ヘテロなクラスタ環境におけるNAS Parallel Benchmarksの適用化

笹生　健[†]　松岡　聡[††]

本研究ではヘテロな計算環境でのアルゴリズム研究を目的として，NAS Parallel Benchmarks のCG,EP,FT,IS,MG の 5 種類のカーネルベンチマークについて，通信の発生する頻度，データ分割法などの点から解析・分類した．そして，CPU ヘテロなクラスタ上での負荷分散手法として，多次元配列データをブロック分割しているアルゴリズムについては，各 PE の性能に応じて割り当てるブロックの個数を変えるという手法を用い，1 次元配列データを分割するアルゴリズムについては，各 PE の性能に応じて分割の幅を変えるという手法を用いて実装し，CPU 性能のみヘテロなクラスタ環境においてヘテロ対応手法の評価実験を行った．その結果，EP，IS，MG では性能向上が見られたものの，CG，FT では通信のオーバーヘッドの増大により逆に性能低下を招く結果となった．

An Efficient NAS Parallel Benchmarks Algorithm for Heterogeneous Clusters

Takeru Sasou [†] and Satoshi Matsuoka [††]

In this study, we implemented the optimization of the Kernel Benchmarks of NAS Parallel Benchmarks for a heterogeneous cluster system and evaluated on the CPU heterogeneous cluster. We used the technique of optimization that load sharing by changing data size corresponding to a performance of each nodes. From the experimental results, our method achieves improvement of performance on EP, IS, and MG. But in the case of CG and FT, increase of a communicative overhead affects the performance, and the performance of our method less than original CG and FT.

1. はじめに

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．

そこで本研究では，NPB(NAS Parallel Benchmarks)[1] の CG,EP,FT,IS,MG の 5 種類のカーネルベンチマークを CPU 性能ヘテロなクラスタ環境向けに最適な負荷分散を取る手法を適用し，実装した．CPU ヘテロなクラスタ向けの負荷分散手法として，多次元配列データをブロック分割しているアルゴリズムについては，各 PE の性能に応じて割り当てるブロックの個数を変えるという手法を用い，1 次元配列データを分割するアルゴリズムについては，各 PE の性能に応じて分割の幅を変えるという手法を用いた．

そして CPU 性能のみヘテロなクラスタ環境においてヘテロ対応手法の評価実験を行い，その有効性について検証を行った．その結果，各 PE の性能に応じて 1 次元配列データの分割幅を変える手法は性能向上が得られたが，割り当てるブロックの個数を変えるという手法については，逆に通信部分のオーバーヘッドが増大してしまい，性能の低下を招く結果となった．

† 東京工業大学
Tokyo Institute of Technology
†† 東京工業大学 学術国際情報センター
Tokyo Institute of Technology, GSIC

2. CPU ヘテロな環境への負荷分散手法

CPU の異なるノードが混在する CPU ヘテロなクラスタにおいては，各 CPU に同量の負荷をかけると遅い CPU の処理が済むまで他の CPU が待たなければならないため，全体の実行効率が最も遅い CPU に依存してしまう．そこで，速い CPU にたくさん負荷をかけて遅い CPU にかける負荷を減らして負荷分散のバランスを取る事で，全体の実行効率を向上させる事ができる．しかし，全体の処理のうち，通信がドミナントなアルゴリズムでは逐次処理の部分で負荷分散を取って実行効率を上げても，全体の実行時間にあまり反映されないばかりか，CPU 性能での負荷分散によって各 PE 間のメッセージサイズに偏りが生じ，通信部分のオーバーヘッドの増加により，逆に全体の実行性能が悪化するといった問題も考えられる．

本研究では，各ベンチマークを CPU ヘテロな環境に適用させるために，各 PE の演算性能に応じて割り当てるデータサイズを変えることで負荷分散を取る．その際，各ベンチマークアルゴリズム毎にデータ分割手法が異なるので，それに応じて割り当てるデータサイズを変える手法も異なる．NPB の各種カーネルベンチマークのデータ分割手法は大きく分けて，1 次元配列データの分割をするアルゴリズムと多次元配列をブロック分割するアルゴリズムに分けられる．それぞれに対する負荷分散は以下の様に行う．

- **1 次元配列の分割 (EP,IS)**
  各 PE の演算性能に応じて，配列の分割幅を変える．
- **多次元配列のブロック分割によるデータ分割 (CG,FT,MG)**
  各 PE の演算性能に応じて，割り当てるブロックの個数を変える．

各ベンチマークの CPU ヘテロへの適用化実装については 3 章にてその概要を述べる．

3. NPB のヘテロ対応化

3.1 EP

EP[2] はガウス分布の疑似乱数を生成するアルゴリズムであり，モンテカルロ法を用いた応用プログラムによく見られる．EP のコードはほぼ完全な並列化が行われている．EP では問題クラス毎に指定された n 個点からなる探索空間を分割して割り当てる事で並列化を行っている．ある PE に割り当てられた点の個数を n_p とすると，その PE の計算量は $2n_p$ となるので，各 PE の処理性能に応じて割り当てる点の個数 n_p を変えることで，全体の負荷分散が取れる．

ヘテロ対応化 EP では，N_p 個の PE に対して n 個点からなる探索空間を分割する際に，$P_i(i = 1, 2, ..., N_p)$ の性能を持つ PE_i に対して割り当てる点の個数 n_i を次のようにして求める．

(1) $nbase_i = \frac{n}{\sum_{k=1}^{N_p} P_k} \times P_i$ を求める
(2) $surplus1 = \mathtt{mod}(n, \sum_{k=1}^{N_p} P_k)$ を求める
(3) $nbase_i = nbase_i + \frac{surplus1}{N_p}$ とする．
(4) $surplus2 = \mathtt{mod}(surplus1, N_p)$ を求める．
(5) もし $P_k(k = 1, 2, ..., N_p)$ を降順にソートしたとき，P_i が先頭から $surplus2$ 番目以内にある場合，$nbase_i$ に 1 加える

上記の方法により，各 PE に割り当てられる点の個数の差は最大 1 に押さえられ，且つ点の個数が 1 多い PE はより性能の高いものとなる．

3.2 IS

IS[2] は，バケツソートによる整数ソートを行うアルゴリズムである．IS の計算フェーズは大きく分けて，

Step1割り当てられたキーのローカルなバケツへの振り分け

Step2ローカルなバケツからグローバルなバケツへの完全交換

Step3グローバルなバケツ内のキーの個数の数え上げ

の 3 ステップから成り，このうち逐次の計算部分は Step1，Step3 である．各 PE の演算性能に応じて負荷分散を取る場合，Step1 では最初に割り当てるキーの個数を PE の性能に応じて変える事で Step1 の負荷分散を取る事ができ，Step3 では完全交換後に割り当てるキーの個数を変える事で Step3 の負荷分散を取る事ができる．

ヘテロ対応化 IS では，N_p 個の PE に対して N 個点からなるキーを分割する際に Step1 における性能 $P1_i(i = 1, 2, ..., N_p)$ と，Step3 における性能 $P3_i(i = 1, 2, ..., N_p)$ に基づいて，Step1 と Step3 におけるキーの割当数を変える．Step1 では PE_i に対して $\frac{N}{\sum_{k=1}^{N_p} P_k} \times P_i$ 個のキーを割り当てるようにする．Step3 では完全交換後に割り当てるバケツの個数を変える事で負荷分散を取る．各プロセスに割り当てるバケツの個数の決定は，PE_i に割り当てるバケツを決める時に $PE_0 - PE_i$ に割り当てられるバケツのキーの合計が

$$\frac{\sum_{j=1}^{N_p} P_j}{\sum_{k=1}^{N_p} P_k} \times N$$

を越えるまで PE_i にバケツを割り当てていくというようにすることで，各プロセスに割り当てられるキーの個数の比が PE の処理性能比に近くなるようにしている．

3.3 CG

CG[2] は Conjugate Gradient(共役勾配) 法[3] を用いて大規模疎行列の最小固有値を求めるアルゴリズムである．CG では係数行列を N_p 個の PE に対して $P \times Q = N_p$ となるようにブロック分割し，各 PE に分配している．処理の大部分は行列-ベクトル積，ベク

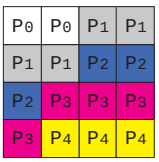

図 1 1 つの PE のブロックが複数の行にまたがる場合

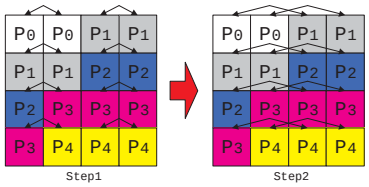

図 2 通信相手が複数の場合)

トル-ベクトル積であり，それぞれにおいて allreduce 通信が発生するため，逐次処理の部分はローカルに割り当てられた行列，ベクトルの演算になる．逐次処理の部分のうち，最も演算量が多いのは行列-ベクトル積なので，各 PE の演算性能に応じた負荷分散は行列-ベクトル積の部分の処理時間に応じて取る．

ヘテロ対応化 CG では，ブロック分割された係数行列の小ブロックを各 PE に対して割り当てる個数を変える事で，負荷分散を取る．この時，複数のブロックに PE を割り当てているため，allreduce 通信や行列転置通信の際の通信相手が複数プロセスになる場合が発生する．行列転置の 1 対 1 通信は，まず最初に MPI_IRECV を全て発行しておき，その後 MPI_SEND を全て実行するというように，行うべき送信要求と受信要求を全て一括して行う事で解決できる．

行列-ベクトル積，ベクトル-ベクトル積演算における集団通信はブロック行毎に行われるが，1 つの PE が複数のブロック行に属する場合，各行の集団通信を逐次に行うと他の PE で待ち時間が発生する．図 1 の様な場合，各 PE が自分に割り当てられたブロック行のうち，単純に上の行から処理しようとすると 1 行目から 4 行目まで逐次に処理することになってしまう．うまく並列に集団通信を行うにしても，ブロック割り当てによって条件が変わるので，スケジューリングが複雑になる．

そこで，今回の実装では全ブロック行における集団通信の足並みを揃えるようなスケジューリングを行っている．つまり，各 PE は，各通信ステップ毎に保持するブロックが行うべき通信を全て処理するようにし，図 2 のように全体で通信ステップを揃えるようにする．この場合，先ほど述べたような集団通信の並列生が損なわれる問題は起きないが，ブロック分割を細かくする事で，単純に通信ステップ数が増大する事による通信のオーバーヘッドの増加が予想される．

3.4 FT

FT[2] は，Partial Differential Equation を 1 回の 3 次元 FFT と数回の逆 3 次元 FFT を用いて解くアルゴリズムである．FT では，3 次元配列を z 軸方向に 1 次元ブロック分割，PE 数が z 軸の要素数を越える場合は y, z 平面上で 2 次元ブロック分割を行い，各 PE に割り当てている．FT で行われる 3 次元 FFT は，1 次元 FFT を x, y, z 軸方向に 1 回ずつの計 3 回行う事で実装されており，それぞれの 1 次元 FFT は各 PE で独立して実行できる．1 次元ブロック分割の場合は 2 回目の FFT と 3 回目の FFT の間で，2 次元ブロック分割の場合は 1 回目の FFT と 2 回目の FFT の間と，2 回目の FFT と 3 回目の FFT の間で配列の完全交換が行われる．FFT の処理中はこの配列の完全交換と 3 次元 FFT 後に行われるチェックサム計算時の allreduce 通信においてのみ通信が発生する．

表 1 V-Cycle マルチグリッド法

z_k	=	$M^k r_k$:	
	if $k > 1$		
		$r_{k-1} = Pr_k$	(restrict residual)
		$z_{k-1} = M^{k-1} r_{k-1}$	(recursive solve)
		$z_k = Qz_{k-1}$	(prolongate)
		$r_k = r_k - Az_k$	(evaluate residual)
		$z_k = z_k + Sr_k$	(apply smoother)
	else		
		$z_1 = Sr_1$	(apply smoother)

FT の逐次処理は大部分が 1 次元 FFT あり，この部分における処理性能を基に負荷分散を行うべきである．1 回目，2 回目，3 回目の各 FFT における演算量のオーダーは各 PE に割り当てられた n_x, n_y, n_z のブロックに対してそれぞれ $n_y \times n_z \frac{n_x \log n_x}{2}$，$n_x \times n_z \frac{n_y \log n_y}{2}$，$n_x \times n_y \frac{n_z \log n_z}{2}$ なので，それぞれの FFT において各 PE の演算量の比が各 PE の演算性能 PE_i の比になるようにすれば，最適な負荷分散が達成できる．

ヘテロ対応化 FT では，ブロック分割された 3 次元配列のブロックを，各 PE に対して割り当てる個数と変える事で，負荷分散を取る．配列の完全交換通信は FT では MPI_alltoall で実装されているが，ヘテロ対応化 FT では割り当てられたブロック数が各 PE 毎に異なるので，担当する全てのブロック分の MPI_IRECV を発行しておき，その後 MPI_SEND を全てのブロック分行うという様に実装している．

3.5 MG

MG[2] は，V-Cycle マルチグリッド法[4] を用いてポアソン方程式を解くアルゴリズムである． MG では問題領域となる格子をなるべく立方体に近くなるように 3 次元ブロック分割して各 PE に割り当てている．MG の処理は表 1 の restrict residual，prolongate，evaluate residual，apply smoother の 4 つのフェーズに分けられ，それぞれのフェーズの最後において隣接ブロックを担当する PE との通信が発生するため，逐次処理の部分もこれらの 4 つのフェーズに分けられる．問題領域の格子のサイズを $N_n = n_x \times n_y \times n_z$，

加算処理を A，乗算処理を M とすると，それぞれの計算フェーズにおける演算量は次のようになる．

- restrict residual $\frac{20A+4M}{8} \times N_n$
- prolongate $\frac{16A+4M}{8} \times N_n$
- evaluate residual $(15A + 4M) \times N_n$
- apply smoother $(15A + 4M) \times N_n$

それぞれのフェーズでの演算量は $O(N_n)$ なので，各 PE の性能に応じて負荷分散を行う際は $n_x \times n_y \times n_z$ のサイズ比が各 PE の演算性能の比になるようにする事で，最適な負荷分散が行える．

ヘテロ対応化 MG では，3 次元ブロック分割された問題領域格子の各ブロックを各 PE に対して割り当てる個数を変更する事で負荷分散を取る．これにより，各 PE に対して割り当てられるデータサイズ N_n の比が各 PE の演算性能比になるので，最適な負荷分散が行える．この時，各計算フェーズで行われる通信の際，通信相手が同じプロセスに割り当てられたブロックとなる場合がある．これについては，境界部分の要素をメモリコピーする事で解決している．また，1 つの PE が複数のブロックを担当するため，1 つのフェーズにおける通信相手も複数になる場合があるが，これについては，担当する各ブロックにおける 6 方向への受信要求を最初にまとめて発行し，それからメッセージの送信を行うように実装している．

4. 評価実験

4.1 評価環境

評価環境として，松岡研究室の PrestoIII クラスタ[5](表 2) を 16 ノード使用した．??における評価実験と同様に，CPU を交換して 900MHz 動作させたノードを 1 ノード混ぜて実験を行った．なお，PrestoIII のノードは SMP マシンだが，ノード内通信とノード間通信のネットワーク性能が不均質になるのを避けるため 1 ノード 1 プロセスずつで実行し，ノード間接続には Myrinet2000 を用いた．以後，1.6GHz ノード，900MHz ノードをそれぞれノード P，Q と呼ぶことにする．

表 2 PrestoIII ノードスペック

CPU	AMD AthlonMP 1900+(1.6GHz)×2
Motherboard	Tyan TigerMP
Memory	768MB DDR SDRAM
Network Card 1	100Base-T (Auxiliary)
Network Card 2	Myricom Myrinet 2000

4.2 実験結果

CPU ヘテロなクラスタに対する負荷分散手法を適用したヘテロ対応化 NPB(Hetero NPB) とオリジナルの NPB(Original NPB) について CPU ヘテロなクラスタ上で比較実験を行った．問題サイズはクラス B を使用し，参考のために CPU 性能の低いノードを混ぜない，ノード P のみで構成された CPU 性能ホモな環境でのオリジナルの NPB の性能 (homo) も載せた．

4.2.1 EP

EP での負荷分散は，1 ノードでの EP 全体の実行時間を元にデータサイズの比を決定する．その結果，ノード P とノード Q 実行時間の逆比は 1.76 : 1 であったので，データサイズ比は 7 : 4 とした．

EP の実験結果を図 3，表 3 に示す．実験の結果，ヘテロ対応化 EP はオリジナル EP と比べ性能向上を達成しており，CPU 性能ホモな環境での性能と比べ，CPU 理論性能の低下分と同程度の性能である．

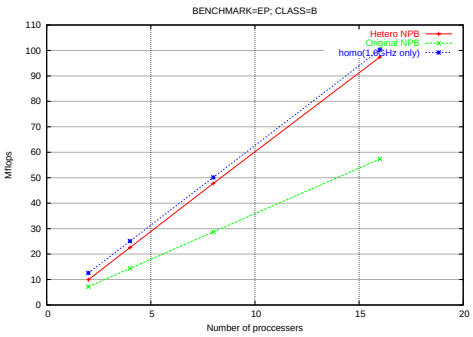

図 3 BENCHMARK=EP, CLASS=B

表 3 BENCHMARK=EP CLASS=B(Mflops)

#CPU	Hetero NPB	Original NPB	homo
2	9.93	7.17	12.57
4	22.58	14.34	25.10
8	47.78	28.71	50.14
16	97.40	57.35	100.26

4.2.2 IS

IS での負荷分散は Step1，Step3 それぞれ個別に行い，1 ノードでの実行時間比を元にデータサイズ比を決定する．Step1 での実行時間の逆比は $P : Q = 1.54 : 1$，Step3 での実行時間の逆比は $P : Q = 1.38 : 1$ であった．実験の際はデータサイズ比は Step1 では 3 : 2，Step3 では 7 : 5 とした．

IS の実験結果を図 4，表 4 に示す．EP 同様，ヘテロ対応化 IS はオリジナル IS と比べ大幅に性能向上を達成している．

表 4 BENCHMARK=IS CLASS=B(Mflops)

#CPU	Hetero NPB	Original NPB	homo
2	19.27	15.72	21.51
4	35.97	29.02	38.12
8	67.95	55.42	71.07
16	113.30	97.72	118.09

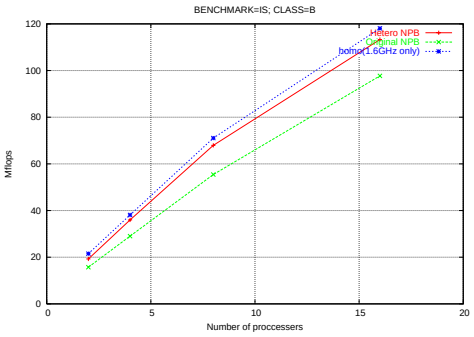

図 4 BENCHMARK=IS, CLASS=B

4.2.3 CG

CG での負荷分散は行列-ベクトル積演算の際の実行時間比を元に決定する．その結果，ノード P とノード Q 実行時間の逆比は 1.47 : 1 であった．ブロック割り当ての比は $P_0 : P_{1\sim15} : Q = 5 : 4 : 3$ とした．

CG の実験結果を図 5，表 5 に示す．ヘテロ対応化 CG は CPU 数が 8,16 の時に，オリジナルの CG と比べ大幅な性能低下が見られる．これは，ヘテロ対応化 CG は CPU 数が 8,16 の時には係数行列を 8×8 のブロックに分割しており，分割が細かくなったために通信回数の増加し，通信オーバーヘッドの増大した事が原因である．表 6 に最も通信回数の多い PE における，allreduce 通信における通信ステップ数と，ベクトル転置のための 1 対 1 通信回数の比較を示す．

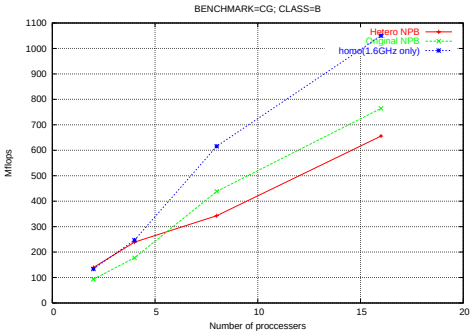

図 5 BENCHMARK=CG, CLASS=B

表 5 BENCHMARK=CG CLASS=B(Mflops)

#CPU	Hetero NPB	Original NPB	homo
2	139.34	93.27	133.54
4	239.08	177.69	247.05
8	342.82	438.42	615.62
16	655.76	764.40	1050.01

4.2.4 FT

FT の負荷分散は 1 次元 FFT の実行時間比を元に決定し，その逆比は $P : Q = 1.51 : 1$ であった．ブロック割り当ての比は $P_0 : P_{1\sim15} : Q = 5 : 4 : 3$ とした．

表 6 CG の通信回数の比較

#CPU		Hetero CG	Original CG
2	allreduce 通信	11850	5925
	ベクトル転置通信	1950	1950
4	allreduce 通信	23700	5925
	ベクトル転置通信	5850	1950
8	allreduce 通信	23700	11850
	ベクトル転置通信	17550	1950
16	allreduce 通信	29625	11850
	ベクトル転置通信	9750	1950

FT の実験結果を図 6，表 8 に示す．2 ノードで実行するとメモリのスワップが発生するため，4 ノード以上でのみ評価を行っている．ヘテロ対応化 FT はオリジナル FT より低い性能となっているが，これは配列の完全交換通信時のメッセージサイズの不均衡による通信のオーバーヘッド増大が最も大きな原因と考えられる．表 7 に示す通信時間の比較を見ると，ヘテロ対応化 FT はオリジナル FT に比べ 30%以上通信時間が増大している．ヘテロ対応化 FT における各 PE へ割り当てられたブロック数の平均は 4 であり，最も多くブロックを割り当てられた PE のブロック数は 5 なので，メッセージサイズの不均衡は 25%である．そのため，メッセージサイズの不均衡による通信のオーバーヘッド増大以外にも，ヘテロ対応化によるオーバーヘッドの増大が考えられるが，詳細については現在調査中である．

表 7 FT の通信時間の比較 (秒)

#CPU	Hetero CG	Original CG
4	49.51	45.78
8	36.30	27.23
16	18.29	13.41

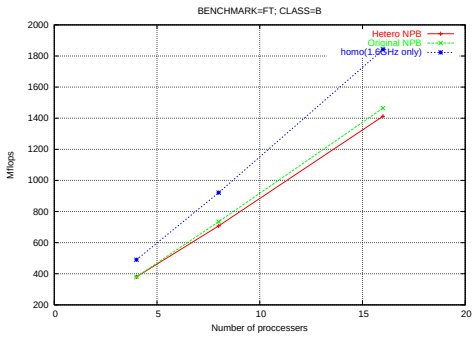

図 6 BENCHMARK=FT, CLASS=B

4.2.5 MG

MG の負荷分散は evaluate residual での実行時間比を元に決定し，その逆比は $P : Q = 1.35 : 1$ であった．

表 8　BENCHMARK=FT CLASS=B(Mflops)

#CPU	Hetero NPB	Original NPB	homo
4	380.24	380.35	490.28
8	708.06	734.75	921.19
16	1412.46	1464.88	1841.40

ブロック割り当ての比は $P_0 : P_{1\sim15} : Q = 5 : 4 : 3$ とした．

MG の実験結果を図 7，表 9 に示す．実験の結果，ヘテロ対応化 MG は 8CPU の場合を除いてオリジナル MG より性能が向上している．理論性能の点から見ると，16CPU において性能ホモな環境と比べて理論性能では 2.7%低下なのに対し，実行性能は 13.3%低下している．これは，単純にブロック割り当て数が増加したことによる通信回数の増大が原因と考えられる．表 10 に最も通信回数の多い PE の通信回数の比較を示す．なぜ 8CPU の時に大きく性能低下しているのかについては調査中である．

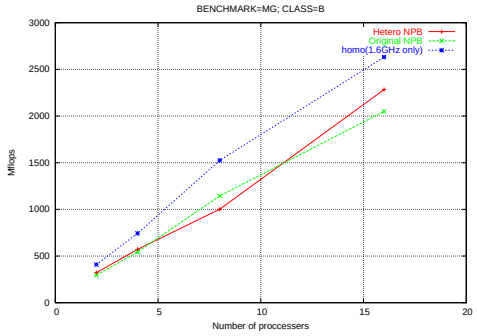

図 7　BENCHMARK=MG, CLASS=B

表 9　BENCHMARK=MG CLASS=B(Mflops)

#CPU	Hetero NPB	Original NPB	homo
2	321.50	295.03	408.28
4	571.72	543.42	743.02
8	1001.73	1144.52	1524.26
16	2283.54	2048.93	2632.27

表 10　MG の通信回数の比較

#CPU	Hetero CG	Original CG
2	4610	2766
4	5512	2766
8	6396	2766
16	8600	2766

5. 考　　察

EP と IS ではヘテロ向けの負荷分散を行ったアルゴリズムが高い性能を達成しており，負荷分散手法による効果が発揮されている．しかし，ブロック分割にデータ分割を行っているアルゴリズムでは，CG や FT で大きく性能が低下している．

CG はブロック分割数が増えるに従って，集団通信の通信ステップ数が増大する．そのため，今回取ったブロック割当数を変えるという手法では負荷分散を取るためにブロック分割数を増やさざるを得ないため，むしろ通信のオーバーヘッドが増大し逆効果であると考えられる．CG の様な集団通信が行われるアルゴリズムでは，各 PE に割り当てるデータサイズを変えたときに，通信ステップ数が増えにくい手法を取る必要があり，そういった手法の考案が今後の課題である．

MG では性能向上が見られたが，FT ではオリジナルのアルゴリズムを下回る結果となった．これは FT は配列の完全交換通信であるのに対し，MG ではブロックの隣接要素の交換のみで，且つ処理中でメッセージサイズが減少していくアルゴリズムであり，FT と MG ではメッセージサイズの不均衡による通信のオーバーヘッドへの影響が違うためである．CPU 性能ヘテロな環境に対して負荷分散を行う際は，メッセージサイズの不均衡が及ぼす影響について考慮が必要である．

6. まとめと今後の課題

本稿では NPB2.3 のカーネルベンチマーク 5 種を CPU ヘテロな環境向けに負荷分散を取るようアルゴリズムを変更して実装し，CPU 性能のみヘテロなクラスタ環境上で評価を行った．その結果，EP，IS,MG では性能向上が見られたものの，CG，FT では集団通信のステップ数の増大や通信時のメッセージサイズの不均衡により通信のオーバーヘッドが増大し，逆に性能低下を招く結果となった．そのため，今後は通信量が増大しないような手法の考案が今後の課題である．

参　考　文　献

1) The NAS Parallel Benchmarks, http://www.nas.nasa.gov/Software/NPB/
2) David Bailey,Tim Harris, William Saphir, Rob vander Wijngaart, Alex Woo, and Maurice Yarrow, “The NAS Parallel Benchmarks 2.0”, NASA Ames Research Center Report, NAS-05-020, 1995.
3) 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.
4) William L. Briggs, A Multigrid Tutorial, SIAM, 1978
5) http://cluster-team.is.titech.ac.jp/