ワーカ数を増やしていった場合に MPI 版 CONFLEX と比較して，性能が少し低い．これは，MPI 版 CONFLEX と CONFLEX-G の初期化を行うプロセスの違いによって起こると考えられる．ここで，MPI 版 CONFLEX では構造最適化を行う前のフェーズで，すべてのワーカプログラムはローカルファイルシステムからエネルギー評価のためのパラメータファイルを読み込み，初期化が終了する．これに対し，CONFLEX-G では試行構造の構造最適化がワーカプログラムに初めて割り当てられたときに，実行時にプログラムを起動し，パラメータファイルをネットワーク転送して初期化する．そのため，CONFLEX-G では，ワーカプログラムの初期化というオーバヘッドが発生する．また，表 4 に示すように，CONFLEX-G は初期化において約 2 MB のファイルをワーカプログラムに転送しており，ワーカプログラムの初期化に約 2 秒かかる．さらに，使用した AlaX04 分子は小さく，その分子から生成された 1 試行構造最適化時間は短いため，初期化のオーバヘッドが全体に占める割合が大きくなってしまう．よって，CONFLEX-G は MPI 版 CONFLEX と比較して性能が低くなっている．

しかし，本来 CONFLEX-G は，生体高分子の分子立体配座探索を目的としているため，巨大分子の配座探索には，構造最適化数の増加やその分子から生成された試行構造最適化の処理時間の増加が予想される．よって，CONFLEX-G ではワーカプログラムの起動やその初期化にかかるオーバヘッドの全体に占める割合が減るため，CONFLEX-G は MPI 版 CONFLEX と同程度の性能が得られると考えられる．

4.3 グリッド環境での性能評価

グリッド環境において，Dennis，Alice，Ume，Toyo クラスタを同時に用いることにより性能評価を行った．まずサンプル分子 AlaX04，AlaX16 を使用し，その実験で Dennis クラスタの 1 ワーカと比較してどのくらい高速化できたかを**図 9** に示す．また，その詳細結果をそれぞれ**表 5**，**表 6** に示す．

両方の場合において，Dennis と Alice クラスタで 64 ワーカを使用したときに最大の性能が得られた．ちなみに，CONFLEX-G は，AlaX04 のときに最大で

表 5 AlaX04 分子の構造最適化の実行時間
Table 5 The elapsed time of searching AlaX04's molecular conformation.

Cluster (# of workers)	Total # of workers	Structures /worker	Total optimization time (s)	Optimization time /structure (s)	Elapsed time (s)	Speed up
Dennis (sequential)	1	320.0	1,786.21	4.96	1,786.21	1.00
Toyo (16)	16	20.0	1,497.08	4.16	196.32	9.10
Dennis (28)	28	11.4	1,905.51	5.29	97.00	18.41
Alice (36)	36	8.9	2,055.25	5.71	87.09	20.51
Ume (56)	56	5.7	2,196.77	6.10	120.69	14.80
Dennis (28) + Toyo (16)	44	7.3	1,630.09	4.53	162.35	11.00
Alice (36) + Toyo (16)	52	6.2	1,774.53	4.93	178.24	10.02
Dennis (28) + Alice (36)	64	5.0	1,999.02	5.55	81.52	21.91
Dennis (28) + Ume (56)	84	3.8	2,085.84	5.79	92.22	19.37
Alice (36) + Ume (56)	92	3.5	2,120.87	5.89	101.25	17.64

表 6 AlaX16 分子の構造最適化の実行時間
Table 6 The elapsed time of searching AlaX16's molecular conformation.

Cluster (# of workers)	Total # of workers	Structures /worker	Total optimization time (s)	Optimization time /structure (s)	Elapsed time (s)	Speed up
Dennis (1)	1	480.0	74,027.80	154.22	74,027.80	1.00
Toyo (16)	16	30.0	70,414.21	146.70	4,699.15	15.75
Dennis (28)	28	17.1	74,027.80	154.22	3,375.60	21.93
Alice (36)	36	13.3	90,047.27	187.60	3,260.41	22.71
Ume (56)	56	8.6	123,399.38	257.08	2,913.63	25.41
Dennis (28) + Toyo (16)	44	10.9	76,747.74	159.89	2,762.10	26.80
Alice (36) + Toyo (16)	52	9.2	82,700.44	172.29	2,246.73	32.95
Dennis (28) + Alice (36)	64	7.5	87,571.30	182.44	2,051.50	36.08
Toyo (16) + Ume (56)	72	6.7	109,671.32	228.48	2,617.85	28.28
Dennis (28) + Ume (56)	84	5.7	102,817.90	214.20	2,478.93	29.86
Dennis(28)+Ume(56)+Toyo(16)	100	4.8	98,238.07	204.66	2,478.93	29.86

21.91 倍，AlaX16 のときに最大で 36.08 倍の高速化が得られている．

AlaX04 分子の場合には，クラスタ間のネットワーク性能が高い場合にのみ性能向上が得られている．しかし，広域ネットワーク環境において複数のクラスタを利用してもほとんど高速化が得られていない．これは，小さな分子構造である AlaX04 分子から生成された 1 試行構造最適化時間は短く，さらに，ワーカプログラムの起動や，ネットワークデータ転送のオーバヘッドが全体の処理の大部分を占めるためである．

特に，ワーカプログラムの初期化に必要なデータは約 2 MB あり，ネットワークの性能が悪い Toyo クラスタのワーカプログラムへ転送は，約 6.7 秒もかかっている．この転送時間は 1 構造最適化の処理時間よりも長いため，CONFLEX-G では，このデータ転送オーバヘッドにかかる時間が大部分を占めている．そのため，CONFLEX-G は，広域ネットワーク環境で大規模な計算ノードを使用してもグリッド並列化の恩恵が得られていない．

AlaX16 の場合には，広域ネットワークにおいても複数のクラスタを使用して，性能向上が得られている．これは，リモートでの計算時間が長いため，ネットワーク遅延などのオーバヘッドを隠蔽できているからである．しかし，CONFLEX-G が複数のクラスタを使用した際にあまり高速化が得られていないのは，ワーカに割り当てられている最適化実行時間のばらつきによるためである．ここで，最高の性能向上が得られた Dennis クラスタと Alice クラスタを使用した場合においても，1 試行構造適化の時間は，最短で約 60 秒，最長で約 550 秒と，約 9 倍もの差があった．よって，CONFLEX-G では，他のワーカが計算を終了させているのにもかかわらず，1 つのジョブを待ち合わせしているため全体として性能向上が改善されなかった．

最後に，大量のワーカを使うことによって，処理の高速化が見込める巨大な分子に対して CONFLEX-G を用いて配座探索を行った．グリッド環境において，Dennis，Ume クラスタを使用し性能評価を行った．それぞれサンプル分子に 1L2Y，1BL1 を使用し，Toyo クラスタで MPI 版 CONFLEX を実行した場合と，グリッド環境で CONFLEX-G を実行した結果をそれぞ

表 7 IL2Y 分子の構造最適化の実行時間
Table 7 The elapsed time of searching trial structure of 1L2Y.

Cluster (# of workers)	Total # of workers	Structures /worker	Optimization time /structure (s)	Elapsed time (s)	Elapsed time (H)	Speed up
Toyo MPI (16)	16	20.7	867	18,696	5.19	15.34
Dennis (28)	28	11.8	803	14,101	3.91	20.35
Dennis (28) + Ume(56)	84	3.9	1,064	8,316	2.31	34.50

表 8 1BL1 分子の構造最適化の実行時間
Table 8 The elapsed time of searching trial structure of 1BL1.

Cluster (# of workers)	Total # of workers	Structures /worker	Optimization time /structure (s)	Elapsed time (s)	Elapsed time (H)	Speed up
Toyo MPI (16)	16	32.4	3,646	120,028	33.34	15.76
Dennis (28)	28	18.5	3,154	61,803	17.16	30.61
Dennis (28) + Ume (56)	84	6.1	4,497	33,502	9.30	56.48

れ**表 7**，**表 8** に示す．この表の Speedup は MPI 版 CONFLEX にて Toyo クラスタの 16 ワーカを使った実行結果より算出した，Toyo クラスタの 1 ワーカの性能を 1 として算出した．

CONFLEX-G は，Dennis と Ume クラスタで 84 ワーカを使用したときに，Toyo クラスタの MPI 版の 1 CPU で計算した場合と比較して，1L2Y のときは，最大で 34.5 倍，1BL1 のときには最大で 56.5 倍の高速化が得られている．この実験では，1 構造最適化にかかる処理時間が長時間にわたるため，CONFLEX-G は，ワーカプログラムの起動や初期化のデータ転送などのオーバヘッドの割合が限りなく小さくなり，このような高速化が得られていると考えられる．

しかし，AlaX16 分子の実験と同じように，1 試行構造の最適化の処理時間のばらつきによる，負荷の不均衡が，CONFLEX-G の性能向上を低くする原因となっている．結果が良かった Dennis と Ume クラスタを利用した場合にも，CONFLEX-G において，1L2Y 分子のときは 1 構造最適化にかかる時間は最短で 190 秒，最長で 2559.27 秒となっており，差は 13.4 倍であった．さらに，1BL1 分子のときは，1 構造最適化時間が最短で 146 秒，最長で 27887 秒と 190 倍も差があった．また，1LB1 分子においては，試行構造最適化が終了しアイドル状態のノードが，最後の試行構造最適化の終了を待つために約 6 時間待ち合わせしていた．これが，CONFLEX-G の性能低下を引き起こしている原因だった．

4.4 考　　察

OmniRPC では，RPC 呼び出しの必要に応じて，呼び出された時点でリモート実行プログラムを起動する．CONFLEX-G では，試行構造の構造最適化がワーカプログラムに初めて割り当てられたときに，ランタイムにプログラムを起動し，パラメータファイルをネットワーク転送し，そのデータを処理して初期化を行っている．しかし，MPI 版 CONFLEX では構造最適化を行う前段階で，すべてのワーカプログラムはローカルなファイルシステムからエネルギー評価のためのパラメータファルを読み込み，そのデータを処理し初期化している．このため構造最適化を行う前にプログラムの初期化が終わっている MPI 版 CONFLEX とは違い，CONFLEX-G は，その初期化の部分が構造最適化のところで実行されているため，MPI 版 CONFLEX と比べて初期化のためのオーバヘッドが増えてしまい，性能が少し低くなっている．

表 9 Dennis クラスタの 28 ワーカを使用したときの試行構造最適化の処理時間の統計
Table 9 The statistics of elapsed time of trial structure optimization using 28 workers in Dennis cluster.

Molecular code	Min (s)	Max (s)	Average (s)	Standard variance
AlaX04	2.0	11.3	5.3	1.7
AlaX16	47.6	920.0	154.2	73.5
1L2Y	114.2	13331.4	803.2	797.9
1BL1	121.0	29641.8	3153.5	1653.7

CONFLEX-G において，いったんリモート実行プログラムが起動されると初期化は必要なくなるため，長時間実行される場合は，その影響が少なくなるが，1 試行構造最適化が比較的短時間で終わる場合に，この影響は無視できない．この問題における解決方法としては，OmniRPC に，あらかじめいくつかのクライアントプログラムを RPC 呼び出しの前に起動させる API を用意するということが考えられる．

表 9 に，Dennis クラスタの 28 ワーカを使い，それぞれのサンプル分子を CONFLEX-G で処理したときの 1 試行構造最適化の処理時間の統計を示す．この表が示すように，1 つの試行構造の最適化にかかる時間のばらつきがあり，1 つの遅い構造最適化の処理がボト

ルネックとなり，負荷の不均衡により性能を低下させている．本実験で，Dennis クラスタで CONFLEX-G を使用し，1BL1 分子の配座探索を行った際に，最長の 1 試行構造最適化の実行時間が全実行の約 80%を占めており，最後の構造最適化が終了するのを他のワーカが，全時間の 86%である約 8 時間アイドル状態のまま待機していた．現在の CONFLEX の実装では，試行構造最適化の処理時間の最長のものと最短のものとの差が最悪 1000 倍ほどになる可能性がある．そのため，試行構造生成において，構造最適化にかかる時間のばらつきをおさえるような構造を生成する方法を考案することがあげられる．これが可能になると，CONFLEX-G はさらに高い台数効果が得られるようになると考えられる．また他のアプローチとして，処理時間が長い構造最適化をより高速な計算機に，処理時間が短いものをより遅い計算機に割り当てるということも考えられる．しかし，試行構造にかかる時間を予測できるのかということは現在分かっていない．

現在の実装では，リモート実行プログラムでは並列化を行っていないが SMP（Symmetric Multiple Processor）で OpenMP などを使用し，OmniRPC との Hybrid プログラミングを行い，1 つの構造最適化の処理を高速化させることも考えられる．

5. おわりに

本論文では，分子配座空間探索プログラム CONFLEX のグリッド環境への実装と性能評価を行った．クラスタ環境において MPI 版 CONFLEX と比較したところ，CONFLEX-G は MPI 版と同じように台数効果が得られていることを確認した．また，広域ネットワーク環境で複数のクラスタを使用し性能評価を行い，分子の原子数が大きく，1 つの試行構造最適化の処理時間が長い場合において性能向上を確認した．これにより，これまで長時間かかっていた分子配座探索がグリッド環境を利用することによって，今回性能評価で用いた 1LB1 分子の場合で，最大 55 倍高速に処理できるようになった．1 つの試行構造最適化にかかる処理時間が長時間で，その処理時間が均一であり，かつ一度に生成される試行構造の数が多い場合において，CONFLEX-G はスケーラビリティが高いと考えられる．

現在，リモートホストにおいて実行プログラムの登録などの設定を手動で行っているが，ホスト数が多くなるに従って，適当な支援ツールが必要である．また，耐故障性の機能についても必要ではないかと考えられる．実験中に計算ノードが動作しなくなってしまう場合が多々あり，それによって計算を正常に終了させることができず，再度最初から計算を行ったことがあった．大規模な分子を計算する場合には，実行時間は長時間になると考えられるため，複数の計算ノードが故障してしまう確率は高くなる．この故障により計算が中止されないよう，大規模な分子を相手にする場合には耐故障性の機能は必須ではないかと考えられる．

今後，CONFLEX の配座最適化のアルゴリズムを見直し，HIV プロテアーゼのような生体高分子についての配座探索を行っていく予定である．

謝辞 実験環境を提供していただいた，産業総合技術研究所グリッド研究センターに感謝いたします．日頃，研究協力いただいている JST-ACT グリッド創薬プラットフォームプロジェクトの皆様に感謝します．本研究の一部は，科学研究費補助金特定領域研究（2）課題番号 14019011「計算物理学分野の Grid アプリケーションと並列プログラミングシステムの研究」，文部科学省産官学連携イノベーション創出事業費補助金「コンホメーション・シーケンサー・システムに関する研究」，および JST-ACT「GRID テクノロジを用いた創薬プラットフォームの構築」による．

参考文献

1) Arnold, D., Agrawal, S., Blackford, S., Dongarra, J., Miller, M., Seymour, K., Sagi, K., Shi, Z. and Vadhiyar, S.: Users' Guide to NetSolve V1.4.1, Innovative Computing Dept. Technical Report ICL-UT-02-05, University of Tennessee, Knoxville, TN (2002).
2) Foster, I. and Kesselman, C.: Globus: A Metacomputing Infrastructure Toolkit, *The International Journal of Supercomputer Applications and High Performance Computing*, Vol.11, No.2, pp.115–128 (1997).
3) Global Grid Forum. http://www.ggf.org/
4) Grid Remote Procedure Call Working Group. http://graal.ens-lyon.fr/GridRPC/
5) Goto, H. and Osawa, E.: An Efficient Algorithm for Searching Low-energy Conformers of Cyclic and Acyclic Molecules, *J. Chem. Soc., Perkin Trans 2*, pp.187–198 (1993).
6) Goto, H., Takahashi, T., Takata, Y., Ohta, K. and Nagashima, U.: CONFLEX: Conformational Behaviors of Polypeptides as Predicted by a Conformational Space Search, *Nanotech 2003*, Vol.1, pp.32–35 (2003).
7) Karonis, N.T., Toonen, B.R. and Foster, I.T.: 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).
8) Seymour, K., Nakada, H., Matsuoka, S., Dongarra, J., Lee, C. and Casanova, H.: GridRPC: A Remote Procedure Call API for Grid Computing.
9) Larson, S.M., Snow, C.D., Shirts, M. and Pande, V.S.: Folding@Home and Genome@Home: Using distributed computing to tackle previously intractable problems in computational biology, *Computational Genomics* (2002).
10) Sato, M., Boku, T. and Takahashi, D.: OmniRPC: A Grid RPC System for Parallel Programming in Cluster and Grid Environment, *Proc. CCGrid 2003*, pp.219–229 (2003).
11) Sato, M., Nakada, H., Sekiguchi, S., Matsuoka, S., Nagashima, U. and Takagi, H.: Ninf: A Network Based Information Library for Global World-Wide Computing Infrastructure, *Proc. HPCN Europe*, pp.491–502 (1997).
12) Ninf Project. http://ninf.apgrid.org/
13) OmniRPC Project. http://www.omni.hpcc.jp/OmniRPC/
14) seti@home project. http://setiathome.ssl.berkeley.edu/
15) 佐藤三久，朴　泰祐，高橋大介：OmniRPC：グリッド環境での並列プログラミングのための Grid RPC システム，情報処理学会論文誌：コンピューティングシステム，Vol.44, No.SIG11(ACS 3), pp.34–45 (2003).
16) 中田秀基，松岡　聡，関口智嗣：Java による階層型グリッド環境 Jojo の設計と実装，先進的計算機基盤システムシンポジウム SACSIS2003 論文集，pp.113–120 (2003).

(平成 15 年 10 月 10 日受付)
(平成 16 年 1 月 22 日採録)

中島　佳宏（学生会員）

昭和 55 年生．平成 15 年筑波大学第三学群情報学類卒業．現在，同大学大学院システム情報工学研究科在学中．グリッドコンピューティング等に関する研究に従事．

佐藤　三久（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 61 年同大学大学院理学系研究科博士課程中退．同年新技術事業団後藤磁束量子情報プロジェクトに参加．平成 3 年通産省電子技術総合研究所入所．平成 8 年新情報処理開発機構並列分散システムパフォーマンス研究室室長．平成 13 年より筑波大学電子・情報工学系教授．同大学計算物理学研究センター勤務．理学博士．並列処理アーキテクチャ，言語およびコンパイラ，計算機性能評価技術，グリッドコンピューティング等の研究に従事．日本応用数理学会，IEEE 各会員．

後藤　仁志

昭和 39 年生．昭和 63 年北海道大学理学部化学第二学科卒業．平成 5 年同大学大学院理学研究科化学専攻博士課程修了．博士（理学）．平成 8 年東北大学反応科学研究所助手，講師を経て，平成 10 年豊橋技術科学大学大学院工学研究科助教授，現在に至る．コンホメーション探索法や遷移状態探索法等の化学計算アルゴリズム，有機化合物のキラリティーデータベース等に関する研究に従事．平成 12 年ミレニアムプロジェクトにより CONFLEX をベースにしたコンホメーション・シーケンサー・システムの開発に取り組む．日本化学会，米国化学会，CBI 学会，日本コンピュータ化学会等各会員．

朴　　泰祐（正会員）

昭和 59 年慶應義塾大学工学部電気工学科卒業．平成 2 年同大学大学院理工学研究科電気工学専攻後期博士課程修了．工学博士．昭和 63 年慶應義塾大学理工学部物理学科助手．平成 4 年筑波大学電子・情報工学系講師，平成 7 年同助教授，現時に至る．超並列処理ネットワーク，超並列計算機アーキテクチャ，ハイパフォーマンスコンピューティング，並列処理システム性能評価の研究に従事．平成 14 年度情報処理学会論文賞受賞．電子情報通信学会，日本応用数理学会，IEEE 各会員．

高橋　大介（正会員）

昭和 45 年生．平成 3 年呉工業高等専門学校電気工学科卒業．平成 5 年豊橋技術科学大学工学部情報工学課程卒業．平成 7 年同大学大学院工学研究科情報工学専攻修士課程修了．平成 9 年東京大学大学院理学系研究科情報科学専攻博士課程中退．同年同大学大型計算機センター助手．平成 12 年埼玉大学大学院理工学研究科助手．平成 13 年筑波大学電子・情報工学系講師．博士（理学）．並列数値計算アルゴリズムに関する研究に従事．平成 10 年度情報処理学会山下記念研究賞，平成 10 年度情報処理学会論文賞各受賞．日本応用数理学会，ACM，IEEE，SIAM 各会員．