ヘテロなクラスタ環境における Strassen の行列積アルゴリズムの並列化

大滝雄介[†] 高橋大介[††]
朴泰祐[††] 佐藤三久[††]

本論文では $n \times n$ 行列積の演算量が $O(n^{\log_2 7})$ である Strassen の行列積アルゴリズムをヘテロなクラスタ環境向けに並列化し，CPU ヘテロなクラスタ環境上で性能評価を行った．CPU ヘテロなクラスタ環境においては，各プロセッサの性能に応じた負荷分散を行うことが全体の性能向上のために必要不可欠である．一方，Strassen アルゴリズムは再帰的に計算することで，再帰 1 回あたり演算量が約 7/8 に減少するため，再帰回数が演算量に大きく影響する．したがって，負荷分散だけでなく再帰回数も考慮する必要がある．本論文では各プロセッサの CPU 性能のほかに，通信や Strassen の行列積アルゴリズム中における再帰回数を考慮して負荷分散を行うことにより，実行時間の最小化を図った．その結果，ヘテロ化を行っていない従来のアルゴリズムと比べ最大で約 20%の性能向上が得られた．

Implementation of Strassen's Matrix Multiplication Algorithm for Heterogeneous Clusters

Yuhsuke Ohtaki,[†] Daisuke Takahashi,[††] Taisuke Boku[††] and Mitsuhisa Sato[††]

In this paper, we evaluate the performance of Strassen's matrix multiplication algorithm in a heterogeneous clustering environment. In the heterogeneous clustering environment, an appropriate data distribution is the most important to achieve maximum performance as a whole. However, Strassen's algorithm reduces a total operation count to about 7/8 times per one recursion and hence recursion level has an effect on a total operation count. Thus, we need to consider not only load balancing but recursion level in Strassen's algorithm. In order to minimize execution time, we consider CPU performance, communication and recursion level in the Strassen's algorithm. As a result, we achieved nearly 20% speedup in a heterogeneous clustering environment compared to the conventional parallel Strassen's algorithm.

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を用いることによるコストパフォーマンスの良さとそのスケーラビリティという点で，近年，非常に注目されている並列計算機である．クラスタ型計算機は，複数の PC やワークステーションなどの汎用計算機をネットワークで接続したものであり，PC を多数並べてネットワークでつないだ PC クラスタが主流になっている．

クラスタ型計算機は，プロセッサを段階的に増設したり，グリッド環境上で複数のクラスタを使用したりする場合に，CPU やネットワーク，キャッシュなどのハードウェア性能が異なるノードが混在することになり，本質的にヘテロジニアス環境を含んでいる．各プロセッサのハードウェア性能がホモジニアスな環境においては，各プロセッサの演算性能が同等であることから，均等に負荷分散を行うだけでロードバランスを保て，比較的容易に高速化が実現する．しかし，それらの性能が異なるプロセッサが混在するヘテロジニアスなクラスタ環境においては，各プロセッサに均等に負荷分散を行ってしまうと，同期が起こるたびに性能の低いプロセッサが高いプロセッサの足を引っ張り，全体の処理効率が著しく低下する．したがって，ヘテロジニアスなクラスタ環境上で計算を行う場合は，各

† 筑波大学大学院理工学研究科
Master's Program in Science and Engineering, University of Tsukuba
†† 筑波大学電子・情報工学系
Institute of Information Sciences and Electronics, University of Tsukuba

プロセッサの演算性能を考慮した負荷分散を考えることが重要である．

Strassen の行列積アルゴリズム[1]は，n 次の行列積を $O(n^{\log_2 7})$ の演算量で行うことのできるアルゴリズムとして知られており，n が大きいときには非常に有効なアルゴリズムである．

ヘテロジニアス環境（以下，ヘテロ環境と呼ぶ）においては，笹生らが Linpack のベンチマークプログラムである HPL を，CPU 性能に応じて割り当てるデータサイズを変更するという手法を用いてヘテロ環境向けに最適化しており，ヘテロ環境でも高い性能が出ることが報告されている[2]．岸本らは，HPL について，CPU 性能に応じた数のプロセスを起動するマルチプロセス法によってロードバランスをとるという手法を用い，最適なプロセッサ構成およびマルチプロセス数を予測するモデルを構築している[3]．また，Beaumont らは，$O(n^3)$ の行列積について，ロードバランスを取りつつ，通信量を最小化する問題をヒューリスティックな手法で解き，通信時間を削減することに成功しており[4]，Dovolnov らはヘテロ環境におけるデータ分割方法を提案している[5]．

Strassen の行列積アルゴリズムの並列化も試みられており，並列計算機上においても通常の行列積アルゴリズムと比べ高速に計算できることが知られている[6),7]．

しかし，ヘテロ環境における Strassen アルゴリズムの研究は，まだ十分に行われていないのが現状である．また，再帰的なアルゴリズムである Strassen アルゴリズムは，その再帰回数が演算回数に大きく影響するため，ロードバランスをとるという手法のみでは，各プロセッサの足並みが揃ったとしても演算回数の増加が原因で結果的に性能が低下する場合があり，演算回数も考慮した最適化手法が必要である．

そこで本論文では，Strassen アルゴリズムにおける演算回数，各プロセッサの CPU 性能，そして通信量を考慮することにより，実行時間の最小化を図り，ヘテロ環境向けの並列化を行う．また，CPU ヘテロなクラスタ環境上で性能評価を行う．

2. Strassen の行列積アルゴリズム

$n \times n$ 行列積の演算量は通常の方法では $O(n^3)$ であるが，以下に示す Strassen の行列積アルゴリズム[1]（以下，Strassen アルゴリズムと呼ぶ）を用いると，$O(n^{\log_2 7})$ で計算可能である．Strassen アルゴリズムでは，n 次正方行列 A，B，C について，$C = AB$ を以下のような 2×2 の行列どうしの積として考える．

$$\begin{pmatrix} C_{11} & C_{12} \\ C_{21} & C_{22} \end{pmatrix} = \begin{pmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{pmatrix} \begin{pmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{pmatrix}$$

通常，2×2 の行列どうしの積は 8 回の乗算と 4 回の加算が必要となるが，Strassen はこれを 7 回の乗算と 18 回の加算で計算できることを示した[1]．n 次正方行列どうしの乗算は $O(n^3)$，加算は $O(n^2)$ の演算量であるから，n が大きくなれば乗算回数の少ない Strassen アルゴリズムの方が演算回数が少なくなる．本論文では，Winograd が加算を 18 回から 15 回に削減した以下の Winograd variation[8] を用いる．

$$\begin{array}{lll} S_1 = A_{21} + A_{22} & P_1 = S_2 S_6 & T_1 = P_1 + P_2 \\ S_2 = S_1 - A_{11} & P_2 = A_{11} B_{11} & T_2 = T_1 + P_4 \\ S_3 = A_{11} - A_{21} & P_3 = A_{12} B_{21} & T_3 = P_5 + P_6 \\ S_4 = A_{12} - S_2 & P_4 = S_3 S_7 & \\ S_5 = B_{12} - B_{11} & P_5 = S_1 S_5 & C_{11} = P_3 + P_2 \\ S_6 = B_{22} - S_5 & P_6 = S_4 B_{22} & C_{12} = T_1 + T_3 \\ S_7 = B_{22} - B_{12} & P_7 = A_{22} S_8 & C_{21} = T_2 - P_7 \\ S_8 = S_6 - B_{21} & & C_{22} = T_2 + P_5 \end{array}$$

$P_1 \sim P_7$ の乗算には Strassen アルゴリズムを再帰的に適用していくことが可能であり，行列サイズが n_0 になるまで Strassen アルゴリズムを適用した場合の演算回数を $T_s(n)$ とおくと，

$$\begin{aligned} T_s(n) &= 7T_s\left(\frac{n}{2}\right) + 15\left(\frac{n}{2}\right)^2 \\ &\approx c \cdot n^{\log_2 7} \end{aligned} \quad (1)$$

となる．ただし，$c = (2n_0^3 + 5n_0^2)/n_0^{\log_2 7}$ である．

また，$n = 32$ のとき，通常の行列積アルゴリズムの演算回数は $2{\cdot}32^3$ である．一方，Strassen アルゴリズムを 1 回適用した場合の演算回数は，$7{\cdot}2{\cdot}16^3 + 15{\cdot}16^2 \approx 1.87 \cdot 32^3$ となるから，$n \geq 32$ ならば Strassen アルゴリズムを 1 回適用した場合に，通常の行列積と比べ演算回数が少なくなる．しかし，Strassen アルゴリズム中における変数の添字処理や再帰呼び出し，加算のオーバヘッドなどのため，n をさらに大きくしなければ通常の行列積より速くはならない．gcc 2.96 でコンパイルし，BLAS に ATLAS 3.4.1 を用いて最適化した $O(n^3)$ の行列積と Strassen アルゴリズムを，Xeon 2.4 GHz 上で実行した場合の実行時間を**図 1** に示す．Strassen アルゴリズムが通常の行列積よりも速くなるのは，$n \geq 1760$ のときであった．

3. 従来の並列行列積アルゴリズム

3.1 Strassen アルゴリズムの並列化

Fox らによる BMR（Broadcasting Multiply Roll）法[9] において，各プロセッサが行列積を計算する部分に Strassen アルゴリズムを適用することで Strassen アルゴリズムを並列化できる[6]．ここでは，その手法

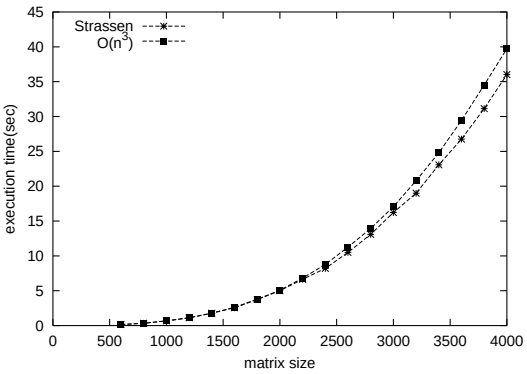


図 1 Strassen アルゴリズムと通常の行列積アルゴリズムの実行時間
Fig. 1 Execution times of Strassen's algorithm and traditional algorithm.

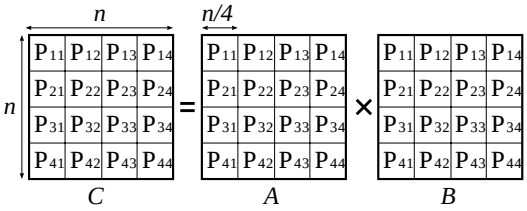


図 2 プロセッサのマッピング例
Fig. 2 An example of data distribution.

を述べる．ここで，プロセッサ台数を p とする．

BMR 法では，行列 A，B，C を**図 2** のように $\sqrt{p} \times \sqrt{p}$ のメッシュに分割し，各プロセッサに割り当てる．このとき，A_{ij}，B_{ij}，C_{ij} が割り当てられているプロセッサを $P_{ij}(1 \leq i, j \leq \sqrt{p})$ と呼ぶことにすると，P_{ij} は，

$$C_{ij} = A_{i1}B_{1j} + A_{i2}B_{2j} + \cdots + A_{i\sqrt{p}}B_{\sqrt{p}j}$$

を計算することになる．

まずはじめに，プロセッサ P_{i1} が A_{i1} を行方向に，プロセッサ P_{1j} が B_{1j} を列方向に broadcast する．**図 3** にその例を示す．この通信により，各プロセッサが $A_{i1}B_{1j}$ を計算することができる．次にプロセッサ P_{i2} とプロセッサ P_{2j} が同様の broadcast を行えば，$A_{i2}B_{2j}$ が計算できる．つまり，プロセッサ P_{ik} が行方向に A_{ik} を，プロセッサ P_{kj} が B_{kj} を列方向に broadcast してから $A_{ik}B_{kj}$ を計算するというステップを $k = 1, 2, \cdots, \sqrt{p}$ として繰り返すことにより，各プロセッサが C_{ij} を計算することができる．図 3 は，$p = 16$ で $k = 1$ の場合の broadcast の例である．**図 4** に BMR 法による並列行列積アルゴリズムを示す．

各ステップで現れる行列積には Strassen アルゴリズムを適用することができ，これによって Strassen アルゴリズムは並列化される．BMR 法では，行列は $\sqrt{p} \times \sqrt{p}$ 個のメッシュに分割されるので，各プロセッサはサイズが $n/\sqrt{p}$ の行列積を $\sqrt{p}$ 回計算する．したがって，全体の演算回数を $T_{\mathrm{comp}}(n, p)$ とおくと，

$$\begin{aligned} T_{\mathrm{comp}}(n,p) &= p \cdot \sqrt{p} \cdot T_s\left(\frac{n}{\sqrt{p}}\right) \\ &= \sqrt{p}^{\log_2 \frac{8}{7}} \cdot T_s(n) \end{aligned} \tag{2}$$

となる．このことから，並列 Strassen アルゴリズムでは行列を $\sqrt{p} \times \sqrt{p}$ 個に分割すると，逐次で計算する場合と比べ，演算回数が $\sqrt{p}^{\log_2(8/7)}$ 倍になり，細かく分割すればするほど演算回数が増えてしまうことが分かる．また，通信量については $(\sqrt{p}-1)(n/\sqrt{p})^2$ 個の変数が行列 A，B についてそれぞれ $\sqrt{p}$ 回通信されるので，通信量を $T_{\mathrm{comm}}(n, p)$ とおくと，

$$\begin{aligned} T_{\mathrm{comm}}(n,p) &= 2\sqrt{p} \cdot (\sqrt{p}-1)\left(\frac{n}{\sqrt{p}}\right)^2 \\ &= \frac{2(\sqrt{p}-1)}{\sqrt{p}} \cdot n^2 \end{aligned} \tag{3}$$

である．

また，行列を分割せずに Strassen アルゴリズム自体を並列化する手法も提案されている[6]．しかし，通信量が $O(n^{\log_2 7})$ となるため，n が大きくなるにつれ BMR 法より通信量が多くなることが分かる．した

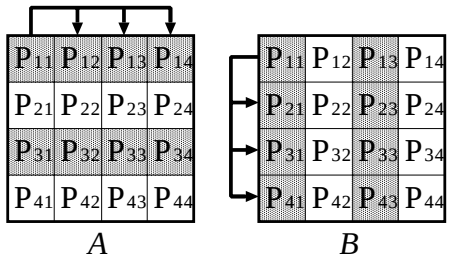


図 3 broadcast の例（$k = 1$）
Fig. 3 An example of broadcast ($k = 1$).

```
procedure BMR_Parallel_MM
k := 1;
begin
  while k <= √p do begin
    プロセッサ P_ik が A_ik を行方向に，プロセッサ P_kj
    が B_kj を列方向に broadcast する．
    C_ij = C_ij + A_ik B_kj
    k := k + 1;
  end
end;
```

図 4 BMR 法による並列行列積アルゴリズム
Fig. 4 BMR algorithm.

がって，演算速度と通信速度に大きな差がある現在のクラスタ環境には適用しても効果はないと考えられる．

3.2 ヘテロ環境での問題点

並列行列積アルゴリズムでは，各プロセッサに均等にデータを割り当てるため，ホモジニアス環境においてはロードバランスをとることができる．しかし，各プロセッサの演算性能が異なるヘテロ環境では，演算性能の高いプロセッサが演算性能の低いプロセッサの計算が終了するのを待ってから broadcast が行われるため，性能の高いプロセッサに待ち時間が発生してしまい，全体の性能が著しく低下する．たとえば，Xeon 2.4 GHz×7 + Athlon MP 1.53 GHz× 1 という環境で並列行列積を計算すると，2.4 GHz のプロセッサは，1.53 GHz のプロセッサが計算を終了するのを待つことになり，この時間の分だけ CPU 資源を無駄にすることになると考えられる．

3.3 ヘテロ化された行列積アルゴリズム

3.3.1 ロードバランシング

ヘテロ性には CPU 性能，ネットワーク性能，メモリ容量など，さまざまなものが考えられ，各プロセッサの性能に応じた負荷分散が必要である．3.2 節で示したように，ヘテロ環境において並列行列積アルゴリズムの性能が低下する原因は，性能の低いプロセッサが性能の高いプロセッサの足を引っ張っていることにある．したがって，演算性能に応じた負荷分散を行うことによって，ロードバランスをとれるようにすれば良いと考えられる．

BMR 法を用いた並列行列積アルゴリズムでは，各プロセッサに等しい個数の行列を割り当てていたが，これを改良し，性能の高いプロセッサに低いプロセッサよりも多くの行列を割り当てればロードバランスをとることができる．たとえば，8 台のプロセッサ P_0〜P_7 で構成されるクラスタ環境があり，各プロセッサの演算性能比が P_0〜$P_3 : P_4$〜$P_7 = 7 : 9$ であるとする．このようなヘテロ環境上で行列積を計算する場合，**図 5** のような割当てを行えば，演算性能に応じて負荷分散がされるので，ロードバランスがうまくとれ，演算性能の高い P_4〜P_7 が他のプロセッサを待つ時間はなくなる．演算性能に応じた負荷分散を行う手法は，行列積に限らず全体の実行時間の多くが演算時間であるアルゴリズムではよく用いられ，naive な実装方法である．

3.3.2 単一サイズ分割の問題点

図 5 のように，$g \times g$ 個の均一な小行列に分割して各プロセッサに割り当てる方法を単一サイズ分割と呼ぶことにする．図 5 について，各プロセッサが逐次で

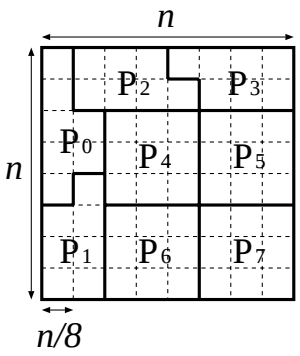


図 5 演算性能のみを考慮した分割の例
Fig. 5 An example of heterogeneous data distribution proportional to computational performances.

行列積を計算する部分に Strassen アルゴリズムを適用することを考えると，式 (2) より演算回数 $T_1(n)$ は

$$T_1(n) = 8^{\log 2^{(8/7)}} \cdot T_s(n) = \left(\frac{8}{7}\right)^3 \cdot T_s(n) \quad (4)$$

となる．$T_s(n)$ は逐次の Strassen アルゴリズムの演算回数であり，$(8/7)^3 \approx 1.49$ であるから，図 5 のような負荷分散を行って計算すると，逐次で計算する場合に比べ，演算量がかなり増えることが分かる．これは行列を細かく分割しているため，Strassen アルゴリズムによる演算回数の削減量が減少していることが原因である．また，通信量は通信の方法を工夫することによって削減できる可能性があるので一概にはいえないが，行列を $g \times g$ 個に分割した場合の通信量は g に比例して多くなるため，図 5 のような負荷分散では通信のオーバヘッドも大きくなることが予想される．一方，一般に行列積では図 5 のように行列を細かく分割した方が，分割された行列の個数が多くなるので，各プロセッサの演算性能に応じた負荷分散が行いやすい．

したがって，Strassen アルゴリズムをヘテロ環境向けに並列化する場合は，ロードバランスをとることと演算回数および通信量を減少させることがトレードオフになり，ロードバランスをとるだけで全体の性能が向上するとは限らない．また，演算性能について最も高いプロセッサと最も低いプロセッサとの差が小さい環境（以下，ヘテロ性が小さい環境と呼ぶ）において性能を向上させるためには，より細かなロードバランシングが必要不可欠であるが，このような環境上で行列を細かく分割して負荷分散を行うと，演算回数および通信量の増加により逆に性能が低下する可能性がある．

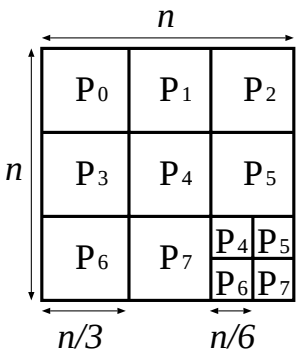


図 6 再帰回数を考慮した分割の例
Fig. 6 An example of recursive data decomposition.

4. 提案する並列 Strasssen アルゴリズム

4.1 再帰的分割

Strassen アルゴリズムをヘテロ環境向けに最適化する場合は，ロードバランス，再帰回数，通信量をすべて考慮して，最も性能が高くなる負荷分散を決定する必要があり，また再帰回数を減少させずに負荷分散を柔軟に行える分割方法が必要である．

そこで，本論文では行列を単一サイズに分割するのではなく，サイズの異なる行列に再帰的に分割する手法を提案する．この手法は行列を細かく分割することを避けることにより，行列サイズを大きく保ち，Strassen アルゴリズムによる演算回数の削減量を増加させると同時に，分割した行列の一部分をさらに細かく分割することによって，ロードバランスをとりやすくするためのものである．3.3.2 項であげた CPU ヘテロなクラスタ環境に対する分割方法を**図 6** に示す．図 6 では，行列を 3^2 個の小行列に分割し，各プロセッサに 1 個ずつ割り当て，余った 1 個をさらに 2^2 個の行列に分割し，演算性能が高い P_4〜P_7 に割り当てている．この手法により，再帰回数を減少させずに大部分の小行列の積を計算でき，一部の小行列をさらに細かく分割することにより，各プロセッサへの負荷の調整を柔軟にできると考えられるが，サイズが $n/6$ である右下の部分自体が並列行列積であり，新たに通信が発生するという欠点がある．図 6 では，各プロセッサがサイズ $n/3$ の行列積を 1 回計算し，P_4〜P_7 はさらにサイズ $n/3$ の行列積 1 回を並列に計算し，これを 1 サイクルとして 3 回繰り返す．P_4〜P_7 が担当する $n/3$ の並列行列積では，各プロセッサがサイズ $n/6$ の行列積を 2 回計算するので，総演算回数 $T_2(n)$ は，

$$\begin{aligned} T_2(n) &= 4 \cdot 3T_s\left(\frac{n}{3}\right) \\ &\quad +4 \cdot 3\left\{T_s\left(\frac{n}{3}\right) + 2T_s\left(\frac{n}{6}\right)\right\} \\ &\quad \approx 1.255T_s(n) < T_1(n) \end{aligned} \tag{5}$$

となり，演算回数は図 5 の方法より少ないことが分かる．したがって，複数の異なる行列サイズに分割する本手法は，分割された行列の中にサイズの大きいものが含まれるので，演算回数が少なくなり，場合によっては，単一サイズに分割する手法よりも有効である可能性がある．本論文では，行列サイズの種類を最大で 2 種類まで考えることとし，2 種類以上の大きさの小行列に分割する手法を再帰的分割と呼ぶことにする．

Strassen アルゴリズムをヘテロ環境向けに最適化するとき，以下の 3 つの方法が考えられ，個々のヘテロ環境に最適な方法を選択する必要がある．

- 単一サイズ分割を用いる方法
  ヘテロ性が大きい環境では，負荷分散を厳密に行わなくても十分な性能向上が得られる場合も考えられ，単一サイズの分割で十分な場合がある．
- 再帰的分割を用いる方法
  ヘテロ性が小さい環境など，負荷分散を細かくしなければ性能向上を見込めない場合に有効だと考えられる．
- 均一に負荷をかける方法
  ヘテロ性が非常に小さい環境では，演算性能に応じた負荷分散によって削減できる演算時間よりも通信のバランスが崩れることや通信量そのものの増加による通信時間の増加量が大きくなり，単一サイズ分割や再帰的分割を適用しても逆に性能が悪化することがある．このような場合は均一に負荷をかける従来の方法を変更するべきではなく，最適化が困難なケースであるといえる．

4.2 最適化手法

ここでは，演算時間・通信時間を含めた総実行時間を予測することにより，最適な分割方法を決定するアルゴリズムについて述べる．ある N について，$N \times N$ の行列を通信するのにかかる時間 T_c およびプロセッサ i が $N \times N$ の行列積を 1 回実行するのに要する時間 $T_{\mathrm{mul}}(i)$ がすでに分かっているとき，プロセッサ i が $n \times n$ 行列積を Strassen アルゴリズムを用いて計算するのに必要な時間は，Strassen アルゴリズムが $O(n^{\log_2 7})$ であることを用いると，$T_{\mathrm{mul}}(i) \cdot 7^{\log_2(n/N)}$ と推定することができる．同様にして，$n \times n$ の行列を通信するのに必要な時間は，$T_c \cdot 4^{\log_2(n/N)}$ と予測できる．本論文では，T_c および $T_{\mathrm{mul}}(i)$ を入力として評価関数に与え，総実行時間の予測を行う．

4.2.1 g^2 個の小行列を演算性能に応じて割り当てるアルゴリズム

行列を $g \times g$ 個に分割してできた小行列を p 個のプロセッサに割り当てるとき，ヘテロ環境では最も演算に時間のかかるプロセッサを他のプロセッサが待つことになる．したがって，各プロセッサの演算時間の最大値が最小になるように割当てを行えばよい．このアルゴリズムを**図 7** に示す．図 7 において，$T_{\mathrm{comp}}(i)$ はプロセッサ i が小行列 1 個の計算を担当した場合に必要な演算時間であり，小行列 1 個を担当したプロセッサがサイズ n/g の行列積を g 回計算することおよび $T_{\mathrm{mul}}(i)$ を用いて求めている．$T(i)$ はプロセッサ i の予測演算時間であり，$T_{\mathrm{next}}(i)$ は仮に割当てが 1 個増えた場合のプロセッサ i の演算時間である．つまり，$T_{\mathrm{next}}(i) = T(i) + T_{\mathrm{comp}}(i)$ である．また，$s(i)$ は各プロセッサの小行列の割当て個数である．

このアルゴリズムでは，$s(i)$ および $T(i)$ を 0 として開始し，現在の各プロセッサの演算時間 $T(i)$ から，仮に割当てが 1 個増えた場合の演算時間 $T_{\mathrm{next}}(i)$ を求め，$T_{\mathrm{next}}(i)$ が最小であるプロセッサの割当てを 1 個増やす処理を $g \times g$ 回繰り返して負荷分散を行う．一例として，演算性能比が $\{P_0 : P_1 : P_2 : P_3\} = \{1 : 2 : 3 : 4\}$ となっている CPU ヘテロな環境に対し，$g^2 = 9$ としてこの処理を適用した結果を**表 1** に示す．ただし，演算性能比から，行列積 1 回あたりの演算時間は $T_{\mathrm{comp}} = \{40, 30, 20, 10\}$ となっているものとする．また，表 1 において，第 j ステップは j 個目の小行列を割り当てるステップである．$j = 1$，すなわち 1 個目の小行列を割り当てる第 1 ステップの開始時では，まだ割当てが行われていないので，$s(i) = 0$ $(0 \le i \le 3)$ である．$T_{\mathrm{next}}(i)$ $(0 \le i \le 3)$ は小行列の割当てが 1 個増えた場合の演算時間を示す．この時点では，$T_{\mathrm{next}}(3)$ が最小であるから，プロセッサ 3 の割当て個数 $s(3)$ を増やし，$T(3)$ を更新する．太字で示した部分は各ステップにおいて行列の割当てが 1 個増えた場合の演算時間 $T_{\mathrm{next}}(i)$ $(0 \le i \le 3)$ が最小になるプロセッサを示す．第 2 ステップ（$j = 2$）では，プロセッサ 3 にすでに小行列が 1 個割り当てられている状態となり，$T_{\mathrm{next}}(i)$，$s(i)$ $(0 \le i \le 3)$ は表に示したようになる．したがって，$T_{\mathrm{next}}(2)$ が最小となり，プロセッサ 2 の割当て個数 $s(2)$ を増やす．このアルゴリズムは，つねに各プロセッサの演算時間の最大値が最小となるように割り当てていくので，この処理を 9 回繰り返すことで演算性能に応じた負荷分散が得られる．行列の割当て個数は $\{P_0, P_1, P_2, P_3\} = \{1, 1, 2, 5\}$，予測演算時間 M は，第 9 ステップ（$j = 9$）において各プロセッサの演算時間 $T(i)$ の最大値である 50 となる．

```
procedure Algorithm1
begin
  T_comp(i) := T_mul(i) · 7^(log2(n/gN)) · g;  (0 ≤ i ≤ p − 1)
  for j = 1 to g^2
    T_next(i) が最小になるプロセッサ i を探す.
    s(i) をインクリメントし，T(i) を更新する.
  end for
  M = max(T(0), ··· , T(p − 1))
end;
```

図 7 g^2 個の小行列を演算性能に応じて割り当てるアルゴリズム
Fig. 7 Optimal data distribution algorithm for g^2 blocks over p processors.

表 1 演算性能比が $\{P_0 : P_1 : P_2 : P_3\} = \{1 : 2 : 3 : 4\}$ となるヘテロ環境に図 7 のアルゴリズムを適用した結果
Table 1 The result of Algorithm1 when relative performaces of processors are $\{P_0 : P_1 : P_2 : P_3\} = \{1 : 2 : 3 : 4\}$.

P_i \ j		1	2	3	4	5	6	7	8	9
P_0	$T_{\mathrm{next}}(0)$	40	40	40	40	40	**40**	80	80	80
	$s(0)$	0	0	0	0	0	**1**	1	1	1
	$T(0)$	0	0	0	0	0	**40**	40	40	40
P_1	$T_{\mathrm{next}}(1)$	30	30	30	**30**	60	60	60	60	60
	$s(1)$	0	0	0	**1**	1	1	1	1	1
	$T(1)$	0	0	0	**30**	30	30	30	30	30
P_2	$T_{\mathrm{next}}(2)$	20	**20**	40	40	40	40	**40**	60	60
	$s(2)$	0	**1**	1	1	1	1	**2**	2	2
	$T(2)$	0	**20**	20	20	20	20	**40**	40	40
P_3	$T_{\mathrm{next}}(3)$	**10**	20	**20**	30	**30**	40	40	**40**	**50**
	$s(3)$	**1**	1	**2**	2	**3**	3	3	**4**	**5**
	$T(3)$	**10**	10	**20**	20	**30**	30	30	**40**	**50**

4.2.2 単一サイズ分割における負荷分散アルゴリズム

Strassen アルゴリズムにおいては，ロードバランスと演算量・通信量がトレードオフの関係にあり，図 7 のアルゴリズムをある特定の行列分割個数 g^2 について適用しただけでは，最適な負荷分散にならないと考えられる．このため，行列分割個数を 1×1，2×2，$\cdots$，$G \times G$ と変化させながら通信時間を含めた全実行時間を予測し，それが最小となる負荷分散を求めることにする．$G \times G$ は行列の分割個数の上限値であり，行列サイズに応じて設定する．

単一サイズ分割において演算性能に応じて負荷分散を行うアルゴリズムを**図 8** に示す．このアルゴリズムは行列の分割個数 g^2 を $1 \times 1, 2 \times 2, \cdots, G \times G$ と増加させながら繰り返し，最適な分割個数 g_{opt}，予測実行時間 T_{opt} およびプロセッサ i における小行列の割当て個数 $s_{\mathrm{opt}}(i)$ を求める．

図 8 において，演算時間については図 6 のアルゴリズムを適用することにより，ある特定の g について

```
procedure Algorithm2
g := 1;
T_opt := ∞;
begin
  while g × g ≤ G × G do begin
    call procedure Algorithm1
    行列 A, B について，各プロセッサ i の通信回数を
    計算，その回数を a_i, b_i とおく．
    T_comm := max_i(a_i + b_i) · 4^log(n/gN) · T_c;
    M := M + T_comm;
    if M < T_opt
      s_opt(i)，T_opt，g_opt を更新する．
    g := g + 1;
  end
end;
```

図 8 単一サイズ分割における負荷分散アルゴリズム
Fig. 8 Data distribution algorithm which gives appropriate number of blocks.

最適化され，予測演算時間 M が求まる．

次に予測通信時間 T_{comm} を求める．通信については，ある特定の g について演算時間が最小化された時点での負荷分散より，どのプロセッサが何回送受信を行うか求めることができる．通信時間は通信が最も集中するプロセッサの通信時間が全体の通信時間となるため，その通信回数と通信 1 回あたりに必要な通信時間から全体の通信時間を求めることができ，通信 1 回あたりのコストは $N \times N$ の行列を通信するのにかかる時間 T_{c} から予測することができる．

演算時間および通信時間の予測値から全実行時間が予測され，この値が最も小さくなるような分割個数 g^2 を選択し，負荷分散を行う．

4.2.3 再帰的分割の適用

再帰的分割が必要なケースは，小行列の個数が少ないことなどにより，単一サイズ分割ではロードバランスがそれほどうまくとれない場合である．そこで，本論文では，単一サイズ分割を行った時点で，ロードインバランスがある閾値を上回った場合にのみ，再帰的分割を適用することにする．ロードインバランスの尺度は，$M = \max(T(0), \cdots, T(p-1))$, $m = \min(T(0), \cdots, T(p-1))$ とおき，M/m の値によって与えることにする．

再帰的分割では，M の値をもとに，最も足を引っ張っているプロセッサおよびそれに近い演算時間を必要とするプロセッサの割当てを 1 個減らすという処理を行い，減らしたことによって余った小行列をさらに分割して各プロセッサに割り当てる．図 6 において，サイズが $n/6$ である右下の部分がこれにあたる．以下，余った小行列をさらに分割してできた小行列を再分割した小行列と呼ぶことにする．

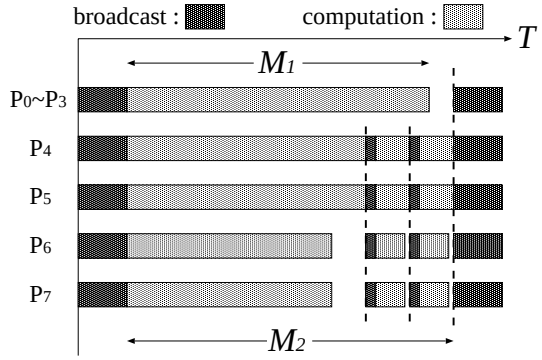


図 9 図 6 における各プロセッサの処理
Fig. 9 The process of each processor when we assign blocks like those in Fig. 6.

単一サイズ分割と再帰的分割の最も異なる点は，再分割した小行列が同期をともなって計算されることである．したがって，これを割り当てられたプロセッサの演算時間がどのように変化するかを考える必要がある．演算性能比が P_0〜$P_3 = 14$, $P_4 = P_5 = 18$, $P_6 = P_7 = 21$ であるようなヘテロ環境において，図 6 のような分割を行った場合の，最初の broadcast から次の broadcast までの各プロセッサの処理を**図 9** に示す．図 9 において，M_1 は P_0〜P_3 の演算時間の最大値，M_2 は P_4〜P_7 について，小行列の演算時間の最大値と，再分割した小行列の計算時間の最大値の和である．

図 9 では，P_0〜P_3 に関しては，小行列を計算し次の broadcast を待つことになる．P_4〜P_7 については各プロセッサは小行列の積をまず計算し，その後に再分割した小行列の積を並列に行う．再分割した小行列の計算は通信をしながら行われるので，再分割した小行列の計算に入る前に同期がとられ，再分割した小行列の計算中にも通信によって同期がとられる．したがって，P_4〜P_7 については，次の小行列の broadcast までにかかる時間は，小行列の演算時間の最大値と再分割したの小行列の計算時間（ただし，通信を含む）の最大値の和となることが分かる．したがって，次の broadcast までにかかる時間は

$$M' = \max(M_1, M_2)$$

として求められる．したがって，これが最小となるように小さい方の小行列の計算を担当させるプロセッサを決定すればよい．本論文では，M' が最小となるような分割を求め，各プロセッサに割り当てている．

5. 性能評価

5.1 ヘテロ性の定量化と性能向上率の上限値

ここで，性能評価の前にヘテロ性の大きさについての尺度を定義することにする．

演算性能比が $a_1 : a_2 : \cdots : a_p$ $(a_1 < a_2 < \cdots < a_n)$ となっているヘテロ環境があるとき，ヘテロ環境向けに最適化を行わない場合，演算性能が a_1 であるプロセッサがボトルネックとなり，演算性能が a_1 のプロセッサが p 台ある環境と演算性能は変わらない．したがって，ヘテロ環境向けの最適化を行ったことで，ロードインバランスが完全に解消されたと仮定すると，演算性能の向上率 s は，

$$s = \frac{a_1 + a_2 + \cdots + a_p}{a_1 \times p}$$

となる．これは最適化を行った場合と行わない場合の演算速度の比を表しており，この値が大きいほど最適化の効果が高い．s をヘテロ性の大きさとして定義し，この値と評価結果をもとに考察を行うことにする．

理想的なヘテロ化は，ヘテロ化によってロードインバランスが完全に解消され，かつ通信時間がまったく増えないことである．ヘテロ環境向けに最適化を行った場合の実行時間を T_{hetero}，最適化を行っていない場合の実行時間を T_{homo} とおく．T_{homo} における演算時間と通信時間の比が $1-t : t$ であるとすると，通信を含めた全処理の速度向上率は，

$$\frac{T_{\mathrm{homo}}}{T_{\mathrm{hetero}}} = \frac{(1-t)+t}{(1-t)/s+t} = \frac{1}{(1-t)/s+t} \quad (6)$$

となり，この値に近づくほど性能が引き出せているといえる．

5.2 性能評価環境

表 2 に示す Xeon クラスタおよび Athlon MP クラスタのうち 8 プロセッサを使用して 3 種類のヘテロ環境を設定し，性能評価を行った． プログラムのコンパイル環境は以下のとおりである．Athlon 1.53 GHz, Xeon 2.4 GHz，Xeon 3.06 GHz での Strassen アルゴリズムの性能は，$n = 4000$ で演算回数を $2n^3$ に換算した場合，それぞれ 2.259 GFLOPS，3.065 GFLOPS，3.820 GFLOPS となった．ここで，BLAS で用いた ATLAS は各 CPU 向けに最適化したものを用いている．これらの値を各プロセッサの演算性能比として用いることにする．以下，Athlon 1.53 GHz を Ath1.53 GHz と書き，Xeon 2.4 GHz，Xeon 3.06 GHz を単に 2.4 GHz，3.06 GHz と書くことにする．以下に，3 種類の環境の特徴を示す．

表 2 性能評価環境
Table 2 Cluster specification.

	Xeon クラスタ	Athlon MP クラスタ
CPU	Xeon 2.4 GHz×2 Xeon 3.06 GHz×2	Athlon MP 1.53 GHz ×2
# of Nodes	15	22
Memory	1 GB DDR SDRAM	1 GB DDR SDRAM
L1 I-Cache	12Kμops	64 Kμops
L1 D-Cache	8 Kμops	64 Kμops
L2 Cache	512 Kμops	256 Kμops
Network	1000 Base-T Gigabit Ethernet	1000 Base-T Gigabit Ethernet
Compiler	gcc 2.96	gcc 3.2.2
OS	Linux(Kernel 2.4.18)	Linux(Kernel 2.4.18)
mpich	version 1.2.5	version 1.2.5
BLAS	ATLAS 3.4.1	ATLAS 3.4.1

評価環境 1：2.4 GHz×4+3.06 GHz×4
プロセッサの演算性能の差が小さく，ヘテロ性が小さい環境であり，単一サイズ分割では最適化が困難な環境であると考えられる．本論文で提案した再帰的分割の効果があるかを見るために，この環境を設定した．$s = (3.065 \times 4 + 3.820 \times 4)/(3.065 \times 8) \approx 1.123$ である．

評価環境 2：2.4 GHz×7+3.06 GHz×1
8 プロセッサで構成した場合，ヘテロ性が最も小さい環境である．このような環境で再帰的分割によって性能向上が得られるのなら，ヘテロ性の大小にかかわらず，さまざまなヘテロ環境上で再帰的分割が有効であると考えられる．$s = (3.065 \times 7 + 3.820 \times 1)/(3.065 \times 8) \approx 1.031$ である．

評価環境 3：Ath1.53 GH+2.4 GHz×3+3.06 GHz ×4
ある程度のヘテロ性を持つ環境であり，単一サイズ分割でも性能向上が見込める環境である．単一サイズ分割と再帰的分割による性能向上の差を見るために，この環境を設定した．$s = (2.259 \times 1 + 3.065 \times 3 + 3.820 \times 4)/(2.259 \times 8) \approx 1.535$ である．

5.3 性能評価方法

再帰的分割を用いてヘテロ環境向けに最適化した Strassen アルゴリズム（以下，H-Strassen-R と書く），単一サイズ分割を用いて最適化した Strassen アルゴリズム（以下，H-Strassen），最適化を行っていない Strassen アルゴリズム（以下，Strassen）の性能の比較を行うことで，最適化による効果および再帰的分割の有効性を検証する．従来の Strassen アルゴリズムは，プロセッサ数が平方数でなければならない．8 プロセッサで性能を評価するため，**図 10** の左のように行列を 4×4 個に分割し，小行列を 2 個ずつ割り当てたものと比較を行うことにする．また，並列 Strassen

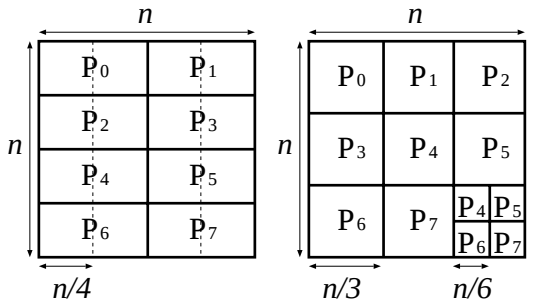


図 10 従来手法の負荷分散（左）と評価環境 1 の H-Strassen-R に用いた負荷分散（右）

Fig. 10 The data distributions to processors, using “Strassen” (left), and using “H-Strassen-R” (right) on Environment 1.

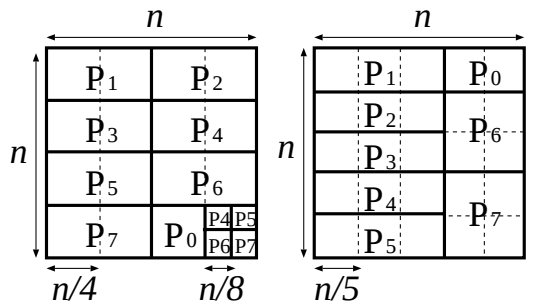


図 11 評価環境 3 の H-Strassen-R に用いた負荷分散（左）と H-Strassen に用いた負荷分散（右）

Fig. 11 The data distributions to processors, using “H-Strassen-R” (left), and using “H-Strassen” (right) on Environment 3.

アルゴリズムは，分割方法によって演算回数が異なるので，単純に FLOPS 値を比較しただけでは，ヘテロ化により性能向上が得られているかが分かりにくい．そこで，並列 Strassen アルゴリズムの演算回数を分割方法によらずに通常の行列積の演算回数である $2n^3$ と見なし，FLOPS 値を算出して比較を行うことにする．この比較方法は実行時間を比較しているのと同値である．

5.4 評価結果

- 評価環境 1
  評価結果を**図 12** に示す．
  Strassen の負荷分散は図 10 の左のようになり，この負荷分散は全評価環境共通で用いている．また，H-Strassen-R の負荷分散については図 10 の右のようになった．なお，図 10 において，P_0〜P_3 が 2.4 GHz，P_4〜P_7 が 3.06 GHz である．H-Strassen-R のピーク性能は $n = 12000$ のとき約 20.88 GFLOPS であり，Strassen のピーク性能は $n = 12000$ のとき約 20.01 GFLOPS であるから，最適化によって約 4.3%性能が向上している．一方，H-Strassen については評価関数が最適と判断した負荷分散が図 10 の左にある Strassen のものと一致してしまい，最適化の効果は現れなかった．
- 評価環境 2
  評価結果を**図 13** に示す．
  この環境上では，Strassen と H-Strassen-R，H-Strassen すべての負荷分散が一致し，図 10 の左のようになったので，最適化の効果は現れなかった．ピーク性能は Strassen，H-Strassen，H-Strassen-R ともに，$n = 12000$ のとき，約 20 GFLOPS である．
- 評価環境 3
  評価結果を**図 14** に示す．また，実際に行われた負荷分散を**図 11** に示す．なお，図 11 において，P_0 が Ath1.53 GHz，P_1〜P_3 が 2.4 GHz，P_4〜P_7 が 3.06 GHz である．H-Strassen-R のピーク性能は $n = 12000$ のとき約 19.24 GFLOPS で

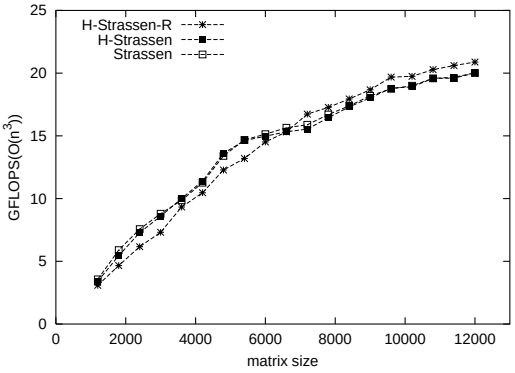


図 12 評価環境 1 における各手法の性能

Fig. 12 Performance results on Environment 1.

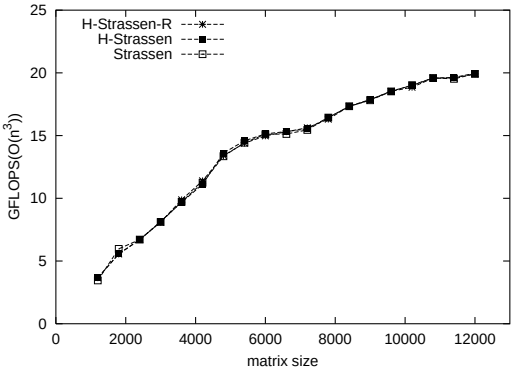


図 13 評価環境 2 における各手法の性能

Fig. 13 Performance results on Environment 2.

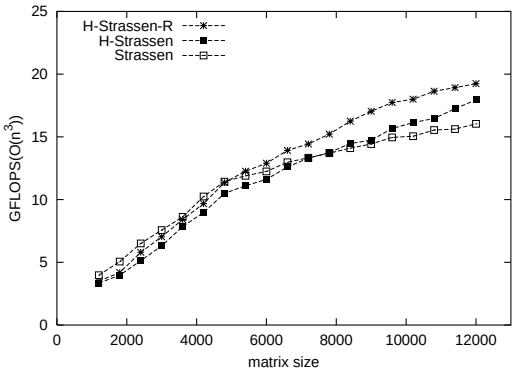


図 14 評価環境 3 における各手法の性能
Fig. 14 Performance results on Environment 3.

あり，Strassen のピーク性能は $n = 12000$ のとき約 16.04 GFLOPS である．したがって，最適化によって約 20.0%性能が向上している．また，H-Strassen のピーク性能は $n = 12000$ のとき約 17.96 GFLOPS であるから，Strassen と比較して約 12.0%性能が向上しているが，H-Strassen-R よりは劣っている．

5.5 考　　察

ヘテロなクラスタ環境を 3 種類設定し，性能評価を行ったところ，評価環境 1 では最適化により約 4.3%，評価環境 3 では約 20%の性能向上が得られた．しかし，評価環境 2 では，最適化の効果は現れなかった．**表 3**，**表 4** に行列サイズを $n = 12000$ とした場合の，評価環境 1 および評価環境 3 での各手法の実行時間の内訳を実測値で示す．各表において,「演算」は演算時間を示している．ただし，再帰的分割を行った H-Strassen-R については，再分割してできた小行列で発生する通信時間を演算時間に含めてある.「通信」は小行列の broadcast にかかる通信時間であり,「同期」は通信フェイズを待つことで発生する待ち時間である．また，最適化が行われたものに対しては，実測時間の下に評価関数が導出した予測時間を太字で示した．

この表から，H-Strassen，H-Strassen-R ともに，ロード インバランスが完全には解消できていないことが分かる．これは，ロード インバランスを完全に解消するために行列の分割を細かくすると，演算回数・通信量ともに増加するため，評価関数がロード バランスをある程度とりつつ，全体として実行時間が最小となるような負荷分散を選択した結果であると考えられる．たとえば表 4 において，H-Strassen の通信時間に着目すると，Strassen と比較してかなり増加しているが，これは Strassen と比較して行列を細かく分割していること，そして通信のバランスが崩れていることが原因である．したがって，ロード インバランスを完全に解消するためにさらに行列を細かく分割すると，それ以上に通信時間が増加し，全体としては逆に性能が低下することが予想される．したがって，図 11 の右の負荷分散が最善であると評価関数は判断したと考えられる．

単一サイズ分割を用いた H-Strassen と再帰的分割を用いた H-Strassen-R を比較すると，評価環境 1，評価環境 3 では H-Strassen-R の方が優れている．評価環境 1 はヘテロ性が小さいので，H-Strassen では最適化を行うためには行列を細かく分割しなければならないが，これは逆に性能が低下するため，評価関数は最適化を行わなかった．一方，H-Strassen-R では再帰的分割によって，演算回数を増加させずにロード バランスの向上が達成できたため，わずかではあるが性能向上が見られた．評価環境 3 では，双方で性能向上が達成された．H-Strassen ではロード インバランスがある程度解消されているものの，通信時間の増加が原

表 3 評価環境 1 における実行時間の内訳（$n = 12000$，単位（秒））
Table 3 Detail of execution times on Environment 1 (in second).

		演算	通信	同期	実行時間
Strassen	2.4 GHz	132.4	40.3	0	172.7
	3.06 GHz	105.9	40.3	27.5	172.7
H-Strassen-R	2.4 GHz	109.7	41.3	14.5	165.5
	（予測）	**108.1**	**38.1**	**14.5**	**160.7**
	3.06 GHz	124.2	41.3	0	165.5
	（予測）	**122.6**	**38.1**	**0**	**160.7**

表 4 評価環境 3 における実行時間の内訳（$n = 12000$，単位（秒））
Table 4 Detail of execution times on Environment 3 (in second).

		演算	通信	同期	実行時間
Strassen	Ath1.53 GHz	172.9	42.6	0	215.5
	2.4 GHz	130.0	42.6	42.9	215.5
	3.06 GHz	106.4	42.6	66.5	215.5
H-Strassen	Ath1.53 GHz	112.5	59.9	20.0	192.4
	（予測）	**117.1**	**54.9**	**20.3**	**192.3**
	2.4 GHz	123.8	59.9	8.7	192.4
	（予測）	**128.8**	**54.9**	**8.6**	**192.3**
	3.06 GHz（P_4〜P_5）	101.1	59.9	32.4	192.4
	（予測）	**103.0**	**54.9**	**34.4**	**192.3**
	3.06 GHz（P_6〜P_7）	132.5	59.9	0	192.4
	（予測）	**137.4**	**54.9**	**0**	**192.3**
H-Strassen-R	Ath1.53 GHz	85.6	49.6	44.4	179.6
	（予測）	**87.6**	**45.1**	**40.9**	**173.6**
	2.4 GHz	130.0	49.6	0	179.6
	（予測）	**128.5**	**45.1**	**0**	**173.6**
	3.06 GHz	127.2	49.6	2.8	179.6
	（予測）	**126.1**	**45.1**	**1.4**	**173.6**

因でそれほど性能が向上していないが，H-Strassen-R では Ath1.53 GHz 以外のプロセッサはほとんど待ち時間がなく，かつ通信時間がそれほど増加していないので，さらに高い性能向上が得られたといえる．

また，理論的な性能向上の上限値と比較すると，評価環境 1 では表 3 における演算時間の最大値および通信時間より式 (6) での t の値は $t \approx 0.233$ である．評価環境 2 の場合，Strassen の性能は 2.4 GHz のプロセッサがボトルネックとなるので図 12 における Strassen の性能と同じになり，$t \approx 0.233$ である．また，評価環境 3 では $t \approx 0.198$ である．したがって，評価環境 1〜3 においては，ヘテロ化によってロードバランスが完全にとれ，通信時間がまったく増加しないという理想的な場合において，$T_{\mathrm{homo}}/T_{\mathrm{hetero}}$ より，それぞれ，およそ 9.1%，2.4%，38.7%の性能向上が得られると予想できる．

しかし，実際には評価環境 1 では約 4.3%の性能向上であり，評価環境 2 では性能が向上せず，評価環境 3 では約 20.0%の性能向上にとどまった．この理由としては，先に述べたとおり，ヘテロ化によって通信が不均衡になることやロードインバランスが完全に解消されていないことが原因である．また再帰的分割の適用により小さい方の小行列について通信が新たに発生するので，全体として通信量が増えていることも理由としてあげられる．また，評価環境 2 で性能が向上しなかったのは，ヘテロ性が非常に小さいことから，最適化による待ち時間の減少量よりも，演算量や通信量が増加することによる演算時間・通信時間の増加量が大きくなると予想され，最適化が行われなかったからだと考えられる．また，評価関数が予想した性能向上率は，評価環境 1 では 172.7/160.7 より約 7.5%，評価環境 3 では 215.5/173.6 より 24.1%となっており，実際の結果に近い値となっている．

6. おわりに

本論文では，Strassen の行列積アルゴリズムをヘテロ環境向けに最適化，実装およびその性能評価を行った．

性能評価結果では，負荷分散を考慮することにより，ヘテロ性が小さな環境上においても性能の向上が見られ，また，最適化しない場合に比べ最大で約 20%の性能向上が得られた．

一般に，通信が非常に少なく，負荷分散が容易に行えるアプリケーションでは，ヘテロ化が容易であり，ヘテロ性が小さな環境上でも性能向上が得られる場合が多い．しかし，並列 Strassen アルゴリズムの場合は，負荷分散を細かく調整するために分割を細かくすると，通信量・演算量共に増加するため，ヘテロ性が小さい環境では逆に性能が低下する可能性がある．

したがって，Strassen アルゴリズムをヘテロ環境向けに最適化する場合は，ロードバランスをとりつつ，行列の分割をできるだけ粗くすることが重要であり，この問題を解決するため，本論文では再帰的分割を提案し，その有効性を示した．

アプリケーションをヘテロ環境向けに並列化する場合に，各プロセッサの演算性能に応じた負荷分散を行うという手法がよく採られるが，ロードバランスをとることだけでなく，通信量や個々のアプリケーションの特性を考慮したうえで最適な負荷分散を行うことが，性能を向上させるためには重要であると考えられる．

今後の課題としては，CPU 性能のほかにも物理的なネットワーク性能が異なる場合や，メモリ，キャッシュ性能がヘテロな場合の性能評価も必要であると思われる．また，さまざまなアプリケーションをヘテロ環境向けに並列化することを通して，ヘテロ環境に対する一般的な負荷分散手法の研究が必要であると考えられる．

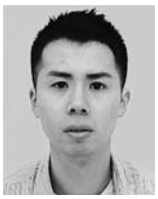
謝辞 本研究の一部は，文部科学省科学研究費補助金若手研究（B）（課題番号 14780185）による．

参考文献

1) Strassen, V.: Gaussian elimination is not optimal, *Numer. Math.*, Vol.13, pp.354–356 (1969).
2) 笹生 健，松岡 聡，建部修見：ヘテロなクラスタ環境における並列 LINPACK の最適化，並列処理シンポジウム JSPP2002 論文集，pp.71–78 (2002).
3) 岸本芳典，市川周一：不均一クラスタ上での実行時間予測モデルとその評価，情報処理学会研究報告 2003-HPC-95, pp.161–166 (2003).
4) Beaumont, O., Boudet, V., Rastello, F. and Robert, Y.: Matrix-Matrix Multiplication on Heterogeneous Platforms, *IEEE Trans. Parallel and Distributed Systems*, Vol.12, pp.1033–1051 (2001).
5) Dovolnov, E., Kalinov, A. and Klimov, S.: Natural Block Data Decomposition for Heterogeneous Clusters, *International Parallel and Distributed Processiong Symposium* (*IPDPS'2003*), pp.1–2 (2003).
6) Luo, Q. and Drake, J.B.: A Scalable Parallel Strassen's Matrix Multiplication Algorithm for Distributed-Memory Compters, *Proc. 1995 ACM Symposium on Applied Computing*, pp.221–226 (1995).

7) Desprez, F. and Suter, F.: Mixed Implememtations of the Top Level Step of Strassen and Winograd Matrix Multiplication Algorithms, *International Parallel and Distributed Processiong Symposium* (*IPDPS'2001*), pp.1–3 (2001).
8) Winograd, S.: On multiplication of 2×2 matrices, *Linear Algebra and Its Applications*, Vol.4, pp.381–388 (1971).
9) Fox, G.C., Otto, S.W. and Hey, A.J.G.: Matrix algorithms on a hypercube I: Matrix multiplication, *Parallel Computing*, Vol.4, pp.17–31 (1987).

(平成 15 年 10 月 10 日受付)
(平成 16 年 1 月 21 日採録)



大滝 雄介

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，同大学大学院理工学研究科在学中．並列数値計算に関する研究に従事．

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．

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，現在に至る．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．

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．