2003－HPC－95　(13)
2003／8／4

OmniRPCによる広域ネットワーク環境でのグリッドアプリケーションの性能評価

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

OmniRPC は，グリッド環境での並列プログラミングための Grid RPC システムである．地理的に広く離れたところにあるクラスタを結ぶグリッド環境を構築し，OmniRPC を使用したグリッドアプリケーションの性能評価を行った．リモートで実行されるジョブの実行時間と通信量を変化させるプログラムにより，ネットワークと性能向上の関係について明らかにした．さらに，実用的なアプリケーションとして，分子の立体配座探索を行なうプログラムである CONFLEX のグリッド化を行なった．CONFLEX のグリッド化について詳細を述べるとともに，広域ネットワーク環境において実験を行ない，分子の立体配座探索を高速に並列処理できることを確認した．OmniRPC にグリッドでの実行を支援するモニタリングの機能を実装した．

Perfomance Evaluation of Grid Applications by OmniRPC in Wide Area Network

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

OmniRPC is a Grid RPC system for parallel programming in a Grid environment. We have evaluated the performance of Grid applications by OmniRPC in a grid which consists of several clusters geographically distributed. We examined the performance by OmniRPC on several configuration of our gird by the synthetic program which varies the execution time in remote nodes and the amount of communication. As a practical application, we parallelized CONFLEX molecular confirmation search program by OmniRPC, and obtained some speedup in our grid environment. To support the execution in a grid, we developed a monitoring tool for OmniRPC.

1. はじめに

インターネットをはじめとし，広域ネットワークの進歩により，広域ネットワーク上の計算機資源やデータの共有，並列分散コンピューティングの支援を可能にするグリッド技術が注目されている．

我々は，グリッド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 GridRPC である OmniRPC[1),2)] を開発している．本稿では，広域ネットワークにおいて OmniRPC を使用したグリッドアプリケーションの性能評価について述べる．

現在，グリッドの標準的な環境として Globus Toolkit[3)] が多く用いられている．Globus では，遠隔の計算機環境でのジョブの起動コマンドインターフェースを提供している．これにより，簡単なジョブスクリプトを記述することができる．グリッド向けのプログラミング環境としては，Globus の MPI プログラミング環境 MPICH-G2 がある．しかし，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ためには MPI ではプログラミングの手間が多すぎる．

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り，計算機資源の状況が変化するグリッド環境では利用するのが難しい場合がある．

開発する並列プログラミングシステムにおいて，簡単に，背後にあるグリッド上の計算機資源の割り当てを意識することなく，計算が行えることが望ましい．そのための遠隔計算機へのインターフェースとして遠隔手続き呼び出し (RPC: Remote Proceduar Call) を用いて，遠隔計算機資源でのプログラムの実行を可能にする機構が提案されている．

OmniRPC もその一つであり，このほかにもグリッド環境で RPC 呼び出しを用いて計算機を利用する，

† 筑波大学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

GridRPC はプログラミングインタフェースとしていくつか提案されている．広域ネットワーク向けの RPC として提案された Ninf[4),5)] を Globus 向けに実装した Ninf-G はグリッド向け RPC の一つである．OmniRPC も Ninf システムから派生し，基本的な API として Ninf システムを踏襲している．ほかに，NetSolve[6)], Jojo[7)] が挙げられる．

我々は，OmniPRC を用いた広域ネットワークによるグリッド環境として，産業総合研究所，徳島大，豊橋技科大，筑波大のクラスタを接続する，グリッド環境を構築した．まず RPC の計算をモデル化するプログラム，NAS Parallel Benchmark の EP を使い OmniRPC の性能評価を行なった．さらに，分子立体構造探索を行なう CONFLEX プログラムをグリッド化し，広域ネットワーク環境において性能評価を行なった．

次章において OmniRPC の概要を述べ，3 章では，構築したグリッド環境ついて報告し，広域ネットワーク環境での OmniRPC の性能評価を述べる．4 章では，グリッド化を行なった CONFLEX の概要を述べ，グリッド化の方法，性能評価を述べる．5 章で今後の課題を挙げる．

2. 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．

プログラミングモデルとして，master-slave 型の並列プログラミングをサポートしている．特に，グリッド環境において典型的な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機能を備えている．

また，利用する計算資源として，単一の PC やワークステーションはもちろんのこと，クラスタを利用することができる．OmniRPC がターゲットとするグリッド環境としては，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．クラスタを構成するネットワークとしてプライベートな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．

API として，基本的に Ninf の API を踏襲しており，リモート側の状態を保持する persistency をサポートしているため，これを利用した効率的なプログラミングを可能としている．並列プログラミングのための API としては，非同期呼び出しを用いることができる．さらに，簡便な並列プログラミングのために RPC をスレッドセーフに実装されているため，OpenMP の指示文による並列プログラミングをサポートし，ベースとなる既存の逐次プログラムをなるべく書き換えずに並列化することができる．

パラメータ検索など並列アプリケーションを効率的にサポートするために，自動初期化実行モジュール機能を提供している．これは，初期化のための大量のデータの転送や計算が必要な場合，これを再利用することにより，効率化を図ることができ，また，余分なデータ転送を防ぐことができる．

認証を行う Grid 環境として，Globus のほかに，SSH による認証も可能となっている．ファイヤウォールのある遠隔の計算機についても，エージェントによる 通信の多重化をおこなう proxy 機能を用いることにより，SSH で login できるシステムならば，計算資源として利用できる．

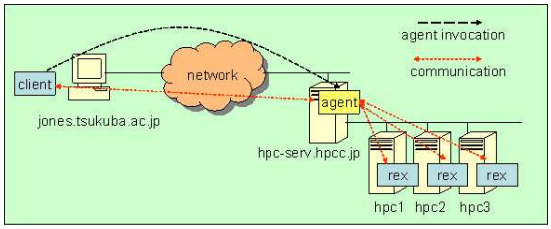

図 1 OmniRPC の動作

OmniRPC では，まず実行開始時に omrpc-agent と呼ばれるプロセスをリモートホストに起動する．このプログラムが実際のリモート実行プログラムを起動する．OmniRPC の動作イメージを図 1 に示す．これは，通信の多重化を行ない，クラスタに RPC 呼び出しを行なっている場合の図である．ここで，omrpc-agent はクライアントプログラムが起動されたときにホストファイルで指定されたホスト上において起動される．クライアントプログラムの RPC 呼び出しに対して，計算ノードにあるスタブプログラムを起動する．さらに omrpc-agent は，proxy 機能を有し通信の多重化を行なうことができる..

ホストファイルで 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．これにより，プログラムの変更なしに，ホストファイルを書き換えることによって動作させる環境を変えることができる．ホストファイルの例を図 2 に示す．このホストファイルの設定では，Globus を使って 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 4 である．

```
<?xml version="1.0" ?>
<OmniRpcConfig>
   <Host name="hpc-serv.hpcc.jp" >
   <Agent invoker="globus" mxio="on" />
   <JobScheduler type="rr" maxjob="4" />
   </Host>
</OmniRpcConfig>
```

図 2 hostfile の例

このホストファイルを適当に設定することにより，並列プログラムの開発をそれぞれのクラスタで行い，グリッド環境でプログラムの変更なしに実行することができる．

2.1 モニタリング

本来グリッド環境において，ジョブを Submit した

ときに，どのホストで計算が行われているのかということを知ることはできない．しかし，どのホストで計算が行われているのかを検知することができないと，正常に動作をしているのかわからずだけでなく，デバッグも行うことができない．そのため，モニタリングを行うプログラムを実装した．

モニタリングの表示部分のスクリーンショットを図 3 に示す．モニタリングは 3 つのプログラムから構成されており，表示部分のプログラム，負荷情報を送るデーモンとデーモンを起動する omrpc-agent である．

このモニタリングの動作は，OmniRPC と同じように動作する．まず，表示プログラムが起動したときに OmniRPC が使用する設定ファイル読み込み，そこで指定されているホストにおいて，omrpc-agent を起動させ，遠隔手続き呼び出しを行うリモートホスト上に，モニタリングを行なうデーモンを立ち上げる．このデーモンは，負荷状況をクライアントホストに転送し，表示プログラムが，その情報をグラフ表示する．この機能によって，リモートホストに遠隔手続き呼び出しをしたが，何も計算されず無い場合などを検知することができる．

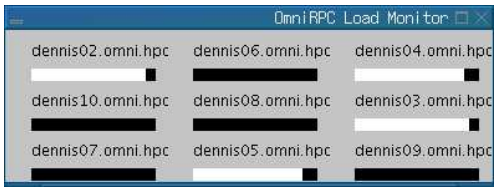

図 3 プロトタイプのスクリーンショット

3. OmniRPC の性能評価

OmniRPC を使用した広域ネットワークの実験においてグリッドアプリケーションの性能評価を行なうにあたり，予備実験として，一回の遠隔手続きの処理時間，遠隔手続き呼び出しの通信量，実行量のパラメータが性能にどのように影響を与えるのか明らかにする．次に NAS Parallel Benchmark 2.3 の EP の性能測定を行なう．

3.1 実 験 環 境

実験に関して，筑波大学，豊橋技術科学，徳島大学，産業技術総合研究所の 5 つのサイトにあるクラスターを使用して行った．実験環境を表 1 に示す．また，netprof， ping を使って Dennis クラスタから，各サイトにあるクラスタのマスターノード間の通信性能測定を行った，その結果を表 2 に示す．

この結果より，実験環境をクラスタ間のネットワークの特性から 3 つの場合に分け，実験を行なった．

- 高速なネットワーク接続の場合
  Dennis と Alice.
- ネットワークの性能が中程度の場合.
  Dennis と UME.
- ネットワークの遅延が大きく，バンド幅が低い場合
  Dennis と Toku，Dennis と Toyo

遠隔手続き呼び出しのホストへの割り当ては，OmniRPC の default のスケジューリングである，round-robin を使用した．

表 2 Dennis Cluster と各クラスタ間のネットワーク性能

Cluster	Round-Trip Time (ms)	Throughput (MB/s)
Alice	0.18	11.22
Toyo	13.0	0.55
Toku	24.4	0.69
UME	2.73	2.12

3.2 モデルプログラム

広域ネットワークにおいて OmniRPC の性能がどのように変化するのかを，遠隔手続き呼び出しのパラメータを変えて実験した．変化させたパラメータは，遠隔手続き呼び出しの回数，それに伴うネットワーク転送量である．

この実験では，OmniRPC を用いてプログラムを作成したときに，どのくらいの転送量とどのくらいのジョブ実行時間があればどのくらいの台数効果を得られるかを調べる．

プログラムは，クライアントプログラムから引数として，指定された長さの文字列をスタブプログラムに転送し，一定時間スタブプログラムを休止し，引数として渡された文字列をクライアントプログラムに送り返すプログラムである．並列に実行をモデル化するため，RPC 呼び出しは非同期に行なわれる．

実験は，クラスタ間のネットワークの性能が高い Dennis と Alice を使った場合，クラスタ間のネットワークの性能が低い Dennis と UME を使用した場合の 2 つに分けて実験を行なった．ここで，同時に実行される実行プログラムを 40．使用する計算機は 1 クラスタであたり最大 20 とする．また，短いジョブ実行時間，長いジョブ時間として，リモートホストあたりでの計算時間を，それぞれ 2 秒の場合と 16 秒の場合とした．さらに，クライアントソフトは Dennis クラスタのマスターノードから動作させ，認証方式として SSH を使用し，通信の多重化は行なっていない．

ジョブの実行時間が 2 秒のときの結果を図 4，16 秒のときの結果を図 5 に示す．

ジョブの実行時間が 2 秒でも，ネットワークの性能が高い場合，性能が得られることがある．

16 秒の場合ではほとんどの場合において台数効果を期待することができるが，ネットワークの性能が低い場合において RPC 呼び出しで 1MB の転送を行なうと性能が得られない．また，RPC 呼び出しの回数が少ない場合には，転送などのオーバヘッドを隠すだけの性能向上が得られない．

また，リモートで計算される時間が長いほど，台数効果が高くなることが分かる．

3.3 NAS Parallel Benchmark EP

NAS Parallel Benchmark の EP を OmniRPC を用いて実装し，1 台のクラスタで起動させるワーカーを最大数を 8 として実験を行なった．なお，2 台のクラスタを使用する時は 16 になる．クラス B の実行時

表 1 広域ネットワークにおいての実験環境

Site	Cluster Name	Machine	Agent-Invoker	Node 数	CPU 数
筑波大学	Dennis	Dual Pentium4 Xeon 2.4GHz	Globus,SSH	10	20
	Alice	Dual Athlon 1800+	Globus,SSH	15	30
豊橋技術科学大学	Toyo	Dual Athlon 1800+	SSH	8	16
徳島大学	Toku	Pentium3 1.0GHz	Globus, SSH	8	8
産業技術総合研究所	UME	Dual Pentium3 1.4GHz	Globus, SSH	32	64

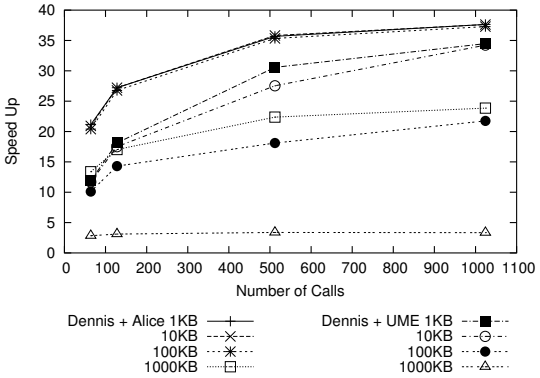

図 4 ジョブの実行時間が 2 秒のときの台数効果

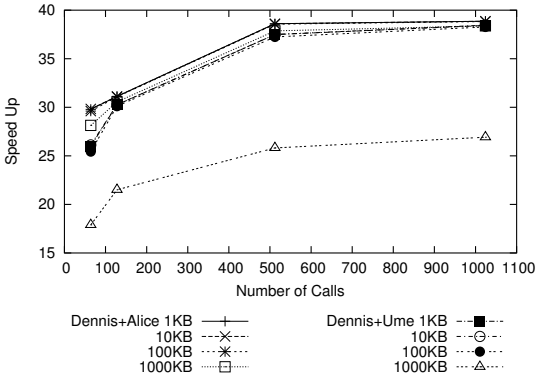

図 5 ジョブの実行時間が 16 秒のときの台数効果

間と各個の中に Dennis の 1 ノードで計算する場合と比較した場合の台数効果を表 3 に示す.

表 3 NAS Parallel Benchmark EP Class B の実行時間と台数効果

	SSH		Globus
クラスタ	MXIO On	MXIO Off	MXIO Off
Dennis	102.43 (7.8)	101.32 (8.0)	102.04 (7.9)
UME	117.73 (6.8)	112.92 (7.1)	113.96 (7.1)
Toku	633.90 (1.2)	204.88 (3.9)	204.93 (3.9)
Dennis+Toku	101.69 (7.9)	71.17 (11.3)	71.90 (11.2)
Dennis+UME	92.60 (8.7)	56.74 (14.2)	57.14 (14.1)
UME+Toku	404.29 (2.0)	80.25 (10.0)	79.57 (10.1)

実験結果より，認証方式として，SSH， Globus を利用しても性能にあまり影響しないことが分かる．

また，2台のクラスタをつないで処理を行なった際に，クラスタ間のネットワークの性能が悪い場合においても，効率的に資源にジョブが配分され，1 つのクラスタにおいて計算させたときよりも性能が向上している．

また，ネットワークの遅延が大きくない場合において，通信の多重化の機能を使用しても性能はあまり変わらない．しかし Firewall が至るところに存在する広域ネットワーク環境においては，通信の多重化は必要となる．ネットワークの遅延が小さい場合において，通信の多重化を行なっても性能はほぼ同じであった．よって，ネットワークの遅延が少ない場合には，通信の多重化は問題なく使用できると考えられる．しかし，SSH で通信の多重化を利用し port forwaring を利用した場合，ネットワークの遅延が大きい環境では大きく性能が低下するのが見られる．

4. CONFLEX のグリッド並列化

配座探索プログラム CONFLEX[8),9)] は，対象とする化合物が取り得るすべての立体配座を自動的に発生させ，化学的に重要な配座異性体の最適化構造をもれなくみつけ出すプログラムである．

高分子を扱う場合，構造最適化の処理に莫大な時間がかかる．その処理は，分子構造のパラメータを受取り，Molcular Mechanics を使い構造最適化させることを繰り返す．一回の構造最適化を行なう計算時間は長く，またこの処理部分は，Master/Worker パラダイムでプログラミングすることができる．よって，構造最適化の処理の部分を並列化することにより，高速化を図ることができる．

そこで，巨大な計算資源を利用できるグリッド環境において，恩恵が得られると考えられるため，CONFLEX のグリッド並列化をおこなった．

4.1 CONFLEX の概要

CONFLEX の配座探索アルゴリズムは，以下のようになる．

(1) 既に保存されている構造の中からの初期構造の選択および適当な構造の生成
(2) 構造最適化
(3) 既に得られている構造と比較し，新しい配座を保存

以上の手順を，終了条件を満たすまで繰り返す．

CONFLEX での最大の特徴は構造の生成であり，中でも，環状部分に採用している，Corner Flap および Edge Flip により，優れた探索を行なうことができる．

以下の方法で構造の生成をおこなっている．

- Corner Flap
  初期構造の環の構成原子から一つを選び，環の平均平面に対して反対側に移動させることによって

新たな出発構造を生成する．

- Edge Flip
  初期構造の環の構成原子から隣接する二つを選び，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．さらに二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．

以上の三つの操作は，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から，エネルギーの低い物から順に初期構造を選択する．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，次にその配座が初期構造とし構造最適化を行なうようにする．これによって，探索領域が速やかにエネルギーの低い方へと向かうようになる．ここで，最安定配座が見つかれば，探索領域を徐々に高い方へげるようにする．これを，池に水が流れ込み周りに水が満たされていく様子ににているので,「Reservoir-Filling（貯水池注水）アルゴリズム」と呼んでいる．

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，全元素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既に見つかっている構造，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る．

4.2 CONFLEX-G: CONFLEX のグリッド化

CONFLEX は生体高分子をターゲットとしているため，配座探索効率の向上が必要である．試行構造の構造最適化の処理が全探索時間の 90%以上を占めており，試行構造の構造最適化の処理を並列化して行うことにより，高速化が図ることができる．

そこで，試行構造を Master ノードにプールしてから，空いている slave ノードに一つづつ試行構造を渡し，構造最適化の処理を行う Master/Slave 型の並列化が可能である．

試行構造の構造最適化の処理の部分を OmniRPC を使ってリモートホストで構造最適化をさせるようにCONFLEX のプログラムの変更を行った．図 6 に，計算の概要について示す．OmniRPC は自動初期化実行モジュール機能を提供しており，初めてスタブプログラムが起動されたときにリモート実行プログラムに設定ファイルを送り，初期化を行なう．それ以降の RPC 呼び出しに対しては，リモートホスト上で構造最適化の度に初期化された状態を再利用されるようになる．RPC 呼び出しごとに初期化することを省くことができ，計算を効率化できる．

クライアントホストでは，それぞれリモートホストに試行構造をを送り，リモートホストにて，試行構造の最適化を行い，その結果をクライアントホストに返すようにする．こ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，この部分を非同期遠隔手続き呼び出しを行うことにより並列に処理を行うようにした．

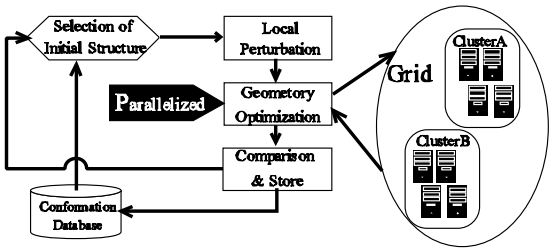

図 6 CONFLEX-G の処理

4.3 CONFLEX-G の性能評価

使用した CONFLEX のバージョンは 402H，サンプルとして 51 原子の cycloheptadecane を使用した．

Dennis をクライアントとして，広域ネットワークを介して Toku，UME の各クラスタで実行した結果を結果は 4 に示す．ここで使用したジョブ数は各クラスターの CPU 数に設定している．また，認証として SSH とし，通信は多重化を行なっていない．さらに台数効果は Dennis の 1 ノードで計算したときとの比である．

比較のために，MPI を使った CONFLEX のプログラムを Dennis クラスタで，ローカルに実行した結果も示す．なお，Dennis クラスタで，ローカルに OmniRPC で実行してみたところ，NFS サーバに問題があり良好な結果が得られなかった．これについては，解決され次第報告することにする．

表 4 広域ネットワーク環境での CONFLEX の単位時間あたりの構造最適化数

Cluster	Max Workers	Structures per second	Speed up
Toku	8	0.50	2.63
UME	64	1.47	8.039
Tok + UME	72	0.89	4.68
MPI (Dennis)	20	0.91	4.79
seq (Dennis)	1	0.19	1.0

この実験において，広域ネットワーク環境で CONFLEX の OmniRPC 版を動かした場合に，1 構造あたり性能向上は 5 倍程度にどどまった．その理由の一つは，今回評価に用いた問題のサイズにあると考えられる．今回用いた分子では 1 回の探索について，50 程度の構造最適化を行う．また，1 つの最適化にかかる時間にバラツキがあり，負荷が不均衡になっている．このため，2 つのクラスタによりジョブ数を増やしても効果が得られない．また，ネットワークの遅延によっても，性能が下がっているのではないかと考えられる．

問題規模を大きくし，探索をする分子の原子数を多くすることにより，構造最適化を行なう試行構造のパターンも増えてくるため，構造最適化の部分の並列度が高くなり，高い台数効果が期待できる．モジュールの初期化の部分も，高分子のものに対しては，構造最適化の処理が増えるため，全体の処理に占める割合が減るので，実際に高分子の配座構造探索を行なう際には処理の高速化を図れると考えられる．

4.4 問 題 点

以下に，今回の実験で明らかになった問題点について述べる．

OmniRPC では，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．このため構造最適化を行なう前にプログラムの初期化が終わっている MPI 版とは違い，OmniRPC 版は，その初期化の部分が構造最適化の所で実行されており OmniRPC 版の性能を低下させている．一旦，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，その影響は少なくなるが，今回のように比較的短時間で終る場合には影響は無視できない．この解決方法としては，あらかじめいくつかのスタブプログラムを RPC 呼び出しを行なう前に起動させる API を用意することで解決できると考えられる．

また，現在の実装ではリモート実行プログラムの初期化が逐次的に行われており，１つのリモート実行プログラムの初期化が行われている場合は他のリモート実行プログラムの初期化ができない．このため，全てのリモート実行プログラムが初期化されるまでは並列化の効果が十分に得られなくなってしまう．これについては，これからリモート実行プログラムの初期化も並列に行うことができるように実装を改善していく予定である．

5. おわりに

本稿では，地理的に離れたところにあるクラスタ結ぶグリッド環境を構築し，広域ネットワーク環境において OmniRPC の基本的な性能評価を行った．これにより，リモートで実行されるジョブに数秒に実行時間があり，数百 K バイト程度の通信を必要とするジョブでは性能向上が見込めることがわかった．さらに，実用的なアプリケーションとして配座探索プログラムである CONFLEX を OmniRPC を用いてグリッド化を行ない，広域ネットワーク環境にて性能評価を行ない，性能向上を確認した．また，実行を支援するツールとしてモニターリングするプログラムを開発した．

現在，OmniRPC では，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したがって，適当な支援ツールが必要である．グリッド環境下においては，環境が動的に変化し，ノードが故障した場合などに対応しなくてはならない．広域ネットワーク環境下において，とくに，大規模なプログラムを使用する際には，長時間計算が行われると考えられるため，動的なリモートホストの追加・削除などの設定変更を行える機構はは必要になると考えられる．

また，今回で見つかった問題点であるリモート実行プログラムの問題点については改善し，大規模な問題についての CONFLEX の性能評価，性能の改善を行なう予定である．

謝辞 実験環境を提供していただいた，産業技術研究所のグリッドリサーチセンターに感謝致します．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」，および JST-ACT 「創薬プラットホームのためのグリッド環境の開発」による．

参 考 文 献

1) M. Sato, T. Boku and D. Takahashi: OmniRPC: a Grid RPC System for Parallel Programming in Cluster and Grid Environment, *Proc. of CCGrid2003*, pp. 219–229 (2003).
2) 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, 先進的計算機基盤システムシンポジウム SACSIS2003 論文集, pp. 105–112 (2003).
3) I. Foster and C. Kesselman: Globus: A meta computing infrastructure toolkit, *Workshop on Environments and Tools* (1996). http://www.globus.org/.
4) Mitsuhisa Sato, Hidemoto Nakada, Satoshi Sekiguchi, Satoshi Matsuoka, Umpei Nagashima and Hiromitsu Takagi: Ninf: A Network Based Information Library for Global World-Wide Computing Infrastructure, *HPCN Europe*, pp. 491–502 (1997).
5) Ninf Project: . http://ninf.apgrid.org/.
6) 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).
7) 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).
8) H.Goto, T. Takahashi, Y. Takata, K. Ohta and U Nagashima: CONFLEX: Conformational Behaviors of Polypeptides as Predicted by a Conformational Space Search, *Nanotech2003*, pp. 32–35 (2003).
9) H. Goto and E. Osawa: An Efficient Algorithm for Searching Low-energy Conformers of Cyclic and Acyclic Molecules, *J. Chem. Soc., Perkin Trans*, Vol. 2, pp. 187–198 (1993).