ヘテロなクラスタ環境における並列 LINPACK アルゴリズム

笹生　健[†]　松岡　聡[†]　建部修見[††]

本研究では LINPACK Benchmark のひとつである HPL をヘテロなクラスタ環境向けに最適化して実装し，CPU が異なるノードの混在する CPU ヘテロなクラスタ上での評価を行なった．ヘテロなクラスタ環境においては各ノードの性能差に応じたロードバランスをとる事が実行効率の向上には不可欠であり，ヘテロな環境向けの最適なアルゴリズム研究が必要である．本研究で用いた HPL の最適化手法は，各ノードにおける CPU 性能やメモリサイズ等の性能差に応じて各プロセスをプロセス格子に複数割り当てる事で割り当てるデータサイズを変えてロードバランスを取るというものである．それにより，ピーク性能において理論性能の 57.1%，通常の HPL と比べ最大 1.49 倍の性能を達成した．

An Efficient LINPACK Algorithm for Heterogeneous Clusters

Takeru Sasou ,[†] Satoshi Matsuoka [†] and Osamu Tatebe[††]

In a heterogeneous clustering environment, appropriate load balancing of respective computing nodes with differing computing capacities is of utmost importance to achieve maximum performance as a whole. As such, study of parallel algorithms that can adapt to arbitrary heterogeneous settings are necessary, especially parallel benchmarks. Unfortuantely, most parallel benchmarks assume uniform performance across the entire machine, and will suffer substantial performance penalty if executed in hetergeneous settings, including HPL, the High-Performance Linpack, whose algorithm has numerous inherent uniformity assumptions. We show that, it is possible to appropriately allocate different numbers of submatrices to achieve load balancing in HPL. Although modifications to the algorithm is substantial, there is no penalty when performance is uniform, achieved nearly 50% speedup in a heterogeneous setting compared to the original HPL.

1. はじめに

近年，コモディティハードウェアを使用する事によるコストパフォーマンスの良さから超並列計算機に代わってパーソナルコンピュータを繋いだコモディティクラスタ型並列計算機が注目されている．クラスタは初期の頃こそ 10Mbps〜100Mbps の低速なネットワークであったが，最近ではギガビット級のネットワークを利用できるようになった事から，世界の高性能コンピュータの上位 500 位をリストアップしている Top500[1] においても多くのクラスタ型並列計算機がランクインするようになってきており，現在も普及の一途を辿っている．

クラスタ型計算機はコモディティハードウェアによって構築されていることから，本質的に多様性をもっている．その多様性は CPU 性能やネットワークハードウェアの種類，メモリ容量の差異，場合によっては OS や CPU アーキテクチャまで及ぶことがある．今後クラスタが普及していくにつれ，ハードウェアの段階的なアップグレードによるノード性能の不均質 (ヘテロ) 化，グリッド環境でのクラスタの複数台同時使用による大規模計算や資源の有効利用といった理由から，この様なヘテロなクラスタ環境の増加は必至である．

性能ヘテロなクラスタ環境においてはノード間の性能差を考慮して負荷分散をすることが計算効率の向上に必要であり，アプリケーションレベルで対応するべき問題である．ところが，高性能にチューニングされた並列アプリケーションは多くの場合プロセッサ間の均質な性能を前提としており，例えばバリア通信をともなう場合などは，プロセッサ間のバリア到達のタイミングが大幅にずれ，全体のピーク性能に対して著しく性能ロスが生じる．従って，アルゴリズムおよびアプリケーションレベルで性能ヘテロな環境に性能ロスなく対応することが今後クラスタ環境では重要になる．

しかし，既存の NAS Parallel Benchmark[2] や LINPACK Benchmark[3] の参考用実装を始めとした，数多く存在するクラスタ型並列計算機向けのベンチマー

† 東京工業大学
Tokyo Institute of Technology
†† 産業技術総合研究所
National Institute of Advanced Industrial Science and Technology

クソフトウェアは各ノードの性能が均質 (ホモ) な環境で動くことを前提としており，各ノードに対して同量の負荷分散を行なっているため，様々な条件のヘテロなクラスタ環境についてポータブルに性能評価をする事ができず，クラスタシステムの性能評価という要求に対し充分に応える事ができない．そこで, 我々はヘテロな計算環境に最適化されたアルゴリズム研究を目的とし, NAS Parallel Benchmark[2] や LINPACK Benchmark[3] の参考用実装など, 既存のクラスタ型並列計算器用のベンチマークソフトウェアをヘテロなクラスタ向けに最適化して実装し, 性能評価を行っている．これらの代表的なアルゴリズムおよびベンチマークの性能ヘテロの対応を実際に行うことにより，1) 性能ヘテロなアルゴリズムを探求し，かつ 2) 一般的なアプリケーションのヘテロ化の戦略の指針・パターンなどを示せる．のみならず，3) NAS や Linpack など，標準ベンチマークのヘテロ化により，実際に性能ヘテロなクラスタの性能評価が正当に行えるようになる．

本研究では，その一環として，Tennessee 大学の J.Dongarra らによって開発された LINPACK Benchmark プログラムのひとつである HPL(High-Performance Linpack Benchmark)[4] を，ヘテロなクラスタ環境向けにアルゴリズムを改善した．HPL は非対称密行列連立 1 次方程式を解く際の演算性能を評価するソフトウェアであり，実行時に様々なパラメータをシステムの構成に合わせて設定する事で，より性能を引き出した性能評価ができるという特徴がある．

本研究で採った最適化手法は，各ノードにおける CPU 性能やメモリサイズ等の性能差に応じて各プロセスをプロセス格子に複数割り当てる事で割り当てるデータサイズを変えてロードバランスを取るというもので, ヘテロ向けに最適化した HPL を CPU ヘテロなクラスタ上で実装し評価を行なった．その結果, PentiumIII 800MHz×3 + PentiumII 400MHz×1 の場合でピーク性能で 1.586GFlops，800MHz×2 + 400MHz×2 の場合ではピーク性能で 1.373GFlops というように，元の HPL でのピーク性能 1.064GFlops，1.045GFlops に対して最大 1.49 倍の性能を達成し，ヘテロ向けの最適化手法が有効である事を示した．

2. HPL

ここでは HPL の概要と，ヘテロなクラスタ環境上で HPL を使用する上で起きる問題について述べる．

2.1 HPL の概要

HPL は，フリーのオープンソースとして提供されている LINPACK Benchmark プログラムである．

LINPACK Benchmark は非対称密行列連立 1 次方程式を解く際の演算性能を評価するもので，主に浮動少数点演算のためのベンチマークであり，規定では解法に関する制約は無いが，演算量は $\frac{2}{3}N^3 + O(N^2)$ で評価する事が決められている．これは係数行列を LU 分解した後, 前進・後退代入演算によって解ベクトルを求めるという直接解法を適用する事を前提にした演算量である．LINPACK Benchmark では係数行列を各要素が乱数で決められた非対象密行列としていることから, 直接解法を適用するのが一般的である．

LINPACK Benchmark は現在

- LINPACK Fortran $n = 100$ Benchmark
- LINPACK $n = 1000$ Benchmark (Toward Peak Performance)
- LINPACK's “ Highly Parallel Computing ” Benchmark

の 3 種類があり，LINPACK Fortran $n = 100$ Benchmark は問題サイズを $n = 100$ で固定, 規定によりソースコードの改変も認められていないもので，LINPACK $n = 1000$ Benchmark は問題サイズは $n = 1000$ で固定だが，ユーザーがソースコードを改変する事が認められている．Highly Parallel Computing は Top500 レポートで使用されており，数値解にある一定以上の精度が要求されるが問題サイズや解法は規定されない．HPL も Highly Parallel Computing のひとつである．

HPL はその実行に際し，MPI1.1[5] 準拠のメッセージ通信ライブラリ，BLAS[6],VSIPL[7] のどちらか 1 つの線形計算ライブラリがシステム上で利用できる必要がある．

HPL では，まず係数行列を複数の正方形のパネルブロックに分割して 2 次元プロセス格子上にブロックサイクリックに分割する．例えば 2 × 3 のプロセス格子にデータ分割すると図 1 の様になる．連立 1 次方程式を解く際に使用するアルゴリズムは外積形式ブロック *LU* 分解アルゴリズムであり，主に`Panel Factorization`, `Panel Broadcast`, `Update`, `Backward Substitution` というフェイズからなっている．それぞれにおいてはパネル列 LU 分解, 分解済パネルの送信, 未分解小行列の更新計算, 後退代入演算による $Ux = y$ の求解を行なう．その際 HPL では`Panel Factorization` において row partial pivoting のためにパネル列内のプロセス間で allgather 通信を行なう．そのため縦方向の同期通信が頻繁に起こる．

また, HPL は問題サイズの他, ブロックサイズ, 再帰的パネル LU 分解のアルゴリズム (外積法, 内積法, クラウト法), 再帰の枝数, 再帰を停めるブロックサイズ, 再帰を停めた後の LU 分解アルゴリズム (外積法, 内積法, クラウト法), パネルブロードキャストの通信トポロジー (increasing-ring, 修正 increasing-ring, increasing-2-ring, 修正 increasing-2-ring, spread-roll), ピボットパネルの先読み段数, ピボット行交換の方法 (binary-exchange, spread-roll, mix) 等, 様々なパラメータを計算機の特性に合わせて設定する事で，より高い性能を得る事ができるようになっている．

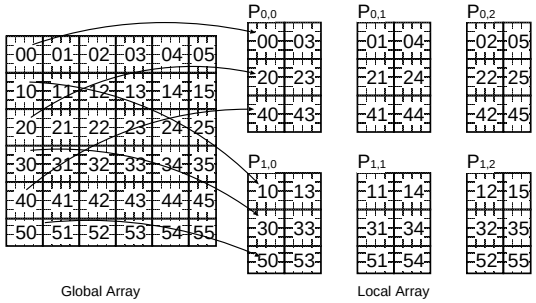

図 1 ブロックサイクリックなデータ分割

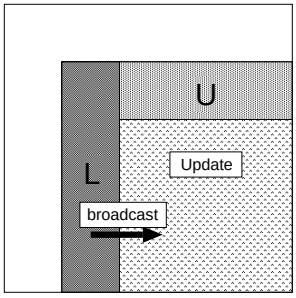

図 2 ブロック LU 分解

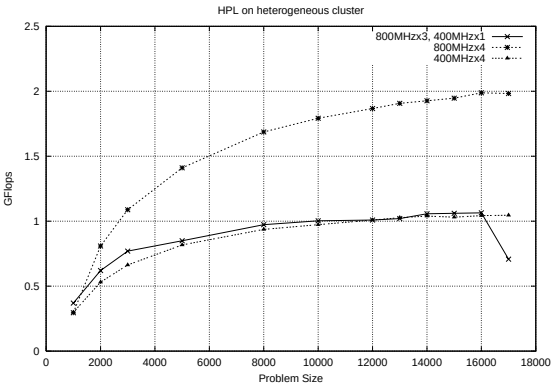

図 3 CPU ヘテロなクラスタ上での HPL の性能

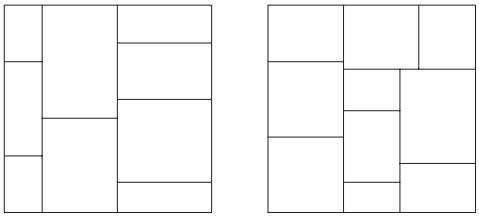

図 4 ヘテロな行列データ分割の例

2.2 ヘテロなクラスタ上での HPL の問題点

HPL ではブロックサイクリックにデータ分割されるため，各 PE(プロセッサエレメント) にはほぼ同量のデータが割り当てられ，性能ホモなクラスタにおいてはロードバランスがとれる．しかし，各ノードの演算性能に差異があるようなヘテロな構成のクラスタ上で HPL を実行した場合，全ての PE にかかる負荷が同量なので演算性能の高い PE が性能の低い PE の処理が終わるのを待つ時間が生じてしまい，全体の性能が落ちてしまう．実際，800MHz×3 + 400MHz×1 という CPU ヘテロな構成のクラスタ上で HPL を実行した場合，図 3 に見られる様に, 800MHz×3 + 400MHz×1 での実行性能は 400MHz×4 の実行性能とほぼ同じであり, 性能の低い PE に足を引っ張られて充分な性能が出ない事がわかる．

3. ヘテロなクラスタ向け HPL

3.1 ロードバランス

ヘテロなクラスタ環境においては，CPU 性能，メモリ容量，ネットワーク性能などハードウェア構成が異なるノードが混在する．そのため，ヘテロなクラスタ上で並列計算を行なう場合，各ノードの性能に応じたロードバランスやスケジューリングが性能向上のために必須となる．例えば，ノード間で CPU 性能に差がある場合，CPU 性能が高い方が単位時間内での演算量は多いので，性能が高い方により多く負荷をかけた方がクラスタ全体の性能が上がる．メモリ容量に差がある場合は，同じサイズでデータ分散するよりもメモリ利用効率が高くなるようにサイズを変えてデータ分散した方がより大規模なサイズのデータを処理できる．

上で述べた様な特性のために，2 章で示したように HPL はヘテロなクラスタ環境上では性能が低下してしまうので，ヘテロなクラスタの構成を考慮してロードバランスを取れるようにする必要がある．

そこで，本研究では HPL をヘテロなクラスタ向けにするため，各 PE の性能に応じて割り当てるデータサイズを変えることでロードバランスをとるようにした．その際，割り当てるサイズをまちまちにした行列データの分割形式として図 4 の様な分割形式が考えられる．しかし，図 4 の分割形式は各 PE へ割り当てるデータサイズを細かく調整できるという利点があるものの，格子分割でなくなるため，`Update` 処理における消去計算で参照するデータの通信サイズを減らせるというブロック LU 分解が適用できなくなるという欠点がある．HPL では通信のオーバーヘッドが性能低下に大きくつながるため，PE 間のロードバランスを取ることによるメリット以上にデメリットの方が大きくなってしまい，有効ではない．

よって，本研究ではブロック LU 分解アルゴリズムを使用しつつ，各 PE へ割り当てるデータサイズを可変にする手法として，一つの PE がプロセス格子上の複数のブロックを受け持つようにするという手法を提案する．例えば，P_0 が他の PE の 2 倍の演算性能をもっている場合，図 5 の様に割り当てると全ての PE

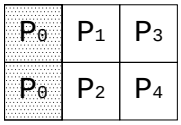

図 5 ヘテロなプロセス格子割り当て

の処理時間が同じになり，演算性能の高い PE が性能の低い PE の処理を待つ事が無くなる．また，P_0 が他の PE の 2 倍メモリ領域を使用できる場合も，図 5 の様に割り当てることでメモリをより有効に利用できる．

3.2 実装の方法

最適化手法を実現する方法は主に以下の 2 つである．

- 1PE 上でプロセスを複数起動する
- 1 プロセスで複数のパネルを処理できるようにする

前者はプログラムに変更を加えなくてもいいので容易に実現できるが，1PE 上で複数のプロセスが動くため，単純に通信回数が増大し，また必ずスリープ状態のプロセスが存在するためプロセス同士の同期を取るのが困難である．HPL では縦方向の同期通信が頻繁に起こるので，うまく同期が取れないと通信のオーバーヘッドが増大してしまう．後者はアルゴリズムなどを変更しなければならないが，うまくロードバランスをとれば各プロセスが同期的に動く事が可能である．なお，4 章でこの 2 種類の方法の性能比較を行なう．

1 プロセスで複数のパネルを処理できるようにした HPL(以下ヘテロ版 HPL) では LU 分解の計算手順の変更はしない．ただし，各プロセスが複数のブロックパネルを処理するようになったので，各パネルの属する列を調べて各フェイズにおいて処理するべきパネル列毎にまとめて処理するようにする．また，横方向で隣接するプロセスが 2 つ以上であったり，パネルのデータ通信相手が自プロセス内の異なるパネルになる場合があるので通信に関わる部分に変更が必要である．また，通信相手の特定や，通信相手の保持するパネルの情報を取得するためにプロセス格子への各プロセスの割り当てのテーブルを保持するようにした．

次に各計算フェイズにおける変更を述べる．

3.2.1 パネル LU 分解

パネル LU 分解では，列の部分ピボッティングにおけるピボット行交換のためにプロセス列中のプロセス同士で binary-exchenge 形式の allgather 通信を行う．この時，処理するべきいくつかのパネルのうちから 1 列だけ選んで交換すればよいので，次のような手順で処理を行うようにした．

(1) プロセス中の処理するべき全てのパネルの最大要素を調べ，最大要素を含む行をもつパネルを選択する．

(2) プロセス間で binary-exchenge 通信を行い，軸として選択された行配列と対角要素を含む行配列を取得する．

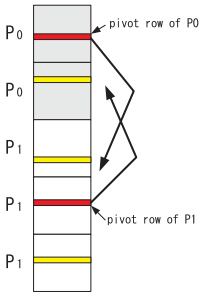

図 6 ピボット行交換通信

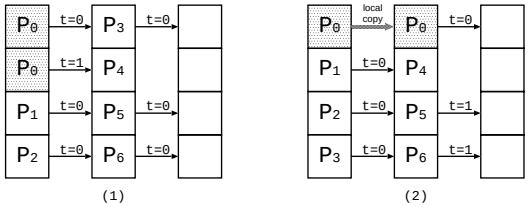

図 7 ヘテロな Broadcast

(3) 選択軸とローカルな行配列のスワップを行ない，選択軸のバッファへのコピーを行う．

なお，現在は部分軸選択付き LU 分解，再帰的 LU 分解ともに right-looking 形式のアルゴリズムのみ実装している．

3.2.2 Panel Broadcast

HPL では，各プロセスが一つのパネルを一つのプロセスに送信すればよかったが，ヘテロ版 HPL では通信相手が 2 つ以上になることもあり，その場合図 7(1) の様に通信ステップが増えるために他のプロセスが`Update` 処理に入るまでに待ち時間が生じてしまう．

また，同じパネル列内に同プロセスが保持するパネルがある場合も図 7(2) の様に他のプロセスとの足並みが乱れる事となる．

現在の実装では，保持するパネルを一つずつ Broadcast し，同じパネル列に属する同プロセス内のパネル同士ではローカルにコピーするようになっている．

3.2.3 Update

`Update` では，まず最初に縦方向に U の Broadcast と行交換のための allgather 通信を行い，本来 U とスワップするべき行配列と U パネルを送受信しあう．ヘテロ版 HPL ではこれを次のような手順で行う．

(1) 同じパネル列に属するパネルがそれぞれ持っている U とスワップするべき行配列を一つの作業領域に集める．

(2) 作業領域に集めたデータを用いて allgather 通信を行う．U パネルを受信したときは，全てのパネルで行交換を行う．

(3) allgather 通信が終わったら，U パネルを全てのパネル構造体にコピーする．

また，プロセスが複数のプロセス列内に属している

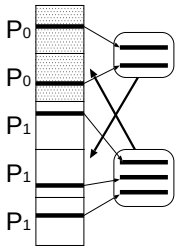

図 8 Update フェイズでの行交換通信

CPU	Intel PentiumIII 800MHz×2
Motherboard	MSI MS-6120N
Memory	640MB
HDD	28.8GB
Network Card 1	DEC 21140AF(Planex)
Network Card 2	ADMtek Centaur-P(AN983)
Network Card 3	ADMtek Centaur-P(AN983)
Operating System	Redhat Linux release6.2(Zoot)
	Kernel 2.2.16

表 1 PrestoII クラスタの仕様

場合，あるパネル列の allgather 通信を終わるまで異なるパネル列の allgather 通信が行えないが，この通信が最初に 1 回だけ行われた後は各プロセスは独立して処理ができる．そこで，保持するパネルの allgather 通信を全て最初に済ませてしまい，その後で消去計算を行う．

4. 性能評価

ここでは CPU がヘテロなクラスタ環境において HPL を 1PE1 プロセスずつで実行した場合 (`1PE:1-Process`)，HPL を 1PE 複数プロセスで実行した場合 (`1PE:Multi-Process`)，ヘテロ版 HPL を実行した場合の 3 通りで性能比較を行い，結果についての考察を述べる．

評価実験には松岡研究室[9] の PrestoII クラスタ[10] を使用した．表 1 に PrestoII の個々のノードの仕様を示す． PrestoII をヘテロな環境とするために，CPU を PentiumII 400MHz に交換したノードを混ぜて，CPU がヘテロな環境を構築した．現在の実装では通信部分にバグがあり，1×Q という形状のプロセス格子でしか実行できないため，ノード数を増やした際の並列化効率が良くないので，4 ノード使用の小規模な構成で実験を行なう事にした．

評価は 800MHz×3 + 400MHz×1，800MHz×2 + 400MHz×2 の 2 つの場合で行ない，問題サイズは N=1000〜17000 の間で計測した．

各 PE のプロセス格子への割り当てを図 9，10 に示す．HPL の実行性能はプロセス格子の構成で変化するので, ここではそれぞれ最も実行性能の良い構成を選んでいる．

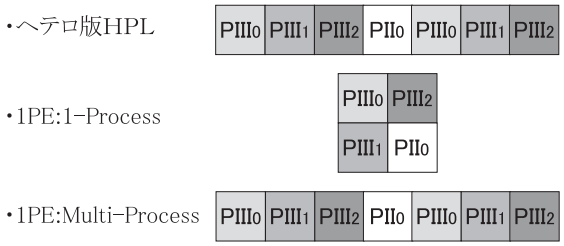

図 9 $PIII_{0,1,2}$=800MHz, PII_0=400MHz

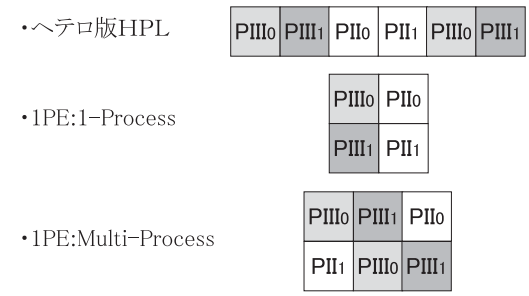

図 10 $PIII_{0,1}$=800MHz, $PII_{0,1}$=400MHz

HPL をコンパイルする際のライブラリや設定は次のようになっている．

- MPI : mpich-1.2.0[5]
- BLAS : ATLAS 3.2.0[8]
- コンパイルオプション

```
-O6 -DAdd_ -DF77_INTEGER=int
-DStringSunStyle -DHPL_CALL_CBLAS
```

また, 問題サイズ, プロセス格子以外の実行時パラメータは NB=120, 再帰的 LU 分解=right-looking, LU 分解=right-looking, NBMIN=2, NDIV=2, ブロードキャストトポロジー=increasing-ring, look-ahead depth=0, ピボット行交換通信=mix, threshold=16.0, swapping threshold=64, L1=transposed, U=transposed, Equilibration=yes, momory alignment in double=8, とする．

4.1 ロードバランスの最適化

各 PE の CPU の違いによる演算性能の差を埋めてそれぞれの処理の足並みをそろえるためには，実行時間の逆比となるサイズでブロックを割り当てれば良い．しかし，各計算フェイズにおけるオーダーは異なるため，どれか一つについて最適化することになる．

次元数 N の行列を $P \times Q$ のプロセス格子上に分割した時の各計算フェイズのオーダーは以下のようになる．

- `Panel Factorization`

$$\frac{3}{2P} \cdot N^2 + O(N)$$

- `Update`

全体の実行時間	3605.02 秒
Panel Factorization	20.01 秒
Update	3042.08 秒
Backward Substitution	1.35 秒

表 2 PrestoII クラスタ上での HPL 実行時間内訳 (N=65000, NB=120, P×Q=8 × 16)

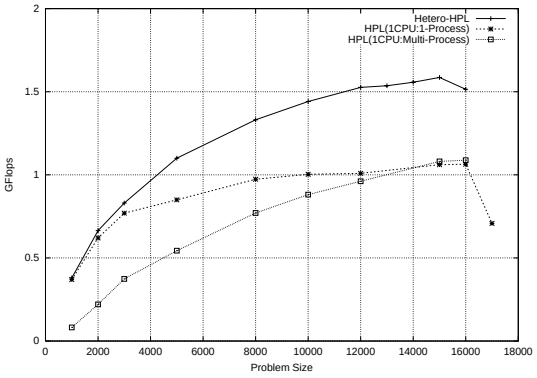

図 11 HPL とヘテロ版 HPL の比較 (800MHz×3, 400MHz×1)

$$\frac{2N^3}{3PQ} + \frac{P+Q}{PQ} \cdot O(N^2) + O(N)$$

- Backword substitution

$$\frac{1}{PQ} \cdot O(N^2)$$

これに見られるように，処理の大部分はUpdate に費される．実際, PrestoII クラスタ 64 ノードにおける HPL の実行時間の内訳 (表 2) を見ると, 全体の実行時間のおよそ 84.4%がUpdate の処理時間である．実行時のパラメータ (N=65000, P×Q=8×16) を上記の式に代入すると，Update 処理の割合は 99.9%となる。実際の実行時間の割合と比べて大きく差があるが、これは通信のオーバーヘッドを含まない値であるためである。

以上から，割り当てるデータサイズの比をUpdate フェイズの処理にかかる時間の逆比とすることでロードバランスを取ることにした．適当な問題サイズで 1CPU での実行時間を測定したところ，400MHz と 800MHz それぞれのUpdate 処理時間の逆比はおよそ 1 : 1.988 となったので，データサイズの比は 1 : 2 とした．

4.2 評価結果

N=1000〜17000 でそれぞれ計測した結果を図 11,12 及び表 3,4 に示す．表 3,4 において未測定となっている箇所は，実行中にメモリのスワップが起きてしまうので最後まで測定していないものである．

4.3 考　　察

ヘテロ版 HPL は 400MHz×1(図 11, 表 3) においてピーク性能 1.586GFlops であり，理論性能 2.8GFlops の 56.6%，1PE:1-Process のピーク性能 1.064GFlops の 1.49 倍を達成している．400MHz×2(図 12, 表 4) においてもピーク性能 1.373GFlops であり，理論性能 2.4GFlops の 57.1%，HPL のピーク性能 1.045GFlops の 1.31 倍を達成している．これはヘテロ向けの最適化が有効である事を示している．

また，性能ホモな状態では PrestoII 上での HPL の

N	ヘテロ版 HPL	HPL 1PE:1-Process	HPL 1PE:Multi-Process
1,000	0.3798	0.3698	0.08186
2,000	0.6642	0.6202	0.2204
3,000	0.8300	0.7696	0.3738
5,000	1.100	0.8495	0.5438
8,000	1.331	0.9734	0.7706
10,000	1.441	1.003	0.8808
12,000	1.526	1.009	0.9612
13,000	1.536	1.020	1.000
14,000	1.557	1.057	1.050
15,000	1.586	1.061	1.081
16,000	1.515	1.064	1.088
17,000	未測定	0.7079	未測定

表 3 HPL とヘテロ版 HPL の比較 (800MHz×3, 400MHz×1)

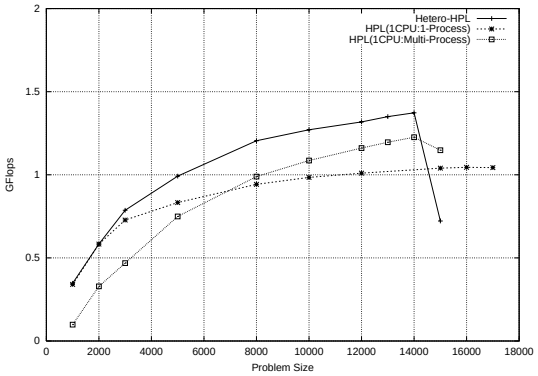

図 12 HPL とヘテロ版 HPL の比較 (800MHz×2, 400MHz×2)

N	ヘテロ版 HPL	HPL 1PE:1-Process	HPL 1PE:Multi-Process
1,000	0.3474	0.3407	0.09841
2,000	0.5836	0.5833	0.3288
3,000	0.7868	0.7277	0.4684
5,000	0.9920	0.8330	0.7494
8,000	1.205	0.9427	0.9891
10,000	1.271	0.9844	1.086
12,000	1.318	1.010	1.161
13,000	1.350	1.020	1.196
14,000	1.373	1.031	1.226
15,000	0.7219	1.040	1.148
16,000	未測定	1.045	未測定
17,000	未測定	1.043	未測定

表 4 HPL とヘテロ版 HPL の比較 (800MHz×2, 400MHz×2)

対理論性能対の割合はおよそ 61.2%である事から考えると，ヘテロ版 HPL は性能効率が若干低下しているが，これは通信トポロジーが (P,Q)=1 × 4 のプロセス格子と同じであることによる性能効率の低下である．HPL では`Update` 処理の計算量に $\frac{P+Q}{PQ} \cdot O(N^2)$ の項があるため, 一般的にプロセス格子が正方形に近い方が計算量が少なくなる．実際，(P,Q)=1 × 4では，性能ホモな PrestoII 上での HPL の対理論性能比は 58%程度である．

一方，`1PE:Multi-Process` は，400MHz×1(図 11, 表 3) においてピーク性能 1.088GFlops，400MHz×2(図 12, 表 4) においては 1.226GFlops というようにヘテロ版 HPL と比べてだいぶ性能が劣る．これは，同じ PE 上で複数のプロセスを起動している場合，あるプロセスが動いている間は同 PE 上の他プロセスはスリープ状態となるため，プロセス間で通信を行なおうとしてもどちらかのプロセスがスリープしているという状態が生じて，通信部分がオーバーヘッドとなったからであると考えられる．なお，ヘテロ版 HPL は`1PE:1-Process` よりも問題サイズが小さいところでピーク性能に達しているが，これは CPU400MHz の PE には CPU800MHz の PE の $\frac{1}{2}$ のサイズしかデータが割り当てられていない事により全体のメモリ利用効率が悪くなっているためである．

5. おわりに

本稿では分散メモリ型並列計算機用のベンチマークソフトウェアである HPL をヘテロなクラスタ環境向けに最適化し，実装してこれを評価した．性能評価では，CPU がヘテロな環境において HPL，HPL をナイーブな最適化の実装をしたものとの比較を行なった．

ヘテロ向けの最適化により，CPU が PentiumIII 800MHz のノードと PentiumII 400MHz ノードが混在するヘテロなクラスタ環境においてヘテロ版 HPL はヘテロ向けの最適化を施さない HPL と比べ最大 1.49 倍の性能を達成し，これにより本研究で提案したヘテロなクラスタ向けの最適化手法が有効であることがわかった．

今後の課題としては，次のようなものが挙げられる．

- プロセス格子への割り当て方による性能差の検証
- メモリサイズがヘテロな場合での評価
- ネットワークがヘテロな場合での評価
- 割り当てるデータサイズなどパラメータの自動チューニング
- HPL 以外のベンチマークソフトウェアのヘテロ対応化

Panel Broadcast の際に一つのプロセスから複数のプロセスへメッセージを送信する事から生じる通信ステップ数の差を, プロセス格子への割り当て方を変える事で減らす事ができるので, プロセス格子の最適な割り当て方による性能差や効率の良いプロセス格子の構築手法などを検証する必要がある．

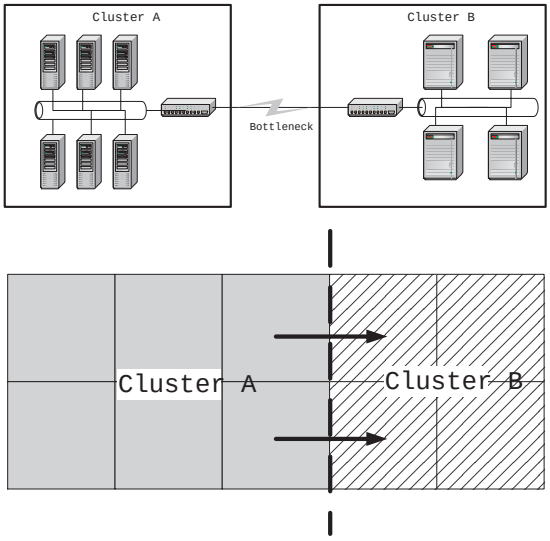

図 13 ヘテロなネットワーク環境における PE のグループ分け

また，演算速度に応じて割り当てるデータサイズを変えるという手法では，演算性能の低い PE のメモリ利用効率が悪くなるため，クラスタの構成次第では逆に性能が遅くなることも考えられ，これについても検証が必要である．

ネットワークがヘテロになる状況は, 現実的にはスイッチの階層化でのスイッチ間接続や, クラスタを複数台使用する際のクラスタ間ネットワークなどのように, ネットワークの特定の一部分がボトルネックとなるだけで, その他の PE を PE 間のネットワーク性能が同じであるもの同士でグループ分けする事ができる場合である．そのような場合, 図 13 の様に, グループ内でのネットワーク性能が同じとなるグループでプロセス格子をグループ分けしてしまい, 行方向通信に比べて回数が圧倒的に少ない Panel Broadcast の列方向通信で 1 回だけネックとなるスループットの低い通信を行うようにするという手法が有効であると予想される．

データサイズを決定するための要因は CPU 性能やメモリサイズ，ネットワーク性能など，実行時に静的に与えられるものであり，一度クラスタ構成に関する情報を取得しておくことで最適なデータサイズの自動セッティング機能は実現できる．今後は様々なヘテロ環境上での今後実装する予定である．

本稿では HPL のアルゴリズムのヘテロ対応を行っているが，現在我々は NAS Parallel Benchmarks のヘテロ対応アルゴリズムの研究と実装も行っており，今後，様々な並列アルゴリズムに対してヘテロ対応をしていく事で，ヘテロな計算環境向けの，より一般的なアルゴリズムモデルを研究していく予定である．

参 考 文 献

1) TOP500 Supercomputer Sites:TOP500 Supercomputer Sites, http://www.top500.org/
2) The NAS Parallel Benchmarks, http://www.nas.nasa.gov/Software/NPB/
3) http://www.netlib.org/benchmark/top500/lists/linpack.html
4) A.Petitet, R.C.Whaley, J.Dongarra, A.Cleary: HPL - A Portable Implementation of the High-Performance Linpack Benchmark for Distributed-Memory Computers, http://netlib.org/benchmark/hpl/index.html
5) MPICH Home Page, http://www-unix.mcs.anl.gov/mpi/mpich/
6) BLAS (Basic Linear Algebra Subprograms), http://www.netlib.org/blas/
7) Vector Signal Image Processing Library, http://www.vsipl.org/
8) Automatically Tuned Linear Algebra Software (ATLAS), http://www.netlib.org/atlas/
9) http://matsu-www.is.titech.ac.jp/
10) http://cluster-team.is.titech.ac.jp/
11) 小国力 編著, 村田健郎, 三好俊郎, ドンガラ.J.J, 長谷川秀彦 著: 行列計算ソフトウェア　WS，スーパーコン，並列計算機.
12) S. Toledo: Locality of Reference in LU Decomposition with partial pivoting, SIAM Journal on Matrix. Anal. Appl., Vol. 18, No. 4, 1997.
13) J. Choi, J. J. Dongarra, and D. W. Walker: Parallel Matrix Transpose Algorithms on Distributed Memory Concurrent Computers, Parallel Computing, Vol. 21, pages 1387-1405, 1995.
14) J. Dongarra, R. van de Geijn and D. Walker: Scalability Issues in the Design of a Library for Dense Linear Algebra, Journal of Parallel and Distributed Computing, Vol. 22, No. 3, pp. 523-537, 1994.
15) P.E. Strazdins: Lookahead and Algorithmic Blocking Techniques Compared for Parallel Matrix Factorization , PDCS'98 10th International Conference on Parallel and Distributed Computing and Systems, IASTED, Las Vegas, Oct 1998, pages 291-297.
16) F. Gustavson: Recursion Leads to Automatic Variable Blocking for Dense Linear-Algebra Algorithms, IBM Journal of Research and Development, Vol. 41, No. 6, pp. 737-755, 1997.
17) Olivier Beaumont, Vincent Boudet, Arnaud Legrand, Fabrice Rastello and Yves Robert: Heterogeneity Considered Harmful to Algorithm Designers, Report LIP RR-2000-24, June 2000.
18) Vincent Boudet, Antoine Petitet, Fabrice Rastello and Yves Robert: Data Allocation Strategies for Dense Linear Algebra Kernels on Heterogeneous Two-dimensional Grids, Report LIP RR-99-31, June 1999.