情報処理学会論文誌　コンピューティングシステム　Vol. 3　No. 3　199–208 (Sep. 2010)

マルチコアアーキテクチャのための密行列LU分解のプログラミング技術

里城晴紀[†1]　吉瀬謙二[†1]　小長谷明彦[†1]

近年，シングルコアプロセッサは消費電力と発熱の制限により性能限界に達したため，多数のプロセッサコアによって性能向上を図るマルチコア，メニーコアのプロセッサが主流となっている．マルチコアプロセッサの性能を引き出すためには，すべてのコアを無駄なく動作させるための並列性の確保と，多数のコアが同時アクセスすることで生じるメモリアクセスボトルネックの解消を同時に満たすことが課題となっている．密な線型方程式を効率良く解く LU 分解は高性能計算の代表的なベンチマークとして知られている．これまで，LU 分解の高速実行アルゴリズムとしては，並列処理を最大限に活用できる right-looking 法が適しているといわれていた．しかしながら，マルチコアプロセッサにおいては演算性能に比べメモリ性能が相対的に低いため，データ転送量の多い right-looking 法が必ずしも最大性能を示すとは限らない．本論文では，LU 分解を題材に，参照局所性が高い left-looking 法が，最大並列性を実現する right-looking 法よりも高性能を実現するマルチコアアーキテクチャの条件を，性能予測モデルと Cell BE での評価実験での結果をふまえて報告する．

On-chip Parallel Programming Techniques for Dense LU Decomposition

Haruki Satoshiro,[†1] Kenji Kise[†1]
and Akihiko Konagaya[†1]

Recently, multicore processor architectures have been getting attention from the viewpoint of the balance of design complexity and CPU performance in the constraint of electronic power consumption and transistor size. In the multicore processor architectures, high performance computing requires not only parallelism to make use a number of cores but also efficient data transfer mechanism to avoid memory access bottleneck. Dense linear algebra LU decomposition is one of the well-known algorithms used for benchmarks in high performance computing. It is usually said that the right-looking method is better than the left-looking method due to the available parallelism in the LU decomposition. However, this is not always true in the multicore architectures due to the memory bandwidth bottleneck. In this paper, architectural conditions in which the left-looking method overperformed the right-looking method are described with performance estimation models and empirical evaluation on Cell BE.

1. はじめに

近年，高性能プロセッサとしてマルチコアプロセッサ，メニーコアプロセッサの開発がさかんになった．主な理由は従来のシングルコアプロセッサではトランジスタの集積度向上による消費電力と発熱の制限が顕著になり性能の限界に達したからである．LSI の集積度の向上は依然続いており，開発費と性能のトレードオフの観点からコア数を増やしたマイクロアーキテクチャが注目を集めている[1)]．

有名なマルチコアプロセッサとして Cell/B.E.[2)] が知られている．Cell/B.E. は PowerPC Processor Element（PPE）と Synergistic Processor Element（SPE）をそれぞれ 1 基，8 基搭載している．制御を担う PPE と計算を担う SPE を混在させることで多様な要件に適するようになっている．特に単精度浮動小数点演算に関しては SPE 1 基あたり 25.6 Gflops の理論性能を持ち，Cell/B.E. 全体の理論性能は 204.8 Gflops となっている．

マルチコアプロセッサの性能を引き出す際の課題として並列性の確保とメモリバンド幅制限の克服があげられる．特にプロセッサの演算性能が著しく向上しているのに対して，メモリ性能の向上率は低く性能のギャップが大きく開いている．さらにマルチコア化によりプロセッサコア 1 基あたりの専用メモリ容量も減少傾向にあるためデータ転送の問題は深刻である．このためマルチコアプロセッサ上では十分な並列性がアルゴリズムに存在するにもかかわらず，メモリアクセスがボトルネックとなりスケールしないことが報告されている[3)]．

LU 分解は密な線形方程式を効率良く解くためのアルゴリズムで LAPACK でも提供されている[4)]．LU 分解の計算手順として right-looking 法，left-looking 法，Crout 法の 3 種類がよく知られている．従来から逐次，並列処理にかかわらず right-looking 法により LU 分解を行うのが最適といわれているが，right-looking 法は参照局所性が低いことを指摘されている[5)]．

本論文では left-looking 法と right-looking 法のそれぞれが有利となるアーキテクチャ条

†1 東京工業大学
Tokyo Institute of Technology

件を求め，コア数が多いプロセッサアーキテクチャ上の LU 分解においては参照局所性の高い left-looking 法が優位になることを示す．さらに Cell/B.E. 上で LU 分解性能を比較し，Cell/B.E. 上の left-looking 法は right-looking 法と同等以上に高速なことを示す．Crout 法は性能的にも参照局所性についても left-looking 法と right-looking 法の中間的な性質を示すため本論文では考察の対象としない．

本論文の構成は次のとおりである．最初に本論文で対象とするマルチコアアーキテクチャを定義する．次にブロック LU 分解アルゴリズムの right-looking 法と left-looking 法それぞれの特徴を示す．そして LU 分解性能予測モデルを示し Cell/B.E. 上の実測性能と比較し妥当なことを確認する．最後に性能予測モデルを基に right-looking 法と left-looking 法のそれぞれが適しているアーキテクチャ条件を求める．

2. マルチコアアーキテクチャ

本論文で考察した性能予測モデルが前提とするマルチコアアーキテクチャは次のとおりである．

(1)　均一性能を持つオンチップマルチコア
(2)　コア専用オンチップメモリ搭載
(3)　コア専用オンチップメモリ–メインメモリ間のデータ転送と計算と同時実行可能な機構
(4)　高性能コア間非同期通信機構

コア内のメモリアーキテクチャとしては，キャッシュメモリではなく計算と同時実行可能な DMA 転送機構を備えたコア専用オンチップメモリを仮定する．これは，計算とメモリ転送を同時実行させることによりメモリアクセスレイテンシによる遅延を隠ぺいするためである．コア間通信としては，データの到着およびタスクの終了を知らせるため非同期通信機構を仮定する．本予測モデルで論ずるアルゴリズムでは高速メモリ転送機構とコア間非同期通信機構があれば十分であり，コア間での高速データ転送機構は必ずしも必要とはしていない．

性能モデルを検証するマルチコアプロセッサとして Cell/B.E. を使用する．Cell/B.E. はヘテロジニアスマルチコアプロセッサであるが，計算に使用するコアは Synergistic Processor Element（SPE）のみのためすべてのコアが同一性能と見なしてよい．SPE のコア専用オンチップメモリはキャッシュメモリではなくプログラムの制御で自由に扱えるローカルストア（LS）であり，データの配置と先読みの最適化が容易である．SPE 間非同期通信のコストも約 300 サイクルと少なく，ブロック LU 分解においては行列積差演算の計算時間（ブロックサイズ 32 × 32 で約 8,000 サイクル，64 × 64 で約 66,000 サイクル）に十分埋め込むことができる．

3. ブロック LU 分解アルゴリズム

LU 分解とは正則行列 $\mathbf{A}$ を下三角行列 $\mathbf{L}$ と上三角行列 $\mathbf{U}$ の積 $\mathbf{LU}$ に分解する操作である．ブロック LU 分解アルゴリズムは正則行列 $\mathbf{A}$ をいくつかのブロック $\mathbf{A}_{ij}$（$0 \le i < n$, $0 \le j < n$）に分割し，それぞれのブロックに対応する $\mathbf{L}_{ij}$ と $\mathbf{U}_{ij}$ を並列に求めるアルゴリズムである．ブロック LU 分解中のほとんどの計算は行列積差演算による要素の更新，

$$\mathbf{A}_{ij} \leftarrow \mathbf{A}_{ij} - \sum_{k=0}^{\min(i,j)-1} \mathbf{L}_{ik}\mathbf{U}_{kj}$$

であり，行列積差演算の最適化が性能向上の鍵となっている．行列積差演算の代表的な計算手順として left-looking 法と right-looking 法の 2 つが知られている．**図 1** (A)，(B) に right-looking 法と left-looking 法について，それぞれの疑似コードと更新パターンを示す．Right-looking 法は同時に行う更新範囲を広くすることで並列性を高めているのに対して，left-looking 法は更新範囲を 1 列ごとに絞ることで参照局所性を高めている．Left-looking 法ではこの 1 列分のデータをソフトウェアキャッシュに格納することでデータ転送量を減らすことができ，さらに 1 列分の更新においてブロックを数個ずつにまとめて行うことで参照局所性が高まる（図 1 (C)）．

図 2 に Cell/B.E. 上の right-looking 法によるブロック LU 分解の計算手順を示す．left-looking 法では更新処理の並列性が right-looking 法に比べて低いため動作しない SPE の割合が多くなるが，前処理と更新が交互に行われるのは right-looking 法と同様である．前処理では軸選択，1 ブロック列内のみの行要素交換および行列単位にならない細かい積差演算による行列要素の更新を実行する．行要素交換の残りは行列積差演算の更新用データ転送の合間に実行する．

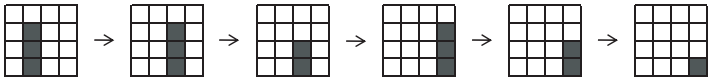
従来の分散並列処理環境では left-looking 法はタスク終了時の同期通信回数の多さがネックになり不利であったが，Cell/B.E. においては SPE 間通信が高速で通信時間は無視できるほどに短いため left-looking 法が不利とはいえない．Right-looking 法ではループカウンタ k の更新ごとにタスク終了のシグナル通知を行うため，SPE 間通信回数のオーダは $O(n)$ である．Left-looking 法では U のブロック単位の分解が終わるごとにタスク終了のシグナル通知を行うため，SPE 間通信回数のオーダは $O(n^2)$ である．どちらにしても LU 分解の

(A)

```
for(k = 0; k < n; k++) {
  for(j = k+1; j < n; j++)
    a[k][j] = a[k][j] / a[k][k];
  for(i = k+1; i < n; i++)
    for(j = k+1; j < n; j++)
      a[i][j] = a[i][j] - a[i][k] * a[k][j];
}
```

(B)

```
for(j = 0; j < n; j++) {
  for(k = 0; k < j; k++) {
    a[k][j] = a[k][j] / a[k][k];
    for(i = k+1; i < n; i++)
      a[i][j] = a[i][j] - a[i][k] * a[k][j];
  }
}
```



(C)

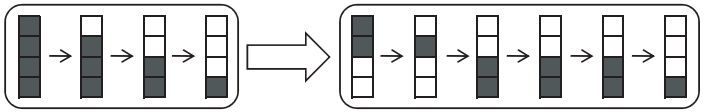


図 1　(A) Right-looking 法の疑似コードと更新パターン，(B) Left-looking 法の疑似コードと更新パターン，(C) Left-looking 法における単純な 1 列分の更新パターンと参照局所性をより高める更新パターン

Fig. 1　(A) Pseudo-code and update pattern of the right-looking method, (B) Pseudo-code and update pattern of the left-looking method, (C) Update pattern of the left-looking method and localized update pattern of the left-looking method.

計算量のオーダ $O(n^3)$ には届かず，かつ Cell/B.E. においてはタスク終了時のシグナル伝達における SPE 間通信時間が 300 サイクル程度であり，通信時間は計算時間に比べ無視できるほど短い．

4. LU 分解性能予測

LU 分解時の行列積差演算と前処理の浮動小数点演算数と計算時間をそれぞれ算出し，合計することで計算性能 P [flops] を予測する．LU 分解する行列のサイズが十分大きければ

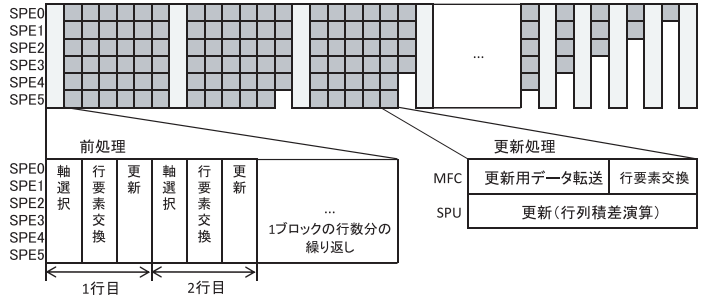


図 2　Cell/B.E. 上の right-looking 法によるブロック LU 分解の計算手順

Fig. 2　Process of Block LU decomposition with right-looking method on Cell/B.E.

計算時間のほとんどは行列積差演算に集中し，行列積差演算性能 P_{max} で予測性能に代えればよいことを後の評価で示す．なお性能予測モデルでは計算精度を問わず適用可能であるが Cell/B.E. が単精度前提で動作するため性能予測モデル中の浮動小数点演算は単精度とする．

まず，予測式と予測の手順を示し，後に right-looking 法と left-looking 法のそれぞれについて詳細を述べる．予測式を次に示す．

$$S_{LSr} \geq S_{code} + 24S_{br}^2 \tag{1}$$

$$S_{LSl} \geq S_{code} + (6 + n_{cache}) \times 4S_{bl}^2 \tag{2}$$

$$P_{maxr} = \min\left(\frac{T_{mem}S_{br}}{6}, P_{cell}\right) \tag{3}$$

$$P_{maxl} = \min\left(\frac{T_{mem}S_{bl}}{2}, P_{cell}\right) \tag{4}$$

$$P = \frac{C}{t_{ml} + t_{pv\alpha} + \epsilon} \tag{5}$$

$$t_{ml} = \frac{c_{ml}}{p_{ml}P_{max}} \tag{6}$$

$$t_{pv\alpha} = \frac{c_{pv\alpha}}{p_{pv\alpha}P_{pv\alpha}} \tag{7}$$

$$C = c_{ml} + c_{pv\alpha} \tag{8}$$

$$c_{ml} = \frac{1}{3}S_b^3 n_b(n_b - 1)(2n_b - 1) \tag{9}$$

$$c_{pv\alpha} = \left(S_b^3 - \frac{1}{2}S_b^2\right)n_b^2 - \left(\frac{1}{3}S_b^3 + \frac{1}{6}S_b\right)n_b \tag{10}$$

それぞれの式の意味は次のとおりである．まず，right-looking 法と left-looking 法の 2 つ

のアルゴリズムが実装可能なアーキテクチャ上の条件を示すのが式 (1)，(2) である．ここで実装可能とする基準はデータ転送時のダブルバッファリングが可能なこと，left-looking 法では加えて計算に必要なデータがすべてコア専用メモリ内のソフトウェアキャッシュに格納可能なこととする．ダブルバッファリングやソフトウェアキャッシュはアルゴリズムの実装として必須ではないが，なければ極端に性能が落ちるため，本論文では必ず実装することを前提にする．実装可能かどうかは right-looking 法はローカルストア容量 S_{LS} [Byte] とブロックサイズ $S_b \times S_b$，left-looking 法はこれらと必要なソフトウェアキャッシュのブロック数 n_{cache} によって判断できる．Right-looking 法は式 (1)，left-looking 法は式 (2) で判断し，不等式を満たすならば実装可能である．ここで S_{code} [Byte] はコードや他のデータの格納に必要な容量である．また，変数の添字の最後の文字は r が right-looking 法，l が left-looking 法の性能予測に使う変数であることを表す．

次に式 (3)，(4) が行列積差演算性能予測を表す．予測に必要なパラメータはブロックサイズに加えて SPE 単体の理論性能 P_{SPE} [flops]，SPE の数 n_{SPE}，メモリアクセススループット T_{mem} [Byte/s] である．ここで SPE 単体の理論性能は 25.6 Gflops とし，Cell/B.E. の理論性能は $P_{cell} = P_{SPE} \times n_{SPE}$ とする．行列積差演算性能はデータ転送速度と計算速度のうち遅い方によって決まる．ダブルバッファリングを施した後ならば，データ転送が計算よりも遅い場合はデータ転送の裏で計算が終わるため性能はデータ転送速度によって決まる．逆に計算がデータ転送よりも遅い場合は計算の裏でデータ転送が終わるため性能は計算速度によって決まる．これを表したのが right-looking 法では式 (3)，left-looking 法では式 (4) である．min 関数内の左側がデータ転送速度からみた性能，右側が計算速度からみた性能である．

さらに詳細な予測では前処理の過程も含め，式 (5) から (10) で表す．行列の次数を S_{prob} とする．軸選択は部分軸選択によるものとすると，行列積差演算の計算量のオーダが $O(S_{prob}^3)$ であるのに対して前処理の計算量のオーダは $O(S_{prob}^2)$ のため，特に S_{prob} が大きいときには影響が少ない．予測式中の t_{ml} と $t_{pv\alpha}$ および c_{ml} と $c_{pv\alpha}$，p_{ml} と $p_{pv\alpha}$ はそれぞれ行列積差演算と，軸選択および関連するブロック更新を含む前処理にかかる時間および浮動小数点演算数，並列化効率である．$P_{pv\alpha}$ は軸選択および関連するブロック更新を含む前処理における浮動小数点演算の計算効率である．ϵ は問題情報の転送，SPE 間通信の準備，計算終了のシグナル伝達，時間計測にかかる時間を表す．

以下では浮動小数点演算数と，right-looking 法と left-looking 法それぞれの実装可能な条件および行列積差演算性能の予測を述べる．

4.1　浮動小数点演算数

計算性能を算出するために LU 分解に必要な浮動小数点演算数を求める．行列積差演算と前処理に必要な浮動小数点演算数をそれぞれ c_{ml}，$c_{pv\alpha}$ は right-looking 法，left-looking 法ともに同数で次の式で表される．

$$c_{ml} = \frac{1}{3} S_b^3 n_b (n_b - 1)(2n_b - 1) \tag{11}$$

$$c_{pv\alpha} = \left(S_b^3 - \frac{1}{2} S_b^2 \right) n_b^2 - \left(\frac{1}{3} S_b^3 + \frac{1}{6} S_b \right) n_b \tag{12}$$

まず，行列積差演算中の浮動小数点演算数 c_{ml} を right-looking 法で考える．1 行に並ぶブロック数を n_b とすると，k_b 回目（$0 \le k_b < n_b$）のループでは，右下のブロック $(n_b - k_b - 1)^2$ 個について行列積差演算を行う．よって合計の行列積差演算数は

$$\sum_{k_b=0}^{n_b-1} (n_b - k_b - 1)^2 = \frac{1}{6} n_b (n_b - 1)(2n_b - 1)$$

である．ブロックサイズを S_b とすると，1 回の行列積差演算中に浮動小数点演算を $2S_b^3$ 回行う必要がある．よって行列積差演算での浮動小数点演算数の合計 c_{ml} は式 (11) に示すとおりとなる．

次に前処理における更新処理の浮動小数点演算数 $c_{pv\alpha}$ を求める．部分軸選択を行うには要素更新後の値が必要なため，S_b 行分の部分軸選択を終わらせるためには S_b 行分をブロック分割せずに分解するときと等しい回数の浮動小数点数演算が必要である．k_b 回目（$0 \le k_b < n_b$）の更新の計算量を考える．対角ブロックでは，行列サイズが S_b の LU 分解と同数の浮動小数点演算を行うことになる．すなわち k 回目（$0 \le k < S_b$）のループでは $k-1$ 回の除算と $(k-1)^2$ 回の積差演算を行うことになり，対角ブロックの浮動小数点演算数 c_{diag} は，

$$c_{diag} = \sum_{j=0}^{S_b-1} \{(k-1) + 2(k-1)^2\} = \frac{2}{3} S_b^3 - \frac{1}{2} S_b^2 + \frac{1}{6} S_b$$

である．右側のブロック $(n_b - k_b - 1)$ 個分の計算では，各要素を対角ブロック中の同じ行の対角成分で除算するとともに，i 行目（$0 \le i < S_b$）の要素は i 回の積差演算を行う．よって 1 ブロックあたりの浮動小数点演算数 c_{right} は

$$c_{right} = S_b^2 + \sum_{i=0}^{S_b-1} 2S_b i = S_b^3$$

である．下側のブロック $(n_b - k_b - 1)$ 個分の計算では，ブロック内の j 列目（$0 \le j < S_b$）の要素は j 回の積差演算を行う．1 ブロックの浮動小数点演算数 c_{under} は

$$c_{under} = \sum_{j=0}^{S_b-1} 2S_b j = S_b^3 - S_b^2$$

である．これらを合計すると次のようになる．

$$\begin{aligned} c_{pv\alpha} &= \sum_{k_b=0}^{n_b-1} \{c_{diag} + (n_b - k_b - 1)(c_{right} + c_{under})\} \\ &= n_b \left(\frac{2}{3}S_b^3 - \frac{1}{2}S_b^2 + \frac{1}{6}S_b\right) + \frac{1}{2}(n_b^2 + n_b)(2S_b^3 - S_b^2) \\ &= \left(S_b^3 - \frac{1}{2}S_b^2\right) n_b^2 - \left(\frac{1}{3}S_b^3 + \frac{1}{6}S_b\right) n_b \end{aligned}$$

このように前処理における浮動小数点演算数 $c_{pv\alpha}$ は式 (12) に示すとおりとなる．

4.2 Right-looking 法の行列積差演算性能予測

まず，right-looking 法が実装可能かどうかを判断する式を次に示す．

$$S_{LSr} \ge S_{code} + 24S_{br}^2 \tag{13}$$

行列積差演算は更新領域 **A** と下三角行列成分 **L**，上三角行列成分 **U** から $\mathbf{A} = \mathbf{A} - \mathbf{LU}$ を行うものである．よって作業領域として必要なローカルストア容量はダブルバッファリングの実装を考えるとブロック 6 個分となる．ブロックサイズは $S_b \times S_b$，データは 4 バイトの単精度浮動小数点数であるから 1 ブロックあたり $4S_b^2$ バイトのデータ量となる．よって，コードや他のデータに必要な容量 S_{code} を含めると式 (13) の右辺となる．ローカルストア容量 S_{LSr} はこれ以上であればよいので実装の可否は式 (13) を満たすかどうかで判断できる．

次に Right-looking 法での行列積差演算性能の予測式を示す．

$$P_{maxr} = \min\left(\frac{T_{mem}S_{br}}{6}, P_{cell}\right) \tag{14}$$

Right-looking 法では 1 回の行列積差演算で **A** の読み書きと **L** または **U** のどちらか一方の読み込みを行うためにメモリアクセスを行う．キャッシュヒット率を高く保つには LU 分解する行列と同等の大きさのストレージがプロセッサ内に必要なため，データをすべてソフトウェアキャッシュに格納することは不可能と仮定した．ブロック 1 つあたりのデータ量は，単精度浮動小数点数 S_{br}^2 個からなるので $4S_{br}^2$ バイトである．行列積差演算 1 回につき 3 ブロック転送するので，データ転送時間は $12S_{br}^2/T_{mem}$ となる．このデータ転送時間と計算時間が一致するときの性能を P_t とすると行列積差演算の浮動小数点演算数は $2S_b^3$ であるから，

$$P_t = 2S_b^3 \times \frac{T_{mem}}{12S_{br}^2} = \frac{T_{mem}S_{br}}{6}$$

である．実行時間の短い方は長い方の実行時間中に終わるため，行列積差演算の実行時間はデータ転送時間と計算時間の遅い方で決まる．つまり，P_t と P_{cell} のどちらか低い方が行列積差演算性能となる．よって行列積差演算性能は式 (14) となる．

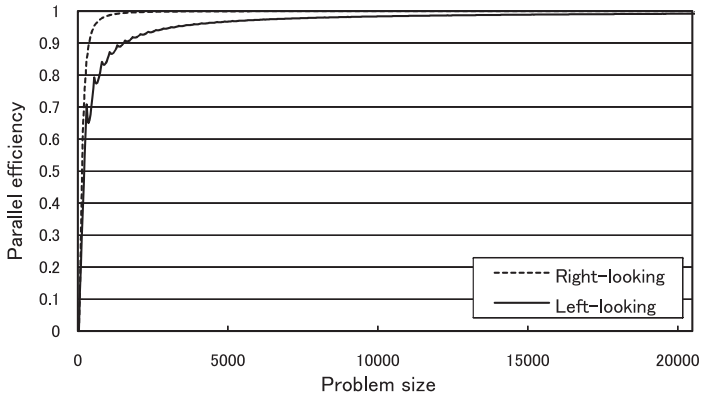


図 3　式 (15)，(18) に基づく left-looking 法と right-looking 法の並列化効率（8 SPEs）
Fig. 3　The difference of parallel efficiency between left and right looking methods based on equation (15) and (18) (8 SPEs).

図 3，**図 4** に right-looking 法，left-looking 法それぞれの行列積差演算の並列化効率の予測を示す．行列積差演算は計算量が固定のため，静的スケジューリングによりなるべく同じ数のタスクを割り当てるものとする．ブロックサイズは 32×32 で，使用 SPE 数は 8 と 32 の場合である．並列化効率はタスクの数が SPE の数で割り切れないとき，各 SPE に割り当てられるタスクの数に差が生じることで減少する．Right-looking 法では k 回目（$0 \le k < n_b$）のループ中のタスク数は $(n_b - k - 1)^2$ である．合計のタスク数は $n_b(n_b - 1)(2n_b - 1)/6$

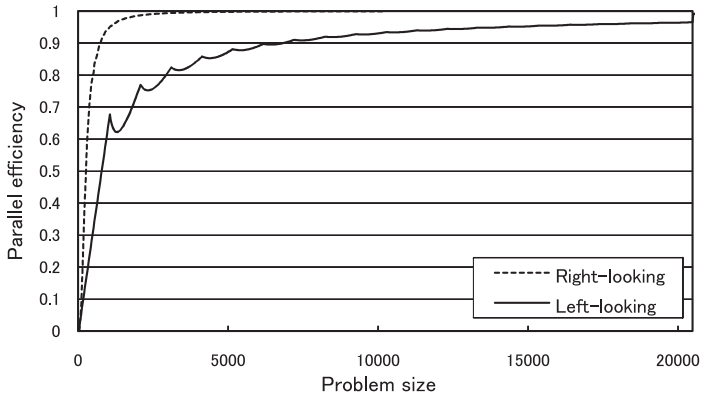


図 4　式 (15)，(18) に基づく left-looking 法と right-looking 法の並列化効率（32 SPEs）
Fig. 4　The difference of parallel efficiency between left and right looking methods based on equation (15) and (18) (32 SPEs).

であるから並列化効率 E_r は次のようになる．

$$E_r = \frac{n_b(n_b-1)(2n_b-1)}{6n_{SPE}\sum_{k=1}^{n_b-1}\lceil k^2/n_{SPE}\rceil} \tag{15}$$

Right-looking 法は並列化効率が良く，行列サイズがおよそ 2,000 以上ならば SPE の数が 32 基になっても並列化効率はほぼ 100%となる．

4.3　Left-looking 法の行列積差演算性能予測

まず，left-looking 法が実装可能かどうか判断する式を示す．

$$S_{LSl} = S_{code} + (6 + n_{cache}) \times 4S_{bl}^2 \tag{16}$$

作業領域は Right-looking 法と同じく **A** と **L**，**U** のダブルバッファリングを行うために 6 ブロック分のデータ領域が必要である．Left-looking 法ではそれに加えてソフトウェアキャッシュをローカルストア内に構築する必要がある．ソフトウェアキャッシュ容量をブロック n_{cache} 個分確保すると並列性は次数 n_{cache} の行列を left-looking 法で LU 分解するときに相当し，ブロック 1 列分が完全にソフトウェアキャッシュに収まるときに並列性は最大となる．図 3，図 4 によると各 SPE が 20 ブロック分の領域を担当すれば並列化効率は 95%を超え，これ以降の上昇率は低いので n_{cache} は少なくとも 20 以上あればよいものとする．

次に Left-looking 法での行列積差演算性能の予測式を示す．

$$P_{maxl} = \min\left(\frac{T_{mem}S_{bl}}{2}, P_{cell}\right) \tag{17}$$

Left-looking 法では行列積差演算ごとに毎回読み込むブロックは **L** のみである．**A** の読み書きの合計回数は $2n_b(n_b-1)$，**U** の読み込みの回数は $n_b(n_b-1)/2$ であり，行列積差演算の回数 $n_b(n_b-1)(n_b-2)$ に対して非常に少ない．実際に 128 × 128 のブロック分割（次数 4096 の行列を 32 × 32 のブロックで分割する場合に相当）で計算するときの **A** と **U** の読み書きの回数は **L** の読み込みの回数の 15 分の 1 である．ブロック分割数がより多い場合の割合はさらに減少する．よって left-looking 法は right-looking 法に比べてデータ転送時間は 3 分の 1 としてよい．このとき転送時間と釣り合う計算性能は right-looking 法の場合の 3 倍となる．Right-looking 法と同様に実行時間は計算時間とデータ転送時間のどちらか遅い方となるので行列積差演算性能は式 (17) となる．

Left-looking 法は更新領域が狭いため right-looking 法よりも並列化効率が低下する．ループカウンタ i が値 n（$0 < n < n_b$）のとき，ループ中のタスク数は n であり，ループカウンタ j が値 n になる回数は n 回である．よって並列化効率は次のとおりになる．

$$E_l = \frac{n_b(n_b-1)(2n_b-1)}{6n_{SPE}\sum_{i=1}^{n_b-1}\{\lceil i/n_{SPE}\rceil \times i\}} \tag{18}$$

図 3，図 4 によると Left-looking 法は right-looking 法に比べて並列化効率が悪いが，係数行列が十分大きいときには並列化効率は 100%に近く，性能に大きな影響を与えることはない．

5. 性 能 評 価

Cell/B.E. 上の LU 分解アルゴリズムで実装可能なのはブロックサイズ 32 × 32 の left-looking 法と，ブロックサイズ 32 × 32 および 64 × 64 の right-looking 法である．これらすべてについて性能予測モデルと実測性能を比較する．

図 5 にブロックサイズ 32 × 32 の left-looking 法で，**図 6** にブロックサイズ 32 × 32 の right-looking 法で，**図 7** にブロックサイズ 64 × 64 の right-looking 法で LU 分解したときの行列積差演算性能予測と実測性能を示す．どのアルゴリズムも係数行列の大きさは 4096 × 4096 である．ブロックサイズ 32 × 32 の left-looking 法と right-looking 法を実行した計算機はリファレンスセットで，行列積差演算については理論性能の 97%の効率を達成している．ブロックサイズ 64 × 64 の right-looking 法の実測性能は文献 6) から引用した．メモリアクセススループットは理論性能と実測性能とでは大きく異なるため，予測性能

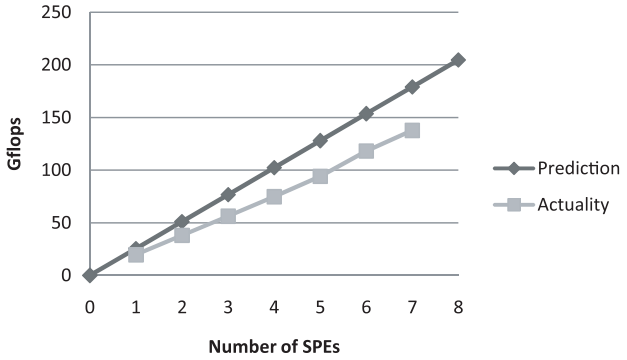


図 5　予測性能（前処理補正なし）と実測の違い（ブロックサイズ 32 × 32 の left-looking 法）
Fig. 5　The difference between prediction performance and real performance without considering preprocessing overhead (32 by 32 block-partitioned left-looking method).

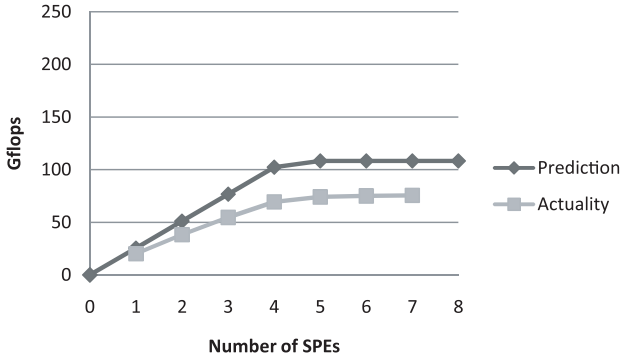


図 6　予測性能（前処理補正なし）と実測の違い（ブロックサイズ 32 × 32 の right-looking 法）
Fig. 6　The difference between prediction performance and real performance without considering preprocessing overhead (32 by 32 block-partitioned right-looking method).

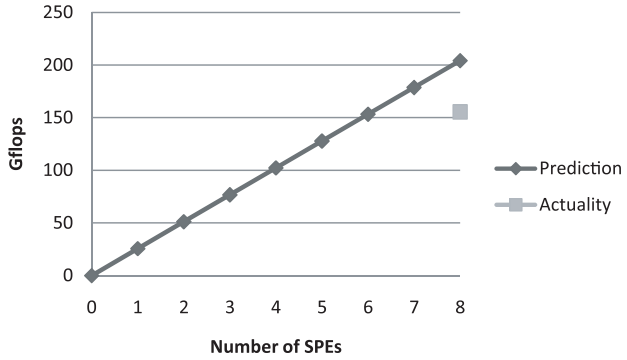


図 7　予測性能（前処理補正なし）と実測の違い（ブロックサイズ 64 × 64 の right-looking 法）
Fig. 7　The difference between prediction performance and real performance without considering preprocessing overhead (64 by 64 block-partitioned right-looking method).

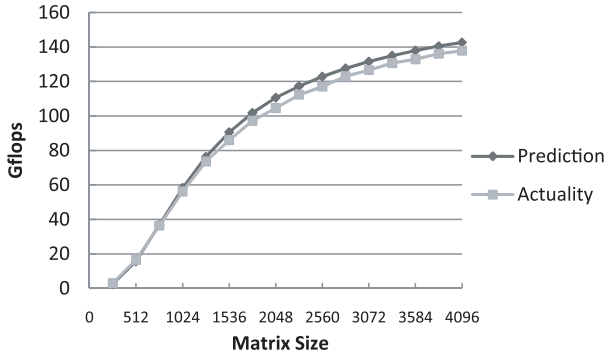


図 8　予測性能（前処理補正あり）と実測の違い（left-looking 法 SPE 7 基時）
Fig. 8　Comparison between predicted performance and actual performance with considering preprocessing overhead (left-looking method on seven SPEs).

の計算では DMA 転送の put 性能を 25 GB/s，get 性能を 18 GB/s とした[7]．left-looking 法による実装において，SPE 7 基を使用し理論性能の 76.9% にあたる 137.8 Gflops を達成した．なお，使用したプログラムにおける n_{cache} は 20 であり 1 列分のブロックを格納するために十分な容量である．一方，right-looking 法による実装では SPE 数が 5 基以上のときにメモリアクセスボトルネックが発生し，SPE 7 基使用時の性能は理論性能の 42.2% にあたる 75.6 Gflops となった．

予測性能と実測性能を比べると，実測性能は予測性能の 70% 前後となっている．これは行列積差演算性能のみを考えたからである．前処理を行列サイズ 4096，SPE 7 基使用時に性能測定すると 11.3 Gflops であり，理論性能よりも著しく低いため前処理の時間が無視できないと分かる．**図 8** に前処理性能と初期化・終了処理にかかる時間を詳細な予測式にあて

はめて得た予測性能と，ブロックサイズ 32 の left-looking 法で様々なサイズの行列を LU 分解したときの実測性能を示す．初期化・終了処理にかかる時間は 256 × 256 の行列の LU 分解を問題とし実際には計算を行わなかったときの時間とした．予測性能が若干実測性能を上回っているが，この差は行列積差演算との合間にある制御や DMA 転送処理によるものと考えられる．これらの処理は行列サイズとブロックサイズが大きくなるほど全体の実行時間に占める割合が減少するため，性能を決める大きな要因は行列積差演算性能であるといえる．

6. LU 分解アルゴリズムの比較

性能予測式を基にプロセッサアーキテクチャ条件によって right-looking 法と left-looking 法のどちらが最適アルゴリズムかを求められる．ここで前処理の実行時間の影響は行列サイズが十分大きい場合は軽微かつアルゴリズムによる差がほとんどないため無視することにし，行列積差演算性能のみで性能を予測する．

はじめにローカルストア容量 S_{ls} からブロックサイズ S_b を決定する．ブロックサイズは大きいほどよく，トレードオフとなるような他のパラメータは存在しないため条件から一意に最適サイズが求められる．式 (19) より right-looking 法でのブロックサイズ S_{br} を，式 (20) より left-looking 法でのブロックサイズ S_{bl} を求める．

$$S_{LS} \geq S_{code} + 24S_{br}^2 \tag{19}$$

$$S_{LS} \geq S_{code} + (6 + n_{cache}) \times 4S_{bl}^2 \tag{20}$$

ここで S_{code} は実行コードやブロック以外のデータがローカルストア内で占める容量で，ここでは経験的に 128 [KiB] とし，n_{cache} は並列化効率が 95%を超える 20 とする．

次にブロックサイズとメモリアクセススループットから right-looking 法と left-looking 法それぞれの行列積差演算における許容性能 P_{right}，P_{left} を予測する．許容性能はデータ転送時間が計算時間を超えない最大の性能である．次の式 (21) より right-looking 法の，式 (22) より left-looking 法の許容性能を求める．

$$P_{right} = \frac{T_{mem} S_{br}}{6} \tag{21}$$

$$P_{left} = \frac{T_{mem} S_{bl}}{2} \tag{22}$$

$P_{right} > P_{left}$ ならば right-looking 法が，$P_{right} < P_{left}$ ならば left-looking 法が適している．P_{right} と P_{left} が同程度ならば制御が簡単な right-looking 法が良い．

図 9 の (A) および (B) にメモリスループットを 25.6 GB 毎秒で固定したときの left-looking 法，right-looking 法それぞれの最適ブロックサイズとその予測性能を示し，図 9 の (C) には (A) と (B) より最高速のアルゴリズムを抜き出した．ブロックサイズを 1 要素単位で決められるならば right-looking 法は SPE 数が少ないときに高速で，left-looking 法は SPE 数が多いときに高速である．ただし，実装上ブロックサイズが 32 の倍数になるため境界付近では最適アルゴリズムがばらつくことになる．また，Cell/B.E. はメモリアクセススループットを向上させなくても，SPE 数をさらに増やして left-looking 法を採用することで高い計算性能を発揮できることを示唆しており，SPE 数が 16 であっても left-looking 法ならば行列積差演算性能 395.8 Gflops を達成できる．

図 10 に (A) left-looking 法と (B) right-looking 法それぞれについて SPE 数とローカル

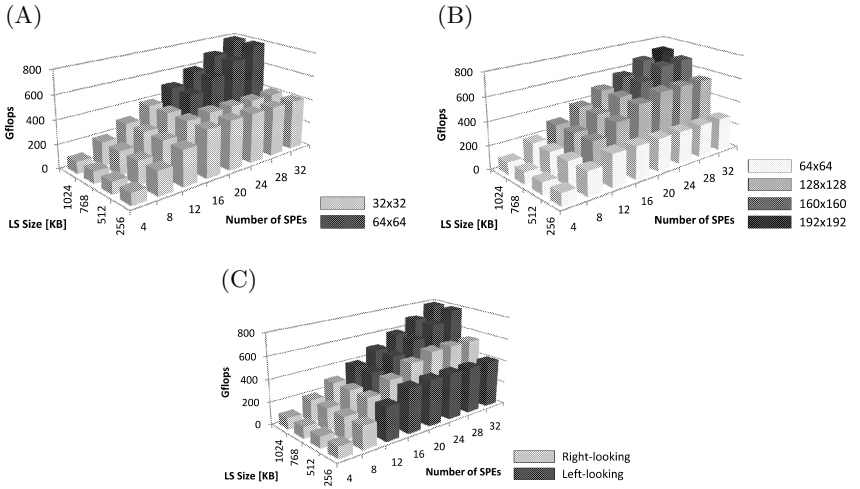


図 9　(A) Left-looking 法におけるアーキテクチャ条件ごとの最適ブロックサイズと予測性能（メモリ転送速度は 25.6 GB/秒に固定），(B) Right-looking 法におけるアーキテクチャ条件ごとの最適ブロックサイズと予測性能（メモリ転送速度は 25.6 GB/秒に固定），(C) 各アルゴリズム（left/right looking 法）が有利なアーキテクチャ条件（LS サイズ/SPE 数）の分布（メモリ転送速度は 25.6 GB/秒に固定）

Fig. 9　(A) Distribution of preferable architectural conditions (LS size and # of SPEs) for the left-looking method when memory data transfer rate is 25.6 GB/s, (B) Distribution of preferable architectural conditions (LS size and # of SPEs) for the right-looking method when memory data transfer rate is 25.6 GB/s, (C) Distribution of preferable architectural conditions (LS size and # of SPEs) for the left/right-looking method when memory data transfer rate is 25.6 GB/s.

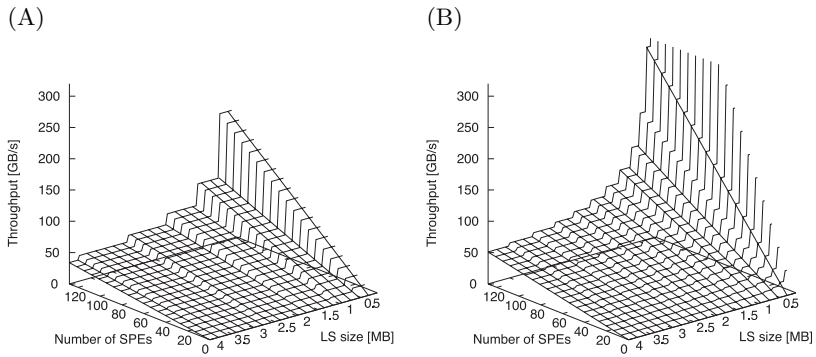


図 10　(A) Left-looking 法で SPE 数を増やしたときに必要なメモリ転送性能とローカルストア容量
(B) Right-looking 法で SPE 数を増やしたときに必要なメモリ転送性能とローカルストア容量

Fig. 10　(A) Required memory throughput and local store size for the left-looking method when changing the number of SPEs.
(B) Required memory throughput and local store size for the right-looking method when changing the number of SPEs.

ストアの容量に対応する必要なメモリスループットの予測を示す．ローカルストアの用途はコードとブロック以外のデータに使われる 128 KB は left-looking 法と right-looking 法ともに共通で，right-looking 法はローカルストアの残りを作業領域に，left-looking 法は作業領域とソフトウェアキャッシュに使用する．ここで left-looking 法のソフトウェアキャッシュの容量は並列性を十分に確保できるブロック 20 個分である．left-looking 法のソフトウェアキャッシュは問題の行列サイズに応じて最適化可能で，行列のブロック 1 列分をソフトウェアキャッシュ容量として設定すると最大の効率でデータ転送ができる．例として**図 11** に行列サイズが 32768 のときについて最適なソフトウェアキャッシュの容量を選んだときの必要メモリスループットの予測を示す．行列サイズに関係なく left-looking 法は right-looking 法の 3 分の 2 程度のメモリスループットで動作し，適切にソフトウェアキャッシュの容量を選ぶことで right-looking 法の 3 分の 1 まで必要メモリスループットを減らすことができる．

このような結果になるのは 2 つのアルゴリズムの特徴から説明できる．right-looking 法の並列性が高い理由はタスク間で同じデータを共有しないからであり，SPE 1 基あたり left-looking 法のおよそ 3 倍のデータ転送性能が必要となっている．逆に right-looking 法はソフトウェアキャッシュにローカルストアの一部を割り当てる必要がないためブロックサイズが left-looking 法より大きくとれる傾向がある．しかし，ブロックサイズが大きくなることに

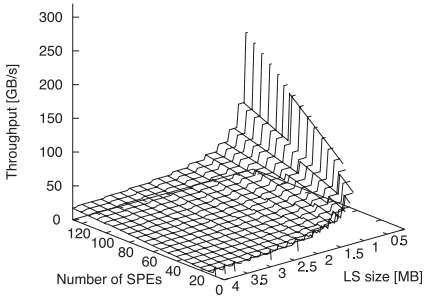


図 11　行列サイズ 32768 に最適化した left-looking 法で SPE 数を増やしたときに必要なメモリ転送性能とローカルストア容量

Fig. 11　Required memory throughput and local store size for the left-looking method optimized for 32768 by 32768 matrices when changing the number of SPEs.

よるデータ転送量の軽減は 3 倍の差を覆すほどではないため，SPE 数の増加に対してメモリアクセスボトルネックが発生しやすい．right-looking 法で性能を伸ばすにはメモリアクセスの大幅な高速化が必要といえる．left-looking 法は並列性こそ right-looking 法に劣るが高い参照局所性によりデータ転送性能をあまり必要としないため，right-looking 法よりも SPE 数増加にともなうメモリアクセスボトルネックが発生しにくい．したがって left-looking 法はメニーコアプロセッサに適しているといえる．

7. まとめと今後の課題

マルチコア，メニーコアプロセッサ上での right-looking 法と left-looking 法それぞれによる LU 分解の予測性能を求め，Cell/B.E. による計測で検証した．Cell/B.E. 上では right-looking 法と left-looking 法の演算性能は同程度であったが，メモリとチップ間のデータ転送量は left-looking 法が right-looking 法の半分程度になった．一般には left-looking 法のデータ転送量は right-looking 法の 3 分の 1 から 3 分の 2 の間に収まると予測される．left-looking 法はデータ転送量削減の効果が高いため left-looking 法が実装できるならば left-looking 法が最適であるといえる．これは left-looking 法の高い参照局所性とマルチコアプロセッサの高速なコア間通信によるものである．今後のメニーコアプロセッサの演算性能とメモリ性能のバランスを考えると最適な LU 分解アルゴリズムは left-looking 法になりつつある．

本論文ではプロセッサコア間の通信速度は一定でコア間の位置関係による差異は無視した．しかしメニーコアプロセッサでは多数のコアを 2 次元に配置する都合から，チップの端

から端までの通信速度が相対的に遅くなる．そのためコアの配置による LU 分解性能の変化が報告されている[8]．したがって right-looking 法と left-looking 法についてはコア配置最適化が与える影響の程度によっては優劣が変わる可能性が残されている．今後はコア配置最適化を含めてメニーコアアーキテクチャの実情に従った解析が必要となっている．

参 考 文 献

1) Geer, D.: Chip Makers Turn to Multicore Processors, *Computer*, Vol.38, No.5, pp.11–13 (2005).
2) Sony Computer Entertainment Inc.: Cell Broadband Engine アーキテクチャVersion 1.01 (2006).
3) Dongarra, J., Gannon, D., Fox, G. and Kennedy, K.: The Impact of Multicore on Computational Science Software, *CTWatch Quarterly*, Vol.3, No.1, pp.3–10 (2007).
4) Anderson, E., Bai, Z., Dongarra, J., Greenbaum, A., McKenney, A., DuCroz, J., Hammerling, S., Demmel, J., Bischof, C. and Sorensen, D.: LAPACK: A portable linear algebra library for high-performance computers, *Supercomputing '90: Proc. 1990 ACM/IEEE conference on Supercomputing*, Los Alamitos, CA, USA, pp.2–11, IEEE Computer Society Press (1990).
5) Toledo, S.: Locality of Reference in LU Decomposition with Partial Pivoting, *SIAM J. Matrix Anal. Appl.*, Vol.18, No.4, pp.1065–1081 (1997).
6) Chen, T., Raghavan, R., Dale, J.N. and Iwata, E.: Cell broadband engine architecture and its first implementation: A performance view, *IBM J. Res. Dev.*, Vol.51, No.5, pp.559–572 (2007).
7) Kistler, M., Perrone, M. and Petrini, F.: Cell Multiprocessor Interconnection Network: Built for Speed, *IEEE Micro*, Vol.26, No.3, pp.10–23 (2006).
8) Venetis, I.E. and Gao, G.R.: Mapping the LU decomposition on a many-core architecture: Challenges and solutions, *CF '09: Proc. 6th ACM conference on Computing frontiers*, New York, NY, USA, pp.71–80, ACM (2009).

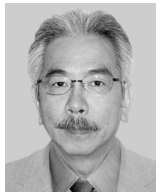
(平成 22 年 1 月 26 日受付)
(平成 22 年 5 月 14 日採録)

里城　晴紀（正会員）

昭和 60 年生．平成 22 年東京工業大学大学院情報理工学研究科計算工学専攻修士課程修了．同年東芝ソリューション（株）入社．Cell Speed Challenge 2008 規定課題部門優勝，Cell Speed Challenge 2008 自由課題部門第 2 位，Cell Challenge 2009 規定課題部門第 3 位．マルチコアプログラミング技法の研究に従事．

吉瀬　謙二（正会員）

昭和 47 年生．平成 7 年名古屋大学工学部電子工学科卒業．平成 12 年東京大学大学院情報工学専攻博士課程修了．博士（工学）．同年電気通信大学大学院情報システム学研究科助手．平成 18 年東京工業大学大学院情報理工学研究科講師．計算機アーキテクチャ，メニーコアプロセッサアーキテクチャ，並列処理に関する研究に従事．電子情報通信学会，IEEE-CS，ACM 各会員．



小長谷明彦（正会員）

昭和 30 年生．昭和 55 年東京工業大学理工学研究科情報科学専攻修士課程修了．同年日本電気株式会社入社．平成 8 年東京工業大学博士（工学）取得．平成 9 年北陸先端科学技術大学院大学知識科学研究科教授．平成 15 年理化学研究所ゲノム科学総合研究センターゲノム情報科学研究グループプロジェクトディレクタ．平成 21 年東京医科歯科大学情報処理センター特任教授．同年 10 月より東京工業大学大学院総合理工学研究科教授．バイオ情報学，グリッド計算の研究に従事．著書に『バイオ情報学：パーソナルゲノム解析から生体シミュレーションまで』（コロナ社）等 4 冊．人工知能学会，ソフトウェア科学会，電子情報通信学会，日本シミュレーション学会，分子生物学会，CBI 学会各会員．