ハイパフォーマンス
コンピューティング 93－2
（２００３．３．１１）

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，SMP クラスタを用いた擬似的なヘテロ環境上で性能評価を行った．ヘテロなクラスタ環境においては各ノード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は必要不可欠である．本研究では各ノードの CPU 性能，通信，そして Strassen の行列積アルゴリズム中における再帰回数について最適化を行うことにより，実行時間の最小化を図った．その結果，最適化しない場合と比べ最大で約 34.6%性能が向上し，演算量が $O(n^3)$ である通常の行列積と比べ最大で約 14.9%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, appropriate data distribution is most important to achieve maximum performance as a whole. In order to minimize execution time, we consider CPU performance, communication and recursion level in the Strassen's algorithm. As a result, we achieve nearly 35% speedup in a heterogeneous clustering environment compared to the conventional parallel Strassen's algorithm and nearly 15% speedup than the traditional $O(n^3)$ 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等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，ノード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，キャッシュ等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．各ノードのハードウェア性能がホモジニアスな環境においては，各ノード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場合もある．しかし，それらの性能が異なるノード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ノードが高いノードの足を引っ張り，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，各ノードの演算性能を考慮した負荷分散を考えなければクラスタの性能を出し切ることができず，高速化は望めない．

ヘテロジニアス環境（以下，ヘテロ環境と呼ぶ）においては，Linpack のベンチマークプログラムである HPL[1] がヘテロ環境向けに最適化されており，ヘテロ環境でも高い性能が出ていることが報告されている[2]．

Strassen の行列積アルゴリズム[3] 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．この Strassen アルゴリズムについては，RISC プロセッサ上で最適化がいくつか試みられており[4)~6)]，並列計算機においても Strassen アルゴリズムは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[7),8)]．

† 筑波大学第三学群情報学類
College of Information Sciences, Third Cluster of Colleges, University of Tsukuba
†† 筑波大学電子・情報工学系
Institute of Information Sciences and Electronics, University of Tsukuba

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．そこで本論文では，Strassen アルゴリズムにおける再帰回数，各ノードの CPU 性能，そして通信量を考慮することにより，実行時間を最小化を図り，ヘテロ環境向けの最適化を行う．また，SMP クラスタを用いて擬似的なヘテロ環境上を構築し，性能評価を行う．

2. Strassen の行列積アルゴリズム

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[3]（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[3]．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[9] 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 $f_s(n)$ とおくと，

$$\begin{aligned} f_s(n) &= 7 f_s\left(\frac{n}{2}\right) + 15\left(\frac{n}{2}\right)^2 \\ &\vdots \\ &= 2n_0^3 \cdot 7^{\log_2 \frac{n}{n_0}} + \frac{15}{4} n^2 \cdot \frac{1 - \left(\frac{7}{4}\right)^{\log_2 \frac{n}{n_0}}}{1 - \frac{7}{4}} \\ &\approx 2n_0^3 \cdot 7^{\log_2 \frac{n}{n_0}} + 5n^2 \cdot \left(\frac{7}{4}\right)^{\log_2 \frac{n}{n_0}} \\ &= c \cdot n^{\log_2 7} \end{aligned}$$

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

3. Strassen アルゴリズムの並列化

ここでは Strassen アルゴリズムの並列化手法として，二次元分割による並列化手法と置換行列による並列化手法[7),8)] を紹介する．以下簡単のため，プロセッサ台数は P^2 であるとして考える．

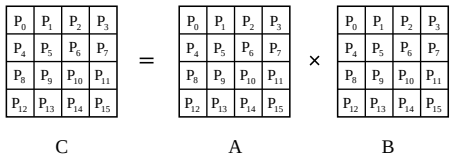

図 1 プロセスのマッピング例

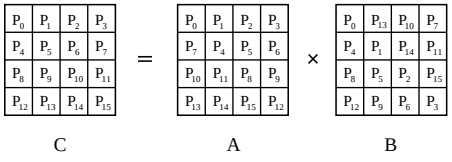

図 2 行列交換後のプロセスのマッピング

```
procedure multiply_2
step := 0;
begin
  C_ij = A_ik B_kj
  while step < P do begin
    通信を行い，行列 A, B をシフトさせる．
    C_ij = C_ij + A_ik B_kj
    step := step + 1;
  end
end;
```

図 3 二次元分割による並列行列積アルゴリズム

3.1 二次元分割

二次元分割では，行列を $P \times P$ のメッシュに分割し，各プロセスに割り当てる．負荷分散の例を図 1 に示す．また，C_{ij} を割り当てられているプロセスをプロセス (i, j) と呼ぶことにすると，プロセス (i, j) は

$$C_{ij} = A_{i1} B_{1j} + A_{i2} B_{2j} + \cdots + A_{iP} B_{Pj}$$

を計算することになる．このままではすぐに計算を始められないプロセスが存在するので，通信によって行列の交換を行い，

$$k = (i + j - 2) \bmod P + 1$$

と定められる定数 k を用いて，図 2 のようにプロセス (i, j) に A_{ik} および B_{kj} を割り当てる．C_{ij} の計算には行列 A の i 行および行列 B の j 列全体が必要なため，通信によって A を行方向へ，B を列方向へシフトさせながら計算を行い，行列積の計算に Strassen アルゴリズムが適用される．図 3 に二次元分割を用いた並列行列積アルゴリズムを示す．

二次元分割を用いて並列化を行った場合，行列は $P \times P$ 個のメッシュに分割される．したがって，各プロセスはサイズが n/P の行列積を P 回計算すること

になり，全体の演算回数を $f_2(n)$ とおくと，

$$f_2(n) = P^2 \cdot P \cdot f_s\left(\frac{n}{P}\right) = P^{\log_2 \frac{8}{7}} \cdot f_s(n)$$

となる．また，通信量は個々のプロセスで $P(n/P)^2$ となるので，全体の通信量を $c_2(n)$ とおくと，

$$c_2(n) = P^2 \cdot P\left(\frac{n}{P}\right)^2 = P \cdot n^2$$

である．

3.2 置換行列による並列化

二次元分割は，行列を分割してから Strassen アルゴリズムを適用するというものであるが，置換行列 Q を用いると，Strassen アルゴリズム自体を並列化することが可能で，演算量を抑えることができる[7),8)]．

置換行列を用いた行列積では，各プロセスに二次元分割と同様の割り当てを行う．ここで，二次元分割と違う点は，通信を行って行列のシフトを行いながら行列積を計算するという方法を取らず，木構造の末端で並列行列積に切替えることである．このアルゴリズムを用いて計算を行った場合は，計算結果として，QCQ^T が格納されるので，正しい計算結果を求めるには，

$$Q(QCQ^T)Q^T = C$$

として置換を戻せば良い．なお，最後に左右からかける置換行列は，単なる行と列の交換と同値なので，$O(n^2)$ の代入ですみ，$O(n^3)$ に比べると n が大きい場合にはほとんど無視できる量である．

置換行列による並列行列積アルゴリズムを**図 4** に示す．置換行列を用いた場合は，Strassen アルゴリズム全体が並列化されているが，切替えサイズが $n_0 \times P$ になるため，演算量 $f_p(n)$ は，

$$f_p(n) = \frac{2(Pn_0)^3 + 5(Pn_0)^2}{(Pn_0)^{\log_2 7}} n^{\log_2 7} < \frac{2(Pn_0)^3 + P \cdot 5(Pn_0)^2}{(Pn_0)^{\log_2 7}} n^{\log_2 7} = f_2(n)$$

となり，置換行列版は二次元分割版よりも演算量は少ないことがわかる．また，通信量については，通常の行列積に切替えたときに全体で $P^2 \cdot P \cdot n_0^2$ の通信量が発生し，再帰の深さが $\log_2 \frac{n}{n_0}$ となるので，全体の通信量 $c_p(n)$ は，

$$c_p(n) = 7^{\log_2 \frac{n}{n_0}} \cdot P^2 \cdot P \cdot n_0^2 = n_0^{\log_2 \frac{4}{7}} P^3 \cdot n^{\log_2 7}$$

である．二次元分割と置換行列を比較すると，演算量では置換行列が有利であるが，通信量では不利であり，どちらが有効であるかはプロセッサ性能とネットワーク性能の比によって変わってくる．しかし，最近はプロセッサの性能がネットワーク性能に比べ飛躍的に伸びていることから二次元分割の方が有効であると考えられ，本論文では二次元分割を用いることにする．

```
procedure parallel_strassen
begin
  if n ≤ n0 then
    call procedure multiply_2.
  else
    begin
      S1～S8 を求める.
      call procedure parallel_strassen.
      T1, T2, C11～C22 を計算する.
    end
end;

procedure multiply_p
begin
  call parallel_strassen.
  C = QCQ^T として置換を戻す.
end;
```

図 4 置換行列による並列行列積アルゴリズム

4. ヘテロなクラスタ環境向けの最適化

4.1 最適化の要素

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るが，本論文では以下の要素を考慮した．

- CPU 性能
  CPU 性能が高いノードに行列を多く割り当てることでロードバランスが取れ，行列の分割個数が多いほど負荷分散を最適化しやすい．
- ネットワーク
  CPU 性能に応じて行列を割り当てるため，CPU 性能の高いプロセスに通信が集中し，見かけ上，CPU 性能の高いプロセスを接続しているネットワーク性能が低いようにみえる．全体の処理時間に占める通信時間の割合が高い場合に考慮する必要がある．分割個数が多いほど通信量は増加する．
- 再帰回数
  Strassen アルゴリズムでは行列のサイズが大きいほど再帰回数が多くなり，それだけ演算回数の削減が可能である．つまり行列の分割個数が少ないほど演算回数が削減できる．したがってロードバランスと Strassen アルゴリズムの有効性はトレードオフの関係にあり，単純にロードバランスを最適化しただけで性能が良くなるとは限らない．

4.2 最適化手法

4.2.1 データファイルの作成

CPU やネットワークの評価を行うために，適当な N を 1 つ定め，事前に最適化されていない並列行列積を実行し，その結果から以下のデータを記述したデータファイルを作成しておく．

- $N \times N$ の行列を交換するのにかかる通信時間．
- 各ノードが $N \times N$ の行列積を 1 回実行するのに要する時間．

4.2.2 CPU 性能についての最適化

まずはじめに CPU 性能について最適化を行う．ノード数を m とする．データファイルから各ノードの Strassen アルゴリズムの実行時間を参照し，$t(p)$ とする．Strassen アルゴリズムは $O(n^{\log_2 7})$ であるから，サイズ n におけるノード p の実行時間 $\mathrm{time}(p)$ は

$$\mathrm{time}(p) = 7^{\log_2 \frac{n}{N}} \times t(p)$$

これより実行速度は以下の式で与えられる．

$$\mathrm{speed}(p) = \frac{1}{\mathrm{time}(p)}$$

これが各ノードの評価値である．行列の分割個数を M^2 とするとき，割り当て個数 $\mathrm{assign}(p)$ は以下の式で与えられる．

$$\mathrm{assign}(p) = \left\lfloor \frac{\mathrm{speed}(p)}{\sum_{k=1}^{m} \mathrm{speed}(k)} M^2 \right\rfloor$$

このとき，M^2 個全てが割り当てられるわけではなく，余りが生じる．この時点での演算時間を $\mathrm{comp}(p)$ とすると，

$$\mathrm{comp}(p) = \mathrm{assign}(p) \times \mathrm{time}(p)$$

となるから，余りについては，割り当てが 1 個増えた場合の演算時間 $\mathrm{next}(p)$ を

$$\mathrm{next}(p) = (\mathrm{assign}(p) + 1) \times \mathrm{time}(p)$$

として求め，$\mathrm{next}(p)$ が最も小さい，つまり割り当てが増えた場合に最も速く演算が終了するノードに割り当てる行列を 1 個追加するという方法を取る．この手順を M^2 個全てを割り当てるまで繰り返せば，CPU 性能について最適化される．

4.2.3 通信時間の推測

CPU 性能について最適化された割り当てに対し，通信量の評価を行う．

通信量は $O(n^2)$ であるから，データファイルから得られた通信時間を t_N とおくと，サイズ n の行列について，行列を 1 個交換するのにかかる通信時間 t_n は，

$$t_n = 4^{\log_2 \frac{n}{N}} \times t_N$$

通信は行列 A, B について行われるので，A の通信にかかる時間と B の通信にかかる時間の和が全体の通信時間となる．

ここで，CPU 性能について最適化された割り当てに対して，行列 A, B について最も通信が集中するノードを求め，その回数をそれぞれ $\max(A)$，$\max(B)$ とおく．このとき，全体の通信時間を comm とおくと，

$$\mathrm{comm} = (\max(A) + \max(B)) \times t_n$$

として求められる．

4.2.4 CPU，ネットワーク，再帰回数を考慮した最適化

$\mathrm{comp}(p)$ および comm を用いると，行列の分割個数が $s \times s$ のとき，ノード p の実行時間 $T(p, s)$ は，

$$T(p, s) = \mathrm{comp}(p) + \mathrm{comm}$$

したがって，全体の実行時間は以下のように推測される．

$$T_{all}(s) = s \times \max(T(p, s))$$

先に述べた通り，ロードバランスと再帰回数はトレードオフの関係にあるので，M^2 の値を $1 \times 1, 2 \times 2 \cdots$ と変えながら最適な割り当てを求める必要がある．

ここで，s^2 の値を大きくしてロードバランスを取ったところ，再帰回数の減少と通信量の増大によって逆に実行時間が増加したと仮定すると，ロードバランスを取ったことが逆効果であったと考えられる．したがって，それ以上分割を細かくしても実行時間を削減することはできないと考えられ，以下の不等式が成立する．

ある s について，

$$T_{all}(s) < T_{all}(2s)$$

ならば

$$\cdots > T_{all}\left(\frac{s}{2}\right) > T_{all}(s) < T_{all}(2s) < \cdots$$

つまり，T_{all} は $s \times s$ 個の分割のときに最小値をとり，実行時間を最小化する最適な分割個数であると考えられる．

5. 性能評価

5.1 性能評価環境

表 1 に示す Pentium II Xeon クラスタのうち 4 ノードを使用し，ノード内の CPU の使用台数をソフトウェアで制御することで，ノード単位で擬似的に CPU ヘテロなクラスタ環境を作り，性能評価を行った．ノード内の演算については，割り当てられた行列にさらに二次元分割を施し，OpenMP を用いて並列化した．以下，各ノードを N0，N1，N2，N3 と呼び，各ノード内部の CPU 使用台数が 1，2，3，4 のとき，これを便宜的に {N0, N1, N2, N3}={1, 2, 3, 4} と書くことにし，評価は {N0，N1，N2，N3}={1, 2, 3, 4}，{N0, N1, N2, N3}={3, 3, 4, 4} の 2 つについて行うことにする．また，ハイブリッド化したプログラムのコンパイル環境は以下の通りである．

- OpenMP コンパイラ：omcc-1.4a[10)]
- MPI：SCore-5.0.1[11)]
- BLAS：ATLAS 3.4.1[12)]

5.2 性能評価方法

ヘテロ環境向けに最適化した Strassen アルゴリズムを「ヘテロ版 Strassen」，最適化していないものを「Strassen」，通常の行列積について同様に「ヘテロ版 $O(n^3)$」，「$O(n^3)$」と呼ぶことにし，最適化による速度向上率およびヘテロ版 $O(n^3)$ とヘテロ版 Strassen の性能比較を行う．また，Strassen アルゴリズムと通常の行列積では演算回数が異なり，単純に MFLOPS 値を比較しただけでは，どちらがより高速に行列積を計算できているか分かりにくいため，Strassen アルゴリズムの演算回数を $O(n^3)$ とみなし，性能比較を行

うことにする．

5.3 評価結果

グラフ中で，Strassen（hetero）はヘテロ版 Strassen の性能，$O(n^3)$（hetero）はヘテロ版 $O(n^3)$ の性能を表すものとする．

- {N0，N1，N2，N3}={1，2，3，4} の場合
  評価結果を**図 5** に示す．
  ヘテロ版 Strassen のピーク性能は $n = 10240$ のとき約 2.273 GFLOPS であり，Strassen のピーク性能は $n = 10240$ のとき約 1.688 GFLOPS である．したがって，最適化によって約 34.6%性能が向上している．
  通常の行列積では，ヘテロ版 $O(n^3)$ のピーク性能は $n = 10240$ のとき約 1.978 GFLOPS，最適化を行っていない $O(n^3)$ のピーク性能が $n = 3072$ のとき約 1.084 GFLOPS となり，それぞれ理論ピーク性能の約 43.9%，約 24.0% に相当し，最適化によって約 82.4%の性能向上が得られている．また，ヘテロ版 Strassen はヘテロ版 $O(n^3)$ に対し，約 14.9%実行速度が向上している．
- {N0，N1，N2，N3}={3，3，4，4} の場合
  評価結果を**図 6** に示す．
  Strassen アルゴリズムに関しては，両者の負荷分散が一致してしまい，最適化の効果が現れず，図 6 においても両者は重なっている．これはロードバランスを取ると，通信量と演算回数の増加により，最適化をしない場合と比べ実行時間が増加してしまうため，最適化が行われなかったからである．また，通常の行列積に関しても，$1024 \leq n \leq 7168$ のときは，ヘテロ版 $O(n^3)$ と $O(n^3)$ の負荷分散が一致し，最適化が行われなかった．これは n がそれほど大きくないため，全体の実行時間に占める通信の割合が小さくなく，ロードバランスを取らない方が実行時間が小さくなるためである．$8192 \leq n \leq 10240$ のときは最適化されたが，$O(n^3)$ のピーク性能は $n = 10240$ のとき約 2.72GFLOPS，ヘテロ版 $O(n^3)$ のピーク性能は $n = 10240$ のとき約 2.866GFLOPS となり，最適化の効果は約 5.3%しか現れなかった．これはロードバランスを取ることによる同期時間の削減効果が小さく，逆に通信時間が増加したためだと考えられる．また，Strassen アルゴリズムのピーク性能は $n = 10240$ のとき約 3.141GFLOPS であり，ヘテロ版 $O(n^3)$ と比べ約 9.5%実行速度が向上している．

5.4 考　　察

ヘテロなクラスタ環境を 2 種類設定し，性能評価を行ったところ，図 5 では最適化により約 34.6%の性能向上が得られ，通常の行列積と比べ最大で約 14.9%実行速度が向上した．しかし，図 6 では，Strassen アルゴリズムの方に関しては，全く最適化の効果はなく，

表 1　Pentium II Xeon クラスタの仕様

CPU	Intel Pentium II Xeon 450MHz×4
Number of Node	8
Memory	2GB SDRAM per Node
L1 instruction Cache	16KB
L1 data Cache	16KB
L2 Cache	1MB
Network	800Mbps Myrinet
Compiler	gcc-3.0.2
Operating System	Linux(Kernel 2.4.18)
SCore[11]	version 5.0.1

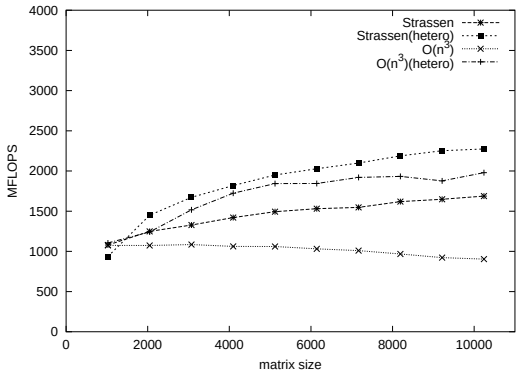

図 5　{N0，N1，N2，N3}={1，2，3，4} の場合の評価結果

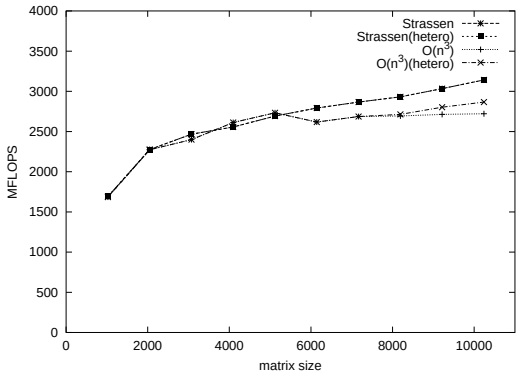

図 6　{N0，N1，N2，N3}={3，3，4，4} の場合の評価結果

通常の行列積に関しても，最適化の効果はほとんど現れなかった．

行列積をヘテロ環境向けに並列化する場合に，行列の分割を細かくしてロードバランスを取ると，通信量が増えるという問題が起こる．したがって，図 6 のように，ヘテロ性が比較的小さい場合は，ロードバランスを取ることによる同期時間の削減効果が小さく，逆に通信時間が増えるので，最適化は難しい．

一方，行列積に Strassen アルゴリズムを適用する場合，先に述べた通信の問題に加え，演算回数が増加してしまうという問題も起こる．この演算回数が増加するという問題は，ロードバランスを取るために行列

の分割を細かくすると，個々の行列のサイズが小さくなってしまうために起こるものであり，ロードバランスを犠牲にしてでも演算回数の削減を狙った方が実行時間を抑えられる場合がある．したがって，Strassen アルゴリズムは，ロードバランスを取ることが通常の行列積と比べに非常に困難であることがいえる．図 5 がそれを示している．通常の行列積に関してはヘテロ環境向けに最適化したことにより，約 82.4%の性能向上が得られたが，Strassen アルゴリズムの場合は約 34.6%の性能向上にとどまった．これはヘテロ版 Strassen のロードバランスが依然として良くないためである．しかし，ロードバランスを取るために行列を細かく分割すると，演算回数が増えるため，逆に実行時間が増加してしまうのである．

両者を比較すると，通常の行列積の方がロードバランスを取ることが容易であり，最適化の効果は高いが，Strassen アルゴリズムの方が演算量が少ないため，通常の行列積と比べ実行時間を削減できる場合もある．したがって Strassen アルゴリズムは行列積を高速に計算する上で有効なアルゴリズムであることがいえる．

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，実装及びその性能評価を行い，その有効性を示した．

性能評価結果では，負荷分散を考慮することにより，最適化しない場合に比べ最大で約 34.6%の性能向上が得られた．また，Strassen アルゴリズムは $O(n^3)$ のアルゴリズムに比べ最大で約 14.9% の性能向上が得られ，ヘテロ環境においても Strassen アルゴリズムを適用することで通常の行列積のアルゴリズムに比べ高速に計算できることも示すことができた．

Strasssen アルゴリズムの特徴的な要素として再帰回数があり，これは演算回数の削減に関係する重要な要素である．Strassen アルゴリズムの並列化を行えば，行列の分割に行うことになるから，個々のプロセスに割り当てられる行列積の規模は分割しない場合と比べ小さくなり，Strassen アルゴリズムにおける再帰回数の減少は避けられない．したがって Strassen アルゴリズムの並列化効率は $O(n^3)$ の行列積のそれに比べて確実に落ちる．ヘテロ環境向けに最適化する場合は，ロードバランスを取る関係で，再帰回数がさらに落ちることが多い．このことから，Strassen アルゴリズムは並列化がしにくく，ヘテロ環境向けの最適化が難しいアルゴリズムであると言える．しかし，並列化効率が悪くても通常の行列積に比べれば高速な演算が可能であり，大規模な行列積の計算には非常に有効なアルゴリズムであると結論付けられる．

今後の課題としては，ノード間のロードバランスの向上のために Strassen アルゴリズムを拡張することや，より複雑な分割方法の確立が挙げられる．また，ヘテロ環境の要素として CPU 性能とネットワーク性能を考慮した最適化を行ったが，今後は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でも性能評価も必要であると思われる．

謝辞　本研究に関して，御助言，御討論頂きました，筑波大学 HPCS 研究室の各位に感謝致します．なお，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) HPL – A Portable Implementation of the High-Performance Linpack Benchmark for Distributed-Memory Computers: . http://www.netlib.org/benchmark/hpl/.
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) Strassen, V.: Gaussian elimination is not optimal, *Numer. Math.*, Vol. 13, pp. 354–356 (1969).
4) Laderman, J., Pan, V. and Sha, X.-H.: On practical algorithms for accelerated matrix multiplication, *Linear Algebra and Its Applications*, Vol. 162–164, pp. 557–588 (1992).
5) Huss-Lederman, S., Jacobson, E. M., Johnson, J. R., Tsao, A. and Turnbull, T.: Implementation of Strassen's Algorithm for Matrix Multiplication, *Proc. Supercomputing '96* (1996). (CD-ROM).
6) Thottethodi, M., Chatterjee, S. and Lebeck, A. R.: Tuning Strassen's Matrix Multiplication for Memory Efficiency, *Proc. SC'98* (1998). (CD-ROM).
7) 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).
8) Grayson, B. and van de Geijn, R.: A High Performance Parallel Strassen Implementation, *Parallel Processing Letters*, Vol. 6, pp. 3–12 (1996).
9) Winograd, S.: On multiplication of 2×2 matrices, *Linear Algebra and Its Applications*, Vol. 4, pp. 381–388 (1971).
10) Omni OpenMP Compiler Project: . http://www.hpcc.jp/Omni/.
11) PC Cluster Consortium: . http://www.pccluster.org.
12) Whaley, R. C., Petitet, A. and Dongarra, J. J.: Automated empirical optimizations of software and the ATLAS project, *Parallel Computing*, Vol. 27, pp. 3–35 (2001).