2004－HPC－99　(24)
2004／7／31

グリッド上での並列分枝限定法アプリケーションの評価

合田 憲人†‡　　大角 知孝†　　中村 心至§

本稿では，グリッド上で並列分枝限定法アプリケーションを効率よく実行するための手法とその性能評価結果について報告する．本アプリケーションでは，グリッド上で細粒度タスクを効率よく実行するために，階層的マスタ・ワーカ方式により並列化され，複数の WAN 上の PC クラスタ群から構成されるグリッド上に GridRPC ミドルウェアである Ninf-G および Ninf を用いて実装される．また PC クラスタ内では，通信オーバーヘッド軽減と効率よい暫定値通知を考慮したタスク粒度の自動的調整が行われ，プログラムの高速化が実現される．本アプリケーションをグリッド実験環境上に実装し，性能評価を行った結果，アプリケーションの有効性が確認された．

Evaluation of the Parallel Branch and Bound Application on the Grid

KENTO AIDA†‡,　TOMOTAKA OSUMI†　and　MOTOYUKI NAKAMURA§

This paper presents a case study to effectively run the parallel branch and bound application on the Grid. The application discussed in this paper is a fine-grain application and is parallelized with the hierarchical master-worker paradigm. The application is implemented on a Grid testbed, which is constructed by multiple PC cluster connected to WAN, by using GridRPC middleware, Ninf-G and Ninf. Also, the task granularity is adaptively controlled to reduce communication overhead and to efficiently advertise the best upper bound on each PC cluster. Experimental results showed effectiveness of the application.

1．はじめに

グリッド計算は，地理的に分散した計算資源を利用することにより，低コストで高性能計算を実現する新しい計算手法として注目されており，従来の大規模計算に要していた時間を短縮するだけでなく，高性能計算の応用範囲を広げる効果も期待されている．しかしグリッド計算環境では，WAN 上の通信オーバーヘッドやグリッド上のセキュリティ処理によるオーバーヘッドが大きく，効率よく実行可能なアプリケーションは，これらオーバーヘッドに対して十分な計算時間（例えば数十秒～数百秒）を持つタスクから構成されるものに限られている[1][2][3][4]．従って，個々のタスク計算時間が短い細粒度アプリケーションをグリッド上で効率よく実行させる手法を開発することは，グリッド計算の応用範囲を広げる意味で重要である．

本稿では，グリッド上で並列分枝限定法アプリケーションを効率よく実行するための手法とその性能評価結果について報告する．分枝限定法は最適化問題解法の一つであり，オペレーションズリサーチ，制御工学，マルチプロセッサスケジューリング等の多くの工学分野で用いられている[5][6][7]．これらのアプリケーションの多くは，細粒度アプリケーションであり，実行時間の短いタスクを大量に処理する必要がある．本稿が述べる並列分枝限定法アプリケーションは，細粒度タスクをグリッド上で効率よく実行するために，階層的マスタ・ワーカ方式[8]を用いて並列化される．階層的マスタ・ワーカ方式では，WAN 上の複数の PC クラスタ間，および各 PC クラスタ内の計算ノード間の 2 階層において，マスタ・ワーカ方式を用いたタスク割り当てが実行される．本方式では，マスタと複数ワーカから構成されるプロセスグループを同一の PC クラスタ上で実行させることにより，頻繁に発生するマスタとワーカ間の通信が PC クラスタ上の高速ネットワーク内に局所化されるため，通信オーバーヘッドを軽減することができる．また，各 PC クラスタ上での未処理タスク数に応じて，WAN を介したタスクの移動が行われ，PC クラスタ間の適切な負荷分散が実現される．さらに，PC クラスタ内でのタスク割り当て時には，マスタ・ワーカ間の通信オーバー

† 東京工業大学／Tokyo Institute of Technology
‡ 科学技術振興機構さきがけ／PRESTO, JST
§ (株)NTT データ／NTT Data Corporation

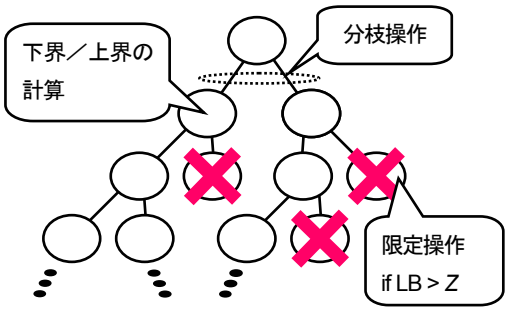

図 1. 探索木

ヘッド軽減と効率よい暫定値通知を考慮したタスク粒度調整が動的かつ自動的に行われ，計算の高速化が図られる．

本アプリケーションは，GridRPC ミドルウェアである Ninf-G[9]および Ninf[10]によりグリッド上に実装される．本実装では，WAN にまたがる複数の PC クラスタ間の通信は Ninf-G によりセキュリティ上安全に実行され，PC クラスタ内の通信は Ninf により高速に実行される．本アプリケーションをグリッド実験環境上に実装し，性能評価を行った結果，Ninf-G および Ninf を組み合わせて実装された階層的マスタ・ワーカ方式は，実行時間が 1[sec]未満の細粒度タスクから構成される並列分枝限定法アプリケーションをグリッド上で効率よく実行可能であること，またタスク粒度自動調整により，ユーザの手作業によるタスク粒度調整を行った場合と同等またはそれ以上の性能を自動的に得られることが確認された．

2. 並列分枝限定法アプリケーション

本節では，分枝限定法について概説するとともに，階層的マスタ・ワーカ方式を用いた並列化手法，およびタスク粒度の自動調整手法について述べる．

2.1 分枝限定法

分枝限定法は，与えられた問題を複数の子問題に分割することを再帰的に繰り返すことにより，最適解を探索する手法である．生成された子問題では，目的関数の下界，上界および解（暫定解）が計算される．また，下界および上界の値は，冗長な解探索を省く限定操作および計算の終了判定にも用いられる．

分枝限定法の手順は，図 1に示される探索木によって表される．分枝限定法では，初めに探索木の根ノードに相当する問題を複数の子問題（図中では 2 個の子ノードに相当）に分割する．本操作は，**分枝操作**と呼ばれる．次に生成された各子問題について，目的関数の下界と上界が計算され，さらに，これまでに計算された子問題の上界を比較することにより，その最小値である**暫定値**を計算する．（本稿では目的関数を最小化する最適化問題を扱っている．）以上の操作は，生成される子問題について再帰的に繰り返され，図 1 に示される探索木が生成される．暫定値は，冗長な探索を省くために行われる**限定操作**に用いられる．限定操作では，各子問題についてその下界（LB）と暫定値（Z）との比較が行われ，下界が暫定値よりも大きい子問題については，それ以上分枝操作を続けても最適解が得られないため，探索木から削除する．最後に，以上の操作を繰り返すことにより得られる暫定値と最小下界が一致するかその差が一定の閾値以下になった場合，全体の計算が終了する．

2.2 階層的マスタ･ワーカ方式による並列化

分枝限定法では，異なる子問題の計算は独立であるため，マスタ・ワーカ方式によりこれら子問題を並列に計算することができる[1][3][11]．本方式では，マスタと呼ばれる 1 プロセスが，探索木上の未処理子問題の計算を複数のワーカと呼ばれるプロセスに割り当てることにより，子問題計算が並列に実行される．しかし一般に通信オーバーヘッドの大きいグリッド上では，単純なマスタ・ワーカ方式による並列化では，マスタとワーカ間の通信時間が大きくなり，特に細粒度アプリケーションについては十分な実行時間短縮を行えないという問題がある[8]．

階層的マスタ・ワーカ方式は，グリッド上でのマスタ・ワーカ方式の性能低下を防ぐ手法の一つである．本方式では，単一のマスタと複数のワーカからなるグループが複数構成され，スーパバイザと呼ばれるプロセスがこれら複数のグループを統括する．タスクのワーカへの割り当ては，スーパバイザからマスタへの割り当て，マスタからワーカへの割り当てという 2 段階で行われ，ワーカ上の計算結果の回収は，逆の順序で同様に行われる．本方式では，同一グループに属するマスタとワーカを PC クラスタ等の単一計算システム上に配置することにより，マスタとワーカ間の通信を PC クラスタ上の高速ネットワーク内に局所化し，マスタ･ワーカ間の通信オーバーヘッドを軽減できるとともに，マスタの処理を複数の計算機に分散するため，単一マスタの処理がボトルネックとなる問題も解決できる．

図 2は，階層的マスタ・ワーカ方式による並列分枝限定法の概略を示す．図中，**S**，**M**，**W** はそれぞれ，スーパバイザ，マスタ，ワーカを意味し，$\mathbf{Z}_{\mathbf{Wi}}$，$\mathbf{Z}_{\mathbf{Mj}}$，**Z** は，それぞれワーカ **i** 上で計算された暫定値，マスタ **j** 上に保存されている暫定値，スーパバイザ上に保存されている暫定値をそれぞれ示す．

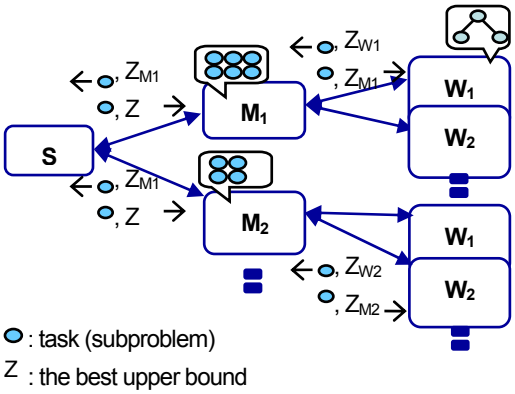

図 2. 階層的マスタ・ワーカ方式による
並列分枝限定法

単一マスタと複数のワーカから構成される各々のグループでは，マスタが複数のワーカに対して探索木上の未処理子問題の処理をタスクとして割り当てる．ワーカでは，割り当てられた子問題に分枝操作を行って新たな探索木（**部分木**）を生成し，部分木上の子問題について下界／上界計算，限定操作を実施した後，得られた解（暫定解），暫定値，限定操作により削除されなかった子問題をマスタに送信する．マスタは，受信した子問題を未処理子問題としてマスタ上のキューに登録するとともに，ワーカから受信した暫定値（Z_{Wi}）がマスタ上の暫定値（Z_{Mj}）よりも小さい場合は Z_{Mj} の値を Z_{Wi} の値に更新する．次にマスタは，アイドル状態のワーカに対して，キュー上の新たな子問題を割り当てると同時に，更新された Z_{Mj} を送信する．各ワーカ上で実施される限定操作は，ワーカ上に保存されている暫定値を用いて行われるため，あるワーカ上で算出されたより良い暫定値を他のワーカに迅速に通知することにより，限定操作の効率が向上し，全体の計算時間を短縮することができる．またマスタは，ワーカ上で分枝操作により生成される部分木の大きさを指定することにより，ワーカに割り当てるタスクの粒度を調整することができる．これは，ワーカ上の 1 タスク当りの計算量は，ワーカ上の分枝操作により生成される部分木の大きさに依存するためである．ここで，部分木の大きさは，部分木の深さにより指定される．例えば図 2では，右上のワーカ（W_1）上で深さ 1 の部分木が生成されている．

スーパバイザは，定期的に各マスタ上の負荷（具体的には未処理子問題数）およびマスタ上に保存されている暫定値（Z_{Mj}）を問い合わせる．ここで，複数のマスタ間で負荷の不均衡が生じている場合は，スーパバイザは，高負荷のマスタから低負荷のマスタへ子問題を移動することにより，負荷分散を行う．負荷分散アルゴリズムの詳細については [12]を参照されたい．マスタから通知された暫定値（Z_{M_j}）がスーパバイザ上に保存されている暫定値（Z）よりも小さい場合，スーパバイザは，Z の値を Z_{Mj} の値に更新するとともに，他のマスタに対して更新された Z を広報する．このように複数のマスタ間で最新の暫定値を共有することにより，ワーカ上での限定操作を促進し，冗長な探索を防ぐことができる．

2.3 タスク粒度自動調整

単一マスタと複数のワーカから構成される各々のグループ内では，マスタから割り当てられる子問題に対するワーカ上での処理（分枝操作，上界／下界計算および限定操作）に要する時間，即ちタスク粒度に関してアプリケーション性能におけるトレードオフが存在する．各グループ内の計算では，マスタからワーカへのタスク割り当てオーバーヘッド，および暫定値の通知頻度が全体の計算時間に影響を与える．前者については，これを減少させることによりアプリケーションの実行時間を短縮することができる．一方，後者については，これを増やすことにより各ワーカに最新の暫定値がより早く通知されるため，ワーカ上での限定操作を促進し，冗長な探索を防ぐことができる．しかしながら，オーバーヘッドを軽減するためには，ワーカ上での計算粒度を増やして相対的にタスク計算時間を大きくすることが必要であるのに対し，マスタからワーカへの暫定値通知頻度を増やすためには，ワーカ上でのタスク粒度を減らし，ワーカへのタスク割り当てを頻繁に行う必要がある．

本稿のアプリケーションでは，オーバーヘッドの軽減と効率よい暫定値通知を実現するために，タスク粒度の自動調整を行う．本手法では，マスタが，マスタ上に保存される暫定値の更新幅を監視し，更新幅が以下の式に定める閾値（th）より大きい場合は，次から割り当てるタスクについて，タスク粒度を減らす，即ちワーカ上の分枝操作で生成される部分木の深さを 1 レベル小さくするように指示し，また更新幅が閾値よりも小さい場合は，部分木の深さを 1 レベル大きく指示する．

$$th = a \times zdiff \qquad (1)$$

ここで $zdiff$ は，アプリケーション実行開始後に暫定値が初めて更新された時点の更新幅，a は定数パラメータを意味する．一方ワーカでは，マスタから割り当てられた子問題に対して，上記で指示された深さまで分枝操作を行い，計算結果をマスタへ送信する．また，ワーカ上で算出される暫定値をより早くマスタへ通知するために，ワーカは，ワーカ上でより良い暫定値

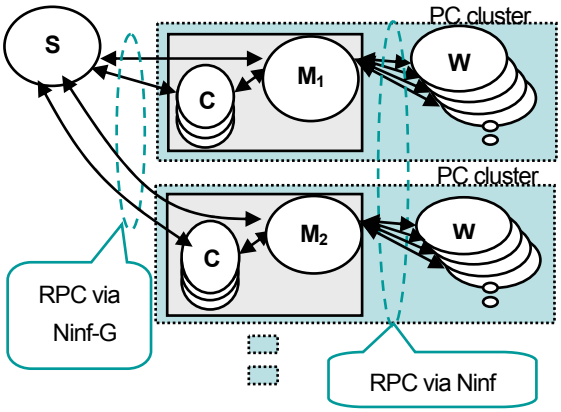

図 3. プロセス配置

が算出される毎に, 部分木上の全子問題計算の終了を待たずに，マスタへ暫定値を送信する.

本手法では, マスタ上の暫定値が大きく更新された場合は，並列計算オーバーヘッドが増大しても，タスク粒度を縮小することにより, 新しい暫定値がより早く他のワーカに通知され, 効果的な限定操作が期待できる．一方，暫定値が大きく更新されない場合は，効果的な限定操作があまり期待されないため, タスク粒度を増大することにより並列計算オーバーヘッドを削減することができる.

3. 実装

本稿では，異なるサイト，ドメインに属する複数の PC クラスタ群から構成されるグリッド上でアプリケーションを実行することを想定している. このようなグリッド上では, 前節で説明した各プロセスを計算資源へ適切に配置することが重要となる. 図 3は, 階層的マスタ・ワーカ方式におけるプロセスの計算資源への配置を表している. 本図では，複数の PC クラスタ（点線の長方形により表示）が WAN を経由して接続されており，図中の S，M，W はそれぞれ，スーパバイザ，マスタ，ワーカを示している．図中 C は，マスタと同一の計算ノード（実線の長方形により表示）上で実行されるプロセスであり，スーパバイザからマスタに対する処理, 具体的にはマスタ上の負荷や暫定値の問い合わせ，子問題の移動，暫定値通知等の処理を中継する.

スーパバイザとマスタ間の通信は，WAN を経由して行われるため, ユーザ認証等のセキュリティを考慮する必要があるのに対して, マスタとワーカ間の通信は同一の PC クラスタ内ネットワークを経由して行われるため, 高いセキュリティよりも高速な通信が要求される. そのため本アプリケーションでは，スーパバイザからマスタへの問い合わせや子問題の移動は，Ninf-G によりユーザ認証等を含むセキュリティを考慮して実行される. また，マスタからワーカへの子問題の割り当ては，Ninf により高速に実行される.

表 1. グリッド実験環境

	specification for a single node	Grid software	RTT [ms]
Client PC	PIII 1.0GHz, 256MB mem. 100BASE-T NIC	GTK 2.2 Ninf-G 1.1.1	
Blade	PIII 1.4GHz x2 512MB mem. 100BASE-T NIC	GTK 2.2 Ninf-G 1.1.1	0.04
Presto III	Athlon 1.6GHz x2, 768MB mem. 100BASE-T NIC	GTK 2.4 Ninf-G 1.1.1	1
Mp	Athlon 1.6GHz x2 512MB mem. 100BASE-T NIC	GTK 2.4 Ninf-G 1.1.1	20
Sdpa	Athlon 2GHz x2, 1024MB mem. 1000BASE-T NIC	GTK 2.4 Ninf-G 1.1.1	14

4. 性能評価

本稿が評価に用いたグリッド実験環境は, 表 1 に示す地理的に 4 つのサイトに分散したクライアント PC と 4 台の PC クラスタから構成される．このうち，クライアント PC と表中の Blade は，同一サイト（東京工業大学，神奈川県横浜市）に設置されている．クライアント PC と他の PC クラスタ，具体的には PrestoIII（東京工業大学，東京都目黒区），Mp（徳島大学，徳島県徳島市），Sdpa（東京電機大学，埼玉県比企郡）間の距離はそれぞれ，30[km]，500[km]，50[km]である．スーパバイザプロセスはクライアント PC 上で実行され，各 PC クラスタ上では，マスタと複数のワーカからなるグループがそれぞれ実行される．また，ユーザおよびホストの認証に用いる証明書は，AIST GTRC CA[13]から発行されたものを用いた.

本評価のベンチマーク問題としては，BMI 固有値問題[5]を用いる．BMI 固有値問題は，双線形行列関数の最大固有値を最小化する問題であり, 制御工学やオペレーションズリサーチの分野で取り扱われている[5][6].

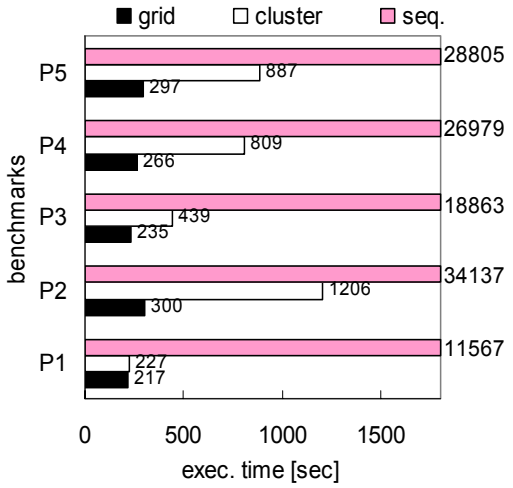

図 4. グリッド上での実行時間

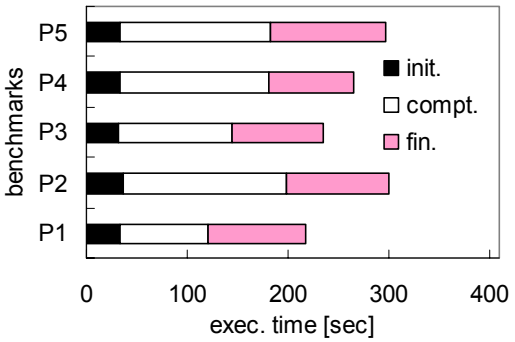

図 5. グリッド上での実行時間内訳

4.1 グリッド実験環境上での評価

図 4は，グリッド実験環境上で 5 種類のベンチマークプログラム（**P1**-**P5**）を求解した場合の実行時間を示す．本実験では，4 サイト上の PC クラスタから合計 348CPU（Blade 上の 73CPU，PrestoIII 上の 97CPU，Sdpa 上の 81CPU，Mp 上の 97CPU）を用いてアプリケーションを実行した. 使用したベンチマーク問題では，それぞれ問題サイズは同じ（解ベクトルサイズ= 6，係数行列サイズ=24×24）であるが，初期値として与えられる係数行列がそれぞれ異なる．図中，**seq** は，Blade 上の 1 ノード上での逐次実行時間，**cluster** は， Ninf を用いたマスタ・ワーカ方式により実装されたアプリケーションによる単一 PC クラスタ（Blade）のみの上での実行時間，**grid** は Ninf-G および Ninf を用いた階層的マスタ・ワーカ方式により実装されたアプリケーションのグリッド実験環境上での実行時間をそれぞれ示す.

図中の結果より，マスタ・ワーカ方式により並列化されたアプリケーションの PC クラスタ上での実行時間が逐次実行時間に比べて大きく短縮できていることがわかる．また，グリッド上で階層的マスタ・ワーカ方式を用いてアプリケーションを並列化することにより, 実行時間をさらに短縮できることがわかる. 例えば最も高い性能を示した **P2** では, 逐次実行では，約 9 時間半の実行時間を要していたが, グリッド計算により実行時間が 5 分に短縮されている．また，同様 **P2** のにグリッド上での実行時間は，単一 PC クラスタ上での実行に比べても 4 倍以上のスピードアップを得られている.

図 5 は, ベンチマーク問題のグリッド上での実行時間（**grid**）の内訳を示す. 図中，**init**，**compt**，**fin** は，それぞれ Ninf-G プロセスの初期化オーバーヘッド，計算時間，Ninf-G プロセスの終了処理オーバーヘッドをそれぞれ示す．本図の結果より，特に Ninf-G プロセス終了処理に要するオーバーヘッドが大きいことがわかり，これが **P1** に関してグリッド上での性能向上が単一 PC クラスタに比べて小さいこと要因の一つと考えられる．しかしながら，本アプリケーションでは，Ninf-G プロセス終了処理開始前にユーザに計算結果が提供される仕様になっており, ユーザの視点では, 図 4に示された実行時間よりも短い時間で計算結果を取得できている．例えば **P1** の場合，ユーザが計算を開始してから計算結果を得るまでの時間は 120[sec]であり，単一 PC クラスタ上での実行に比べて大きく短縮されている.

本評価で用いたベンチマーク問題の 1 タスク当りの平均実行時間は 1[sec]未満である．このような細粒度アプリケーションをグリッド上で従来のマスタ・ワーカ方式を用いて実行した場合，WAN 上で細粒度タスクを転送するオーバーヘッドが大きいため, 性能が大きく低下する[8]．これに対して，本評価の結果より, Ninf-G および Ninf を組み合わせて実装された階層的マスタ・ワーカ方式では，実行時間が 1[sec]未満の細粒度タスクから構成される並列分枝限定法アプリケーションをグリッド上で効率よく実行可能であることが確認された.

4.2 タスク粒度自動調整手法の評価

図5 は，Blade 上の 33CPU を用いてマスタ・ワーカ方式により並列化したアプリケーションを実行した場合の実行時間を示す. 評価に用いたベンチマークプログラム（**P6**-**P10**）では，3 種類の問題サイズを持つ問題（P6，P7 および P8，P9 および P10）から構成されており，係数行列がそれぞれ異なる．図中,

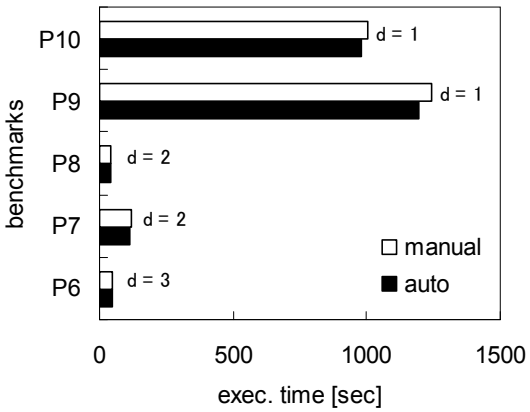

図 6. タスク粒度自動調整手法の評価結果

manual は, アプリケーション実行中に割り当てられる全てのタスクの粒度を固定とした場合の実行時間を示し, **auto** は2.3節で述べたタスク粒度自動調整手法を用いた場合の実行時間をそれぞれ示す. なお前者については, タスク粒度を試行錯誤的に決定した複数の条件下で得られた結果のうち最短の実行時間を示しており, グラフ右に記される **d** の値がその時のタスク粒度, 具体的にはワーカ上で生成される部分木の深さを示している. またタスク粒度自動調整手法では, (1)式において a=0.5 とした.

図の結果より, タスク粒度をアプリケーション実行中に固定する場合に比べて, タスク粒度自動調整手法を用いた場合の実行時間は同等か短縮されていることがわかる. 前者の場合に最適な性能を得るためには, 事前にユーザがタスク粒度の調整を試行錯誤的に行う必要があるが, 本結果より, タスク粒度自動調整手法は, 手作業により最適な調整を行った場合と同等またはそれ以上の性能を得られることが確認された.

5. まとめ

本稿では, グリッド上で並列分枝限定法アプリケーションを効率よく実行するための手法とその性能評価結果について報告した. 本アプリケーションをグリッド実験環境上に実装し, 性能評価を行った結果, Ninf-G および Ninf を組み合わせて実装された階層的マスタ・ワーカ方式を用いた本アプリケーションは, 実行時間が 1[sec]未満の細粒度タスクから構成される並列分枝限定法アプリケーションをグリッド上で効率よく実行可能であること, またタスク粒度自動調整により, ユーザの手作業によるタスク粒度調整を行った場合と同等またはそれ以上の性能を自動的に得られることが確認された.

謝辞 本研究の一部は, 科学技術振興機構計算科学技術活用型特定研究開発推進事業（ACT-JST）研究課題「コモディティグリッド技術によるテラスケール大規模数理最適化」の援助による.

参 考 文 献

[1] J. Goux, S. Kulkarni, J. Linderoth, and M. Yoder, An enabling framework for master-worker applications on the computational grid, In *Proc. the 9th IEEE Symposium on High Performance Distributed Computing (HPDC9)*, 2000.

[2] E. Heymann, M. A. Senar, E. Luque, and M. Livny, Adaptive scheduling for master-worker applications on the computational grid, *Proc. of the 1st IEEE/ACM International Workshop on Grid Computing (Grid2000)*, 2000.

[3] M. O. Neary and P. Cappello, Advanced Eager Scheduling for Java-Based Adaptively Parallel Computing, *Proc. of the 2002 joint ACM-ISCOPE conference on Java Grande*, 2002.

[4] H. Takemiya, K. Shudo, Y. Tanaka and S. Sekiguchi. Development of Grid Applications on Standard Grid Middleware, *Porc. of the GGF8 Workshop on Grid Applications and Programming Tools*, 2003.

[5] H. Fujioka and K. Hoshijima. Bounds for the bmi eigenvalue problem. Trans. of the Society of Instrument and Control Engineers, 33(7):616–621, 1997.

[6] M. Fukuda and M. Kojima, Branch-and-Cut Algorithms for the Bilinear Matrix Inequality Eigenvalue Problem, *Computational Optimization and Applications*, 19(1):79-105, 2001.

[7] H. Kasahara and S. Narita, Practical Multiprocessor Scheduling Algorithms for Efficient Parallel Processing, *IEEE Trans. on Computers,* C-33(11), pp.1023-1029, 1984.

[8] K. Aida, W. Natsume and Y. Futakata, Distributed Computing with Hierarchical Master-worker Paradigm for Parallel Branch and Bound Algorithm, *Proc. of the 3rd IEEE/ACM International Symposium on Cluster Computing and the Grid (CCGrid 2003)*, 2003.

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

[10] S. Matsuoka, H. Nakada, M. Sato and S. Sekiguchi, Design Issues of Network Enabled Server Systems for the Grid, *Grid Computing – Grid 2000, LNCS1971*, pp.4-17, 2000.

[11] K. Aida, Y Futakata and S, Hara, High-performance Parallel and Distributed Computing for the BMI Eigenvalue Problem, Proc. 16th IEEE International Parallel and Distributed Processing Symposium (IPDPS 2002), 2002

[12] 大角，合田，階層的マスタワーカ方式を用いたグリッドアプリケーションにおける負荷分散の性能評価，情報処理学会 HPC 研究会，2004 年 8 月

[13] ApGrid, http://www.apgrod.org/