ソフトウェア自動チューニングにおける標本点逐次追加型性能パラメータ推定法の疎行列計算への適用

田中輝雄[†,☆] 片桐孝洋[†,☆☆] 弓場敏嗣[†,☆☆☆]

最低限の数の標本点を用いた推定からはじめて，必要な標本点を選択し追加しながら，最適な性能パラメータの値を推定する「標本点逐次追加型性能パラメータ推定法」を実行時ソフトウェア自動チューニングに適用した．実行時に行列上の非零要素の位置が決定する疎行列計算に対し，性能パラメータとして疎行列のブロック化を行うときのブロック化サイズを取り上げ，実機を用いた実験を行った．その結果，(1) 実測した複数の形状の疎行列の平均で，計算機ごとに 1.07 倍から 2.23 倍のブロック化の効果を得た．(2) 提案手法の振舞いを分析し，提案手法は局所最適化でなく大域的探索を行うことが分かった．(3) 提案手法を用いて新たに必要となる推定時間は，疎行列の行列ベクトル計算処理時間と比較して，数%以下に抑えられ無視できることを確認した．

An Incremental Parameter Estimation Method for Software Automatic Performance Tuning Applied to Sparse Matrix Computation

Teruo Tanaka,[†,☆] Takahiro Katagiri[†,☆☆] and Toshitsugu Yuba[†,☆☆☆]

In this study, an Incremental Parameter Estimation Method is applied to software automatic performance tuning at run-time. In the method, the estimation is started from the least sampling points, and the sampling points are incremented dynamically to improve accuracy. For the evaluation of the method, it was applied to sparse matrix computation to estimate block size for sparse matrix structure as a performance parameter. The results of the evaluation showed: 1) effects of optimized block sizes were between 107% and 223% according to computers, 2) the method behaved global optimization rather than local optimization for parameter estimation and 3) the execution time required for the parameter estimation was only 1/1000 in average of the time required for sparse matrix computation.

1. はじめに

大規模行列計算に代表される科学技術計算分野においては，専門知識を駆使して実装した数値計算ライブラリが広く利用されている．計算環境，すなわち使用する計算機システムおよび対象とするユーザプログラムに応じて，数値計算ライブラリを自動的に最適に設定する，いわゆる「ソフトウェア自動チューニング」の研究が PHiPAC[1]，ATLAS[2]，FIBER[3] などにみられるように近年さかんになってきている．

ソフトウェア自動チューニングの目的は，数値計算ライブラリを計算環境に合わせて最適化し，その能力を最大限に引き出すことにある．

このソフトウェア自動チューニングを実行するタイミングとして，次の 2 つがある．

(1) インストール時自動チューニング：数値計算ライブラリを計算機にインストールするとき
(2) 実行時自動チューニング：対象とする問題を解くために，ユーザプログラムを介して数値計算ライブラリを実行するとき

ソフトウェア自動チューニングを実現するために，まず，数値計算ライブラリの性能チューニング項目を

† 電気通信大学大学院情報システム学研究科
Graduate School of Information Systems, The University of Electro-Communications
☆ 現在，日立超 LSI システムズ
Presently with Hitachi ULSI Systems Ltd.
☆☆ 現在，東京大学情報基盤センタースーパーコンピューティング部門
Presently with Supercomputing Division, Information Technology Center, The University of Tokyo
☆☆☆ 現在，電気通信大学名誉教授
Presently with Professor Emeritus, The University of Electro-Communications

パラメータ化する．次に，性能パラメータのとりうる値から複数の標本点を選択し，それら標本点ごとの値を設定した数値計算ライブラリを実行し，得られた複数の実測データをもとに，実行時間を最小にする性能パラメータの最適値を推定する．

この性能パラメータ推定に対して，筆者らは，最低限の標本点からはじめて，必要な標本点を選択し追加しながらコスト定義関数☆を順次更新し，最適値を推定する “標本点逐次追加型性能パラメータ推定法” を提案した[5),7)]．さらに，文献 5), 7) では，インストール時自動チューニングへの適用を行い，その有効性を示した．

一方，たとえば疎行列を扱うような問題においては，実際にユーザプログラムを実行するまで行列の構造が決定しないため，インストール時自動チューニングは適用できない．このような問題に対しては，ユーザプログラムの実行時に自動チューニングを行う必要がある．実行時自動チューニングの特徴を次に示す．

(1) 問題の規模（行列サイズ）や問題の特性（疎行列の行列の構造）などの情報が確定しているため，性能パラメータの推定範囲を狭めることができる．
(2) ユーザプログラムの実行時に行うため，自動チューニングに要する時間がそのままユーザプログラムの実行時間の増加につながる．

この実行時自動チューニングにおいて，効率的な性能パラメータの推定を行うことが，本研究の目的である．

本論文では，2 章で，標本点逐次追加型性能パラメータ推定法について説明する．3 章では，対象とする疎行列計算について定義する．4 章で，疎行列計算を対象に実証実験を行い，5 章でまとめる．

2. 標本点逐次追加型性能パラメータ推定法[5),7)]

2.1 性能パラメータの推定方法に対する課題

通常，実測データに基づくパラメータ推定では，事前に与えられた標本点における実測データだけを用いる．それらの実測データをもとに，実測していないパラメータのとりうる値を含めて，コスト定義関数を用いて最適なパラメータ値の推定を行う．しかし，ソフトウェア自動チューニングにおいては，（時間が許せば）どの性能パラメータのとりうる値でも標本点として実測することができる．したがって，最初から実測データを揃えるのではなく，まず，最低限のコスト定義関数の形状を表現するのに必要な数の実測データから推定を開始し，実測データが十分であるか否かを判定する．もし十分でなければ，実測すべき標本点を逐次追加する．この場合，繰返し推定のための判定基準や，標本点を追加するたびに必要となるコスト定義関数の再計算時間など，次の 2 つの課題を解決する必要がある．

(1) コスト定義関数の選択
(2) 標本点を動的に追加する方法を用いるときの 2 つの基準
  (a) 標本点を追加するか否かの終了判定基準
  (b) 標本点を追加するときの選択基準

2.2 コスト定義関数 d-Spline の導入

性能パラメータのとりうる値ごとの数値計算ライブラリの実行時間で構成されるコスト定義関数の関数形は，滑らかさとか凸性は保証されない．また，標本点を追加するたびに再計算する必要がある．そこで，データの動きに柔軟に追随する “柔らかさ” を持ち，さらに，標本点が少なくても安定に解が得られ，かつ，計算量の少ないコスト定義関数として，近似関数 $f(x)$ を n 個の離散点 x_j 上の値 $f_j = f(x_j)$，$1 \leqq j \leqq n$ で表現する．つまり，$f = (f_1, f_2, f_3, \ldots, f_j, \ldots, f_n)^t$，t は転置を示す．ここで x_j は等間隔とする（$x_1 < \ldots < x_j < \ldots < x_n$）．滑らかさを出すために，$n$ は性能パラメータのとりうる値の数 N より十分大きくとる（$N < n$）．いま N 個の中から k 個の標本点の実測データがとられているとする（$k \leqq N$）．この実測データを y_i（$1 \leqq i \leqq k$）とする．$y = (y_1, y_2, y_3, \ldots, y_i, \ldots, y_k)^t$，t は転置を示す．

近似関数 f と k 個の実測データの集合 y および性能パラメータのとりうる値との関係を**図 1** に示す．図において，横軸は，近似関数 f の定義域 x_j（$1 \leqq j \leqq n$）である．その一部が性能パラメータのとりうる N 個の値であり，図中の□で囲んだ x_j となる．□のうち

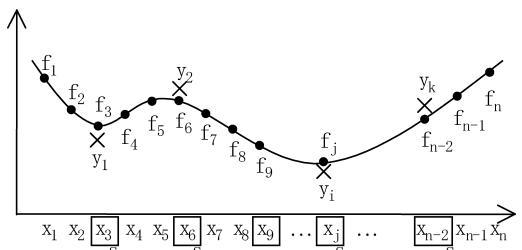


図 1 近似関数 f と実測データの集合 y の関係
Fig. 1 Fitting function f and experimental data y.

☆ コスト定義関数とは，パラメータのとりうる値ごとのコスト（本論文では，数値計算ライブラリの実行時間となる）により，コスト全体の形状を表す近似関数である．

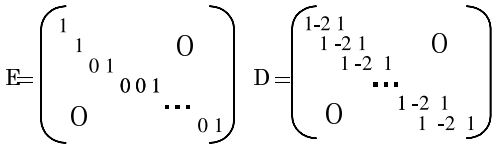


図 2 行列 E，D の構造
Fig. 2 Structure of matrices E and D.

の□$_s$ で示した k 個の x_j が標本点である．縦軸は，f の形状と，標本点に対する実測データ y_i であり，単位は実行時間となる．すべての□で囲んだ x_j が標本点にはならない．たとえば，図 1 では x_9 に対して実測データ y_i は存在していない．

y だけでは y_i より f_j の数の方が多いので，f を規定するために，f の滑らかさを式 (1) で表す．

$$|f_{j-1} - 2f_j + f_{j+1}|^2, 2 \leq j \leq n-1. \quad (1)$$

この近似関数 f を評価関数

$$\min_f(||y - Ef||^2 + \alpha^2||Df||^2) \quad (2)$$

を最小にするように選ぶ．ここで，E，D は**図 2** に示す行列であり，それぞれ $k \times n$，$(n-2) \times n$ のサイズである．式 (2) の第 1 項はデータ y と近似関数 f との距離であり，第 2 項は近似関数 f の滑らかさを表す．スカラ値 α により第 2 項の近似関数の滑らかさが決まる．ここでは，近似関数の形状ではなく，数値計算ライブラリの実行時間を最小とする最適な性能パラメータの値を見つけることが目的である．したがって，できるだけ実測データに追随するように，α の値は十分小さく設定する☆．近似関数 f は微分連続性を保障していないので，離散型のスプラインとして d-Spline と呼ぶ．この d-Spline をコスト定義関数として用いる．

コスト定義関数 d-Spline は $O(n)$ の計算量で求めることができる[6]．一方，標本点を決め，その標本点での実行時間を求める数値計算ライブラリは行列計算なので，行列サイズを L とすると，$O(L^2)$ あるいは (L^3) の計算量となる．一般に行列サイズ L は数百から数千であるのに対して，性能パラメータのとりうる値の数 n はそれより少ないので，コスト定義関数 d-Spline を求める計算量は無視することができる．

2.3 標本点の追加判定と選択基準

今回提案する性能パラメータ推定手法の特徴は，最低限の数の標本点の集合からスタートし，十分な結果が得られるまで標本点を選択し追加していく点にある．まず標本点を追加する選択基準を示す．

選択基準 1 として，推定した最適値がまだ標本点として使われていない場合，その推定した最適値を次の標本点として選択する．推定した最適値が正しければ，その標本点はその後も続けて最適値として推定される可能性が高い．逆に，他の標本点の実測データの影響による間違った推定であれば，その標本点での数値計算ライブラリの実行時間を得ることにより，最適値の候補からはずすことができる．

選択基準 2 として，推定した最適値がすでに標本点の集合に含まれているときは，まだ標本点に選択されていない性能パラメータのとりうる値（図 1 の□$_s$ でない□で囲んだ x_j）の中から $max_j|f_{j-1} - 2f_j + f_{j+1}|$ となる f_j に対応する x_j を標本点として選択する．これは，コスト定義関数 d-Spline の中で変化量が一番大きいところを選び，関数形状をできるだけ大きく変化させるためである．

以上，選択基準 1 および 2 を順に用いて，標本点を選択する．

次に，標本点を追加するか否かの終了判定基準については，コスト定義関数 d-Spline を計算し推定した最適値の安定度を用いる．具体的には，決められた回数の間，連続して推定した最適値が同じであるならば，その値を解とする．この同一最適値推定の連続回数を終了判定基準とする．なお，性能パラメータがとりうる値がすべて標本点になった場合も終了とする．

上記のような終了判定基準を設定することにより，

(1) 変動が激しく推定が難しいケースについては多くの標本点を使用し，
(2) 単調な関数に近いケースについては，比較的少ない標本点で推定が終了する

ように，効率良く判定が作用することがインストール時自動チューニングに対して，実証されている[5),7)]．

3. 実行時ソフトウェア自動チューニングへの標本点逐次追加型性能パラメータ推定法の適用

3.1 実行時ソフトウェア自動チューニングの特徴

実行時ソフトウェア自動チューニングでは，実際にユーザプログラムを実行するときに，性能パラメータの最適化を行う．そのため，ソフトウェア自動チューニングに要する時間は，そのままユーザプログラムの実行時間に反映される．したがって，ソフトウェア自動チューニングに要する時間の抑制が重要となる．

図 3 に自動チューニングを行うユーザプログラムの実行手順を示す．実行時自動チューニングは，ユーザ

☆ 評価では $\alpha = 0.01$ とした．この程度小さければ α の変化による影響はほとんどない．

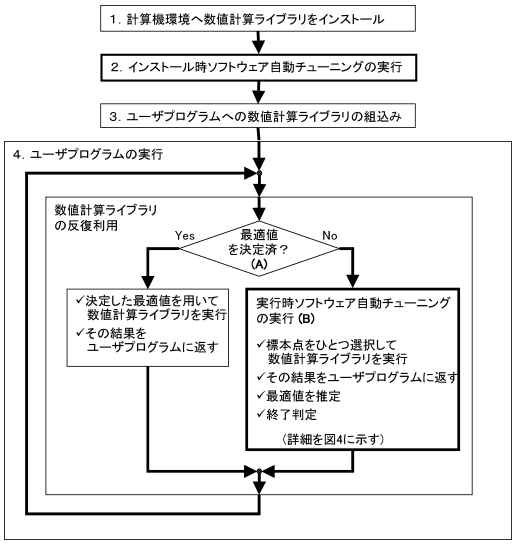


図 3 数値計算ライブラリを用いたユーザプログラムの実行手順
Fig. 3 Execution process of user program using numerical library.

プログラムが繰り返し数値計算プログラムを反復利用する中で行う．具体的には，

(1) 条件分岐 (A) において，性能パラメータの最適値が決定しているかどうかを調べる．

(2) もし，最適値がすでに決定しているのであれば，左側のラインの処理として，“決定した最適値を用いて数値計算ライブラリを実行し，その結果をユーザプログラムに返す” ことを実施する．

(3) 一方，まだ最適値が決まっていない場合は，右側のラインの処理として，実行時ソフトウェア自動チューニングを実行する (B)．ここで，手続き (B) は内部にループ構造を持たないことに注意．最適値が決定するまでは，数値計算ライブラリが呼び出されるたびに手続き (B) を実行する．

手続き (B) の手順を**図 4** を用いて説明する．

(1) 手順 1 から手順 7 により，数値計算ライブラリを実行するための性能パラメータの値を設定する．ここで，*iter_count* は，ユーザプログラムからこの数値計算ライブラリが呼び出された回数を示す．ケース 1 からケース 4 までは，それぞれ第 1 回目から第 4 回目までの呼び出しに対応し，初期値となる 4 つの性能パラメータの値を設定する．この 4 点は，性能パラメータのとりうる値のうち，最小値 xp_1，最大値 xp_N を含む等間隔である．ケース 5 以降は，2.3 節で示した選択基準で選び出された標本点である．

(2) 手順 8 では，設定した性能パラメータの値（標本点）を用いて数値計算ライブラリを実行し，その計算結果をユーザプログラムに返すとともに，数値計算ライブラリの実行時間を計測する．

(3) 手順 10 以降では，計測された 4 点以上の標本点を用いて，以下，最適値の選択を行う．

(4) 手順 11，12 では，ここまでに計測したすべての実測値を用いて，コスト定義関数 d-Spline を計算し，最小となる最適値を推定する．

(5) 手順 13，14 では，2.3 節で設定した終了判定を行う．(a) すべての性能パラメータのとりうる値を用いて実測した，あるいは，(b) 同一の最適値があらかじめ指定した回数 p 回連続して推定した場合は，推定した最適値を数値計算ライブラリの実行時間を最小とする性能パラメータの値であると決定する．以降，ユーザプログラムから数値計算ライブラリが呼び出されるときは，この最適値が用いられ，自動チューニングは行われない．

(6) もし，手順 13 の条件が満たされないときは，手順 16 から手順 21 までの処理が行われる．ここでは，2.3 節で設定した選択基準 1，2 を用い

```
1.  case ( iter_count ) of
2.      1: xp_s = xp_1
3.      2: xp_s = xp_N
4.      3: xp_s = (2xp_1+xp_N)/3)※
5.      4: xp_s = (xp_1+2xp_N)/3)※
6.      5以上: xp_s =xp_new_sp (追加標本点)
7.  end case
8.  パラメタの値xp_sを用いて数値計算ライブラリを実行
    結果をユーザプログラムに返すとともに実行時間を計測
9.  Φsp = Φsp+{xp_s}, Φnonsp = Φnonsp+{xp_s}
10. if ( iter_count≧4 ) then
11.     実測値からコスト定義関数d-Spline f を計算
12.     d-Spline f から数値計算ライブラリの実行時間を
        最小とする最適値xp_opt∈Φを推定
13.     if ( (Φnonsp = {}) or (最適値x_optがp回連続一致) ) then
                                              【終了判定基準】
14.         推定した最適値x_optを数値計算ライブラリ の
            実行時間が最小とするパラメタの値と決定
15.     else
16.         if ( not (xp_opt in Φsp) ) then
17.                 xp_new_sp = xp_opt            【選択基準1】
18.         else
19.             j = max_j || f(xp_j-1)-2f(xp_j)+f(xp_j+1) || (xp_j in Φnonsp)
20.             xp_new_sp = xp_j                  【選択基準2】
21.         end if
22.     end if
23. end if
```

注: ※はΦの中から一番近い要素を選択
- Φは性能パラメタが取りうるN個の値の集合
  (Φ = {xp1, xp2, ..., xpN}, xp1<xp2< ... <xpN)
- Φspは標本点の集合
- Φnonspは標本点以外の性能パラメタが取りうる値の集合 (Φ-Φsp)
- p∈**N**(自然数)はあらかじめ設定した固定値

図 4 標本点逐次追加型性能パラメータ推定法の実行手順
Fig. 4 Algorithm of incremental performance parameter estimation method.

て，次の標本点 xp_{newsp} を決定する．

このときの計算量を考察する．数値計算ライブラリの計算自体は，自動チューニングに関係なく実行されることであり，特に新たに必要になるわけではない．新規に発生するのは，d-Spline の生成時間と xp_{newsp} の探索時間である☆．これらの実測結果は 3.3.5 項に示す．

3.2 実験計算機環境と自動チューニングの対象

3.2.1 疎行列の行列ベクトル積

標本点逐次追加型性能パラメータ推定法の有効性を評価するために，実機を用いて実験を行った．ここでは，ベンチマークとして疎行列の行列ベクトル積を対象とする[8),10),11)]．

一般に疎行列の非零要素のデータの位置情報は実行時にしか分からない．たとえば，回路シミュレーションにおいて，回路の配線構造を行列で表した場合，回路の配線を 1 カ所でも変更すれば，行列の非零要素の位置は変わってしまう．したがって，疎行列を対象とする自動チューニングは実行時に行う必要がある．

疎行列を扱う計算処理では，その *fill-in* を抑え，できるだけスパース性を維持するために，前処理付反復解法が一般によく用いられる．このとき，行列ベクトル積の計算処理が重要となる．

3.2.2 疎行列のブロック化

この行列ベクトル積の計算プログラムに対する性能パラメータとして，疎行列のデータ構造のブロック化を対象とする．本評価では，疎行列のデータ格納形式として，CRS 形式（Compressed Row Storage，圧縮行格納形式）を用いる[4)]．CRS 形式では，行列を行方向に非零要素のみを連続してメモリ上に配置する．データを保持するために，データ自体を保持する配列 *val* とデータの行列の位置を示すための配列（*col_ind*，*row_ptr*）を用いる．式 (3) の非対称行列 A を例として，CRS 形式のデータの保持の仕方を**図 5** に示す．

配列 *val* は行列 A の非零要素の値を行方向に連続に格納する．配列 *col_ind* は対応する配列 val の要素の列インデックスを格納する．すなわち，$val(k) = a_{i,j}$ に対して，$col\_ind(k) = j$ となる．配列 *row_ptr* は配列 *val* 上での行の開始位置を示す．この方法により，行列の格納領域は，$2n_{nonzero} + n + 1$ で済む．ここで，$n_{nonzero}$ は非零要素数である．

CRS 形式では，疎行列の非零要素のみを保持するために，メモリ利用効率が良く，かつ，演算量も少なく済む．しかしながら，すべての要素に対して，間接アドレス付けをするので，演算効率は高くない．

$$A = \begin{pmatrix} 1 & 2 & 0 & 3 \\ 4 & 5 & 6 & 0 \\ 0 & 0 & 7 & 8 \\ 0 & 0 & 0 & 9 \end{pmatrix} \tag{3}$$

val	1	2	3	4	5	6	7	8	9
col_ind	1	2	4	1	2	3	3	4	4
row_ptr	1	4	7	9	10				

図 5 CRS 形式の構造
Fig. 5 Structure of CRS.

$$A = \left(\begin{array}{cc|cc} 1 & 2 & 0 & 3 \\ 4 & 5 & 6 & 0 \\ \hline 0 & 0 & 7 & 8 \\ 0 & 0 & 0 & 9 \end{array}\right) \tag{4}$$

val	1	2	4	5	0	3	6	0	7	8	0	9
col_ind	1	2	2									
row_blk	1	3	4									

図 6 BCRS 形式の構造
Fig. 6 Structure of BCRS.

次に，データをブロック化して保持する BCRS 形式（Block Compressed Row Storage，ブロック圧縮行格納形式）を示す．疎行列が非零要素からなる正方密ブロックの規則的なパターンで構成されているのであれば，小ブロックを基本としてデータを格納することができる．s_{blk} をブロックのサイズ，n_{blk} を非零ブロックの数とすると，全体で必要なメモリ量は $n_{blk} \times s_{blk}^2$ となる．先ほどの非対称行列 A を BCRS 形式で保持すると，**図 6** が得られる．

BCRS 形式では，CRS 形式と同様に，3 つの配列 *val*，*col_ind*，*row_blk* を用いる．配列 *col_ind* は先ほどと異なり，要素単位でなく，ブロック単位に場所を特定する．配列 *val* はブロックごとにまとめて保持する．ここで，もし，ブロック内に零要素がある場合はその場所も確保する．

BCRS 形式により，ブロック内に要素をまとめることができれば，間接アドレスが大幅に削減される．また，ブロック内は連続領域に格納されるので，キャッシュが有効に利用される．一方，データ自体が分散され，ブロック内にまとめることができない場合は，各ブロックに零要素を多くかかえ込むことになり，零要素を保持する無駄なメモリ量が必要になる．さらには，零要素に応じた不要な演算が発生する．このため，実

☆ もともとのデータが CRS 形式で与えられるのか，あるいは，CCS 形式などの別の形式で与えられるか種々のケースがあるので，実験では，疎行列データは実行するためのそれぞれの形式で準備されていると仮定した．

行時間ならびにメモリ容量は，疎行列の非零要素の位置に大きく依存する．

今回の実験では，ブロックサイズ s_{blk} は 2×2 から 16×16 までの 15 通りとした．それぞれ BCRS(2) ないし BCRS(16) と呼ぶことにする．また，基本となる CRS 形式も対象とした．CRS 形式は BCRS 形式でブロックサイズを 1 としたケースと考えられる．

標本点逐次追加型性能パラメータ推定法における終了判定基準は，推定した最適値の同一連続選択数を 2 回から 5 回までの 4 パターンを対象とした．

3.2.3 実験計算機環境

実験に用いた計算機環境として，計算機，数値計算処理，性能パラメータ，疎行列データを**図 7** に示す．

計算機は以下の 4 機種を使用した：東大情報基盤センターのスーパーテクニカルサーバ SR11000[SR11k] ならびに SR8000[SR8k]，京大学術情報メディアセンターのスカラ型スーパコンピュータ PRIMEPOWER HPC2500[Ppower]，研究室の PC サーバ（Pentium4）[PCn1]．今回はすべて非並列とし，1 ノード，1 プロセッサの構成で実施した．

3.2.4 疎行列データ

対象とする疎行列データは文献 9) から入手した．文献 9) では，複数の形式で疎行列が格納されている．その中から CCS 形式（Compressed Column Storage, 圧縮列格納法☆）で保持され，かつ，プログラムの開発環境とした研究室の PC サーバ [PCn1] で実行可能範囲で大規模なデータを対象とした．CCS 形式のデータを CRS 形式に変換し，さらに，CRS 形式から 2×2 から 16×16 までの BCRS 形式に変換して用いた．

使用した 28 種の疎行列データの一覧を**表 1** に示す．各疎行列はそれぞれ特徴のある構造を持っている．

各疎行列データの非零データパターンの一例を**図 8** に示す．図 8 では，色の濃さは非零要素の密集度を示し，空白部分は零要素を表す．ブロック化時にブロックの中に含まれる非零要素の割合の変化を**図 9** に示す．図 9 において，横軸はブロックサイズを示し，縦軸は，そのときに格納された非零要素の割合を示す．つまり，“非零データ数/(ブロックサイズ)^2” を示す．下限のラインは，各ブロックの中に 1 つだけデータが保持された場合，“1/(ブロックサイズ)^2” を示す．したがって，非零要素の割合は，100%と下限のラインの間となる．

たとえば，項番 15 の raefsky3 は 8×8 の小密行列

(1) 実験に用いた計算機とその構成
(a) スーパコンピュータSR11000: [SR11k]
－ コンパイラ:Hitachi Optimized Fortran 90, - sopt
－1ノード, 1プロセッサ

(b) スーパコンピュータSR8000: [SR8k]
－ コンパイラ:Hitachi Optimized Fortran 90 V02-04, - sopt
－1ノード, 1プロセッサ

(c) PrimePower [Ppower]
－IPF
－ コンパイラ:Fujitsu Fortran 90 , - oss
－1ノード, 1プロセッサ

(d) PCサーバ(1ノード, IA32) [PCn1]
－Intel Pentium4(2.0GHz)
－1ノード, 1プロセッサ

－OS:Linux 2.4.9
－コンパイラ:PGI Fortran90 4.0-2 – fast

(2)実験に用いた数値計算処理と性能チューニング評価項目
(a) 疎行列の行列ベクトル積
(b) 疎行列格納形式のブロック化

(3)実験に用いた疎行列データ
(a) Webサイト:
University of Florida Sparse Matrix Collection
より入手した28種

図 7 実験計算機環境と自動チューニングの対象
Fig. 7 Specification of computers, performance parameters and sparse matrices data source.

表 1 疎行列データ
Table 1 Sparse matrix data.

	データ名	問題サイズ	非ゼロ要素数	割合
1	bbmat	38,744	1,771,722	0.12%
2	cage12	130,228	2,032,536	0.01%
3	cage8	1,015	11,003	1.07%
4	circuit1	2,624	35,823	0.52%
5	ex11	16,614	1,096,948	0.40%
6	ex19	12,005	259,879	0.18%
7	ex35	19,716	227,872	0.06%
8	ex40	7,740	458,012	0.76%
9	lhr34	35,152	764,014	0.06%
10	lhr71c	70,304	1,528,092	0.03%
11	li	22,695	1,215,181	0.24%
12	matrix9	103,430	1,205,518	0.01%
13	olafu	16,146	1,015,156	0.39%
14	raefsky1	3,242	294,276	2.80%
15	raefsky3	21,200	1,488,768	0.33%
16	raefsky4	19,779	1,316,789	0.34%
17	raefsky5	6,316	168,658	0.42%
18	raefsky6	3,402	137,845	1.19%
19	rim	22,560	1,014,951	0.20%
20	rma10	46,835	2,374,001	0.11%
21	shyy161	76,480	329,762	0.01%
22	std1Jac2	21,982	1,248,213	0.26%
23	std1Jac3	21,982	1,455,374	0.30%
24	twotone	120,750	1,206,265	0.01%
25	xenon1	48,600	1,181,120	0.05%
26	xenon2	157,464	3,866,688	0.02%
27	ZdJac3	22,835	1,915,726	0.37%
28	ZdJac3db	22,835	713,907	0.14%

☆ CCS 形式は CRS 形式の行と列を入れ替えた格納方法．

をもとに 3 重対角行列を構成している．BCRS 形式で保持する場合，BCRS(2)，BCRS(4) および BRS(8) の構成では，分割したブロックは密な小行列となる．したがって，図 9 において，それぞれ 100%となる．項番 15 の raefsky3 はブロック化効率が高い．項番 13 の olafu は，3 の倍数で効率が良くなっており，基本となる小行列は 6×6 である．一方，項番 3 の cage8 はほとんど下限に張り付いており，ブロック内にはほとんど 1 つしか非零データが存在していないことが分かる．それに対して，項番 20 の rma10 はブロック内にある程度の数の非零要素がかたまって存在しているのが分かる．そのほかに，項番 1 bbmat，項番 6 ex19，項番 17 raefsky5，項番 25 xenon1，項番 26 xenon2 などは BCRS(2)，BCRS(4) などの 2 の倍数，項番 12 matrix9，項番 18 raefsky6 などは 3 の倍数のブロック内非零データ率が高い．

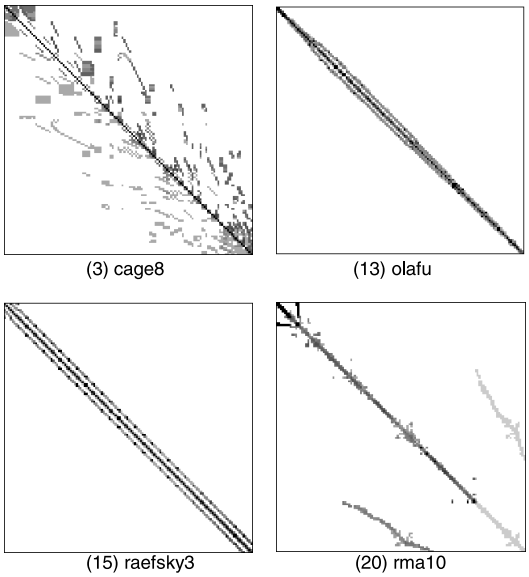


図 8 疎行列データの非零データ位置パターン（例）（文献 9) から転記）
Fig. 8 Example of non-zero elements pattern.

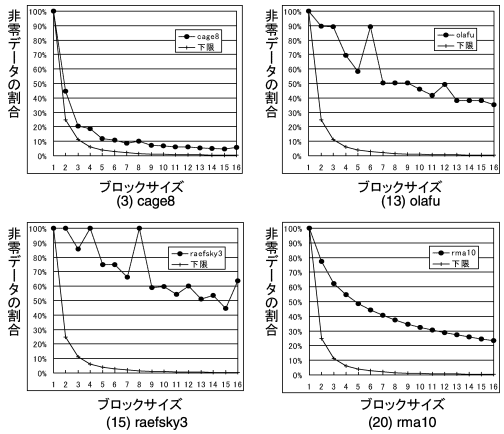


図 9 ブロック化時の非零データ率（例）
Fig. 9 Example of ratio of non-zero elements pattern.

ブロック内非零要素の割合と実際の性能とは相関を持つ．しかしながら，相関の度合いについては，計算機ごとに異なるので，実行時ソフトウェア自動チューニングにより，性能パラメータ推定を行い，疎行列の行列ベクトル積演算の実行時間が一番短いときのブロック幅を探索する．

3.3 実験結果と分析

3.3.1 ブロック化の効果

まず，BCRS 形式を用いたブロック化の効果について示す．評価する疎行列データの全体の特徴を把握するために，CRS 形式ならびに，2×2 から 16×16 までのすべての BCRS 形式に対して，それらに対応する行列–ベクトル積の数値計算プログラムを実行した．その結果得られたそれぞれの疎行列データごとの最適なブロックサイズを**表 2** に示す．表 2 から分かるように，[SR8k] および [SR11k] で 28 パターン中 25 ないし 23 パターンでブロック化の効果が得られた．一

表 2 すべての性能パラメータがとりうる値を用いたときの実験結果
Table 2 Expeliment result using all values of performance parameter.

	データ名	SR11k	SR8k	Ppower	PCn1
1	bbmat	BCRS(4)	BCRS(8)	CRS	CRS
2	cage12	BCRS(2)	BCRS(4)	CRS	CRS
3	cage8	BCRS(2)	BCRS(4)	CRS	CRS
4	circuit1	BCRS(2)	BCRS(4)	CRS	CRS
5	ex11	BCRS(4)	BCRS(4)	BCRS(2)	BCRS(2)
6	ex19	BCRS(2)	BCRS(4)	CRS	CRS
7	ex35	BCRS(4)	BCRS(7)	CRS	CRS
8	ex40	BCRS(4)	BCRS(4)	BCRS(4)	CRS
9	lhr34	BCRS(4)	BCRS(7)	CRS	CRS
10	lhr71c	BCRS(4)	BCRS(7)	CRS	CRS
11	li	BCRS(2)	BCRS(4)	CRS	CRS
12	matrix9	BCRS(3)	BCRS(4)	BCRS(3)	BCRS(3)
13	olafu	BCRS(6)	BCRS(6)	BCRS(6)	BCRS(6)
14	raefsky1	BCRS(2)	BCRS(7)	BCRS(2)	BCRS(2)
15	raefsky3	BCRS(4)	BCRS(8)	BCRS(8)	BCRS(4)
16	raefsky4	BCRS(4)	BCRS(4)	CRS	BCRS(2)
17	raefsky5	BCRS(6)	BCRS(4)	BCRS(4)	BCRS(2)
18	raefsky6	BCRS(3)	BCRS(4)	CRS	BCRS(2)
19	rim	BCRS(2)	BCRS(8)	CRS	CRS
20	rma10	BCRS(4)	BCRS(4)	BCRS(4)	BCRS(2)
21	shyy161	BCRS(4)	CRS	CRS	CRS
22	std1Jac2	CRS	CRS	CRS	CRS
23	std1Jac3	BCRS(2)	CRS	CRS	CRS
24	twotone	CRS	CRS	CRS	CRS
25	xenon1	BCRS(3)	BCRS(6)	BCRS(3)	BCRS(3)
26	xenon2	BCRS(3)	BCRS(6)	BCRS(6)	BCRS(3)
27	ZdJac3	BCRS(2)	CRS	CRS	CRS
28	ZdJac3db	CRS	BCRS(8)	CRS	CRS

表 3 ブロック化の効果（CRS = 1 としたときの倍率）
Table 3 Effect of block structure (CRS = 1).

終了条件	平均	SR11k	SR8k	Ppower	PCn1
連続 4 回	1.45 (97.3%)	1.43	2.23	1.07	1.07
連続 5 回	1.46 (98.0%)	1.44	2.24	1.08	1.08
すべて実測	1.49 (100%)	1.49	2.26	1.10	1.10

表 4 ブロック化の効果（CRS = 1 としたときの倍率）母数として CRS が最適であったケースを除く
Table 4 Effect of block structure (CRS = 1) excluding case of CRS selected.

終了条件	平均	SR11k	SR8k	Ppower	PCn1
連続 4 回	2.35 (97.1%)	1.60	2.71	3.01	2.73
連続 5 回	2.36 (97.5%)	1.61	2.72	3.01	2.74
すべて実測	2.42 (100%)	1.67	2.76	3.09	2.81

方，[Ppower] および [PCn1] ではブロック化の効果は 10 ないし 11 パターンにとどまった．

次に，**表 3** に CRS を 1 としたときのブロック化の効果を示す．ブロック化の効果としては，計算機ごとに，1.10 倍から 2.26 倍という結果となった（“すべて実測” の項を参照のこと）．

ここで，今回用いた標本点逐次追加型性能パラメータ推定法の結果としては，終了判定基準を連続 4 回ないし 5 回とした場合，表 3 に示すように，“すべてを実測” した場合に比べて，平均で 97.3%ないし 98.0%の効果を得る値を選択することができた．

計算機ごとの特性を明確にするために，母数として，CRS が最適であったケースを除いた（すなわち，ブロック化の効果があったケースのみ）場合の効果を**表 4** に示す．表 4 から分かるように，ブロック化の効果が得られたケースの中では，[SR8k]，[Ppower] および [PCn1] では一様に 3 倍前後の効果が得られた．しかしながら，[SR11k] は 1.67 倍にとどまった（この理由については，3.3.3 項で図 11 を用いて後述する）．

以下，標本点逐次追加型性能パラメータ推定法の特性について分析する．

3.3.2 正答率と利用率

図 10 に，標本点逐次追加型性能パラメータ推定法を用いた実験結果を示す．

縦軸の正答率とは，28 種の疎行列に対して，最適な性能パラメータの値が正しく求められた割合を示す．また横軸の利用率とは，連続判定するまでに用いた標本点の数を性能パラメータがとりうる標本値の数で割った値である．利用率はとりうるすべての標本点に対して数値計算ライブラリを実測したときの実行時間を 1 としたときの実行時間の比となる．

図中の各値は，各計算機ごとに 28 種の疎行列を実行した結果の平均値である．各数字は，それぞれの判定基準としての連続数（回）である．また，対角線上の “ランダム” とは，各利用率で，確率的にランダムに標本点を選んだときに得られる正答率との関係を示している．“ランダム” のラインの上側にあることが効率良く最適値を選択できていることを示している．

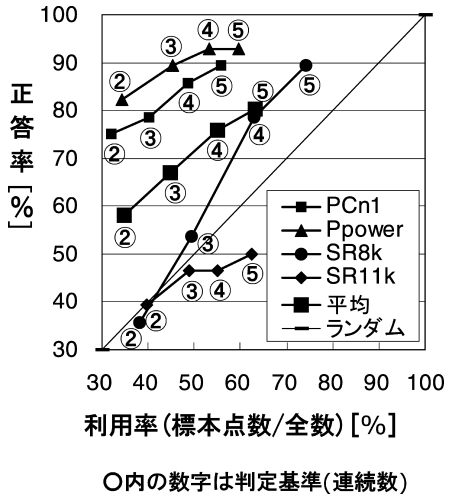


図 10 計算機環境ごとの実験結果（実行時）
Fig. 10 Experiment result of each computers.

全体の傾向を知るために，平均で比較すると，終了判定基準 4 回連続では，正答率 75.9%に対して標本点の利用率 55.0%となった．同じ正答率を得るために「ランダム」では 75.9%の標本点を要すると考えられるので，利用率に関し 20.9 ポイントの効果があるといえる．同様に，終了判定基準 5 回連続では，正答率 80.4%に対して利用率 63.2%となり，「ランダム」に対して，17.2 ポイントの効果がある．最大でみると，終了判定基準 4 回連続では，[Ppower] のケースで，正答率 92.9%を利用率 53.3%の少ない利用率で実現している．“平均” と同じように比較すると，「ランダム」に対して，39.6 ポイントの効果があるといえる．

計算機ごとに，詳細にみると，まず，[PCn1] と [Ppower] では，低い利用率で高い正答率が得られた．[Ppower] では判定基準が 2 回連続では 82.1%の正答率を 34.4%の利用率で [PCn1] では判定基準が 2 回連続でも 75.0%の正答率を 32.1%の利用率で得た．

この理由は，[PCn1]，[Ppower] では，推定すべき関数形状が比較的単調なものが多いことによる．さらに，[PCn1]，[Ppower] では，CRS(BCRS = 1) が最適値になることが多く，初期値に 1 が含まれている効果も大きい．初期値に 1 を含むことは，“初期値に両端の値を含む” とする標本点逐次追加型性能パラメータ推定法の手順（図 4 参照）によるものである．また，こ

のことは，ブロック化のサイズを決める前に，ブロック化をするか否かの選択について，手順を区別せずに，かつ高効率に行うことができることを示しており，本推定法によるアルゴリズムの特徴となっている．

一方，[SR8k] では，比較的多くの標本点を用いることにより，判定基準を 4，5 回連続とすることで，それぞれ 78.6%，89.3%の正答率を 62.9%，74.1%の利用率で実現することができた．

[SR8k] では，推定すべき関数形状として “山あり谷あり” の特性が多く見られた．“山あり谷あり” の関数形状に対して，本推定法が効率良く動作することは文献 6) でその振舞いを詳細に示しているので参照されたい．

3.3.3 SR11000 での実験結果と分析

今回のケースでは，スーパコンピュータ SR11000[SR11k] が “ランダム” のライン上という結果となった．つまり，正答率の観点からは，標本点をランダムに選択した場合と差がないことになる．そこで，正しく選択されていないケースについて，どのような特性になっているのかを分析した．

3.3.1 項の表 2 で説明したように，[SR11k] では，BCRS(2) が最適値となるケースが多い．そこで，[SR11k] の BCRS(2) が最適値となる典型的な項番 27 Zd_Jac3 を例に，逐次追加型推定法による標本点選択・追加の様子を**図 11** に示す．図 11 において，横軸は，性能パラメータであるブロックサイズをとる．ここで，ブロックサイズ 1 とは，CRS（非ブロック化）を意味する．縦軸は，数値計算ライブラリの実行時間である．

図 4 の手順に従って説明する．まず，初期値として，CRS，BCRS(6)，BCRS(11)，BCRS(16) の 4 点を標本点として選択し，それらを性能パラメータの値として，数値計算ライブラリを実行する．それらの実測値をもとに，d-Spline を計算し，その結果，最適値として CRS を推定する．次に，5 番目の標本点を選択する．2.3 節の選択条件 1 により選択される CRS は，すでに標本点として選択されているので，選択条件 2 を用いて BCRS(10) を選択する．この 5 つの標本点の実測値を用いて，d-Spline を再計算する．その結果，最適値として同じく CRS を推定する．さらに同様に，6 番目，7 番目，... を選択条件 2 を用いて，ブロックサイズの値の大きな性能パラメータを順次選択する．この際，推定される最適値は，つねに CRS のまま変化しない．したがって，連続して CRS が推定されるため，終了判定条件に従い，2 回から 5 回連続では CRS を最適値として決定する．

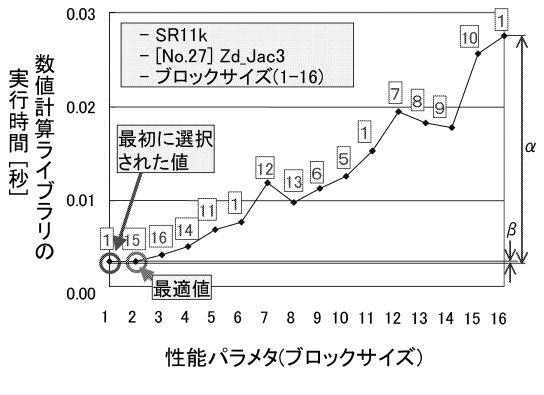


図 11 標本点逐次追加型性能パラメータ推定法による標本点選択・追加の振舞い

Fig. 11 Behavior of incremental parameter estimation method.

図 11 から分かるように，選択された CRS と BCRS(2) はほとんど性能としての差はない．このような場合，提案した標本点逐次追加型性能パラメータ推定法では，最初に推定した最適値（ここでは “1”）の周辺を探索するような局所最適化を行うのではなく，性能パラメータのとりうる値全域を対象とする大域的な探索を行うように作用する．これは，提案方法の標本点選択基準のためである．

3.3.4 標本点逐次追加型性能パラメータ推定方式の評価尺度

標本点のとりうる値の分布に対して，標本点逐次追加型性能パラメータ推定法による推定値が，もし，正しい最適値でなくても，どの程度，最適値に近いかの評価を行うために，次のような評価尺度を定義する．

$$EvalMeasure = \frac{\frac{T_{select}}{T_{min}} - 1}{\frac{T_{max}}{T_{min}} - 1} = \frac{T_{select} - T_{min}}{T_{max} - T_{min}} \tag{5}$$

ここで，T_{select} は選択した値の実行時間，T_{min} は最適値の実行時間および T_{max} は最悪値の実行時間を示す．これは図 11 における β/α を意味する．評価尺度 *EvalMeasure* は，最適値を選択したとき，すなわち $T_{select} = T_{min}$ のとき “0” を，最悪値を選択したとき，すなわち $T_{select} = T_{max}$ のとき “1” となる．したがって，$[0, 1]$ 区間の値をとる．

この評価尺度 *EvalMeasure* を用いて，評価した結果を**図 12** に示す．図 12 において，横軸は，終了判定基準による連続回数を示す．縦軸は，評価尺度 *EvalMeasure* を示す．各ラインは，計算機環境ごとの値を示している．今回，正答率の観点から問題となった [SR11k]

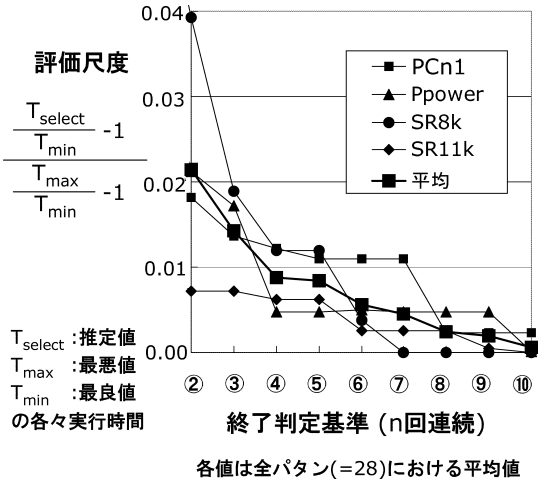


図 12 評価尺度に対する計算機環境ごとの振舞い
Fig. 12 Behavior of *EvalMeasure*.

表 5 d-Spline の生成時間
Table 5 Execution time of d-Spline.

処理内容	実行時間
全実行時間の平均	34 msec.
28 種各々のブロックサイズの最小実行時間（最適値での実行時間）の平均	9.2 msec.
d-Spline の生成時間と xp_{newsp} の探索時間	28 μsec.

の評価尺度 EvalMeasure は，平均よりも小さい値となっている．したがって，最適値ではないけれども，他の計算機環境に比べて，平均的により最適値に近い値を選択していることが分かる．

3.3.5 性能パラメータ推定に要する時間

2.2 節において，d-Spline の計算量について述べたが，今回の評価実験において，d-Spline の生成時間を PC サーバ【PCn1】の計算機環境で測定した（**表 5**）．

d-Spline の生成時間は 28 μsec となった．疎行列ベクトル積の計算全体の平均値 34 msec. の約 1/1000，疎行列ベクトル積の各モデルの計算時間の最小値の平均値 9.2 msec. よりもさらに 2 桁小さい．したがって，実行時最適化において，d-Spline を用いた性能パラメータ推定の計算量は無視することができることを確認した．

4. おわりに

最低限の数の標本点からはじめて，必要な標本点を選択し追加しながらコスト定義関数を順次更新し，最適な性能パラメータの値を推定する標本点逐次追加型性能パラメータ推定法を実行時ソフトウェア自動チューニングに適用した．数値計算ライブラリとして，実行時に行列上の非零要素の位置が決定する，疎行列の行列ベクトル積を扱った．評価性能パラメータとして，疎行列のブロック化を行うときのブロック化サイズを取り上げ，スーパコンピュータならびに PC サーバを対象とした．実機を用いた実証実験を行った．

(1) 終了判定基準を連続 4 回とした場合，ブロック化の効果として，計算機ごとに，1.07 倍から 2.23 倍となった．すべての性能パラメータのとりうる値を実測した場合の結果は，1.10 倍から 2.26 倍となることから，平均で最適値の 97.3%の効果を得る値を選択することができた．

(2) 正答率で比較すると，平均では，終了判定基準 4 回連続では，正答率 75.9%に対して標本点の利用率 55.0%となった．同じ正答率を得るために「ランダム」では 75.9%の標本点を必要とすると考えられるので，利用率に関し 20.9 ポイントの効果があるといえる．最大でみると，終了判定基準 4 回連続では，[Ppower] のケースで，正答率 92.9%を 53.3%の少ない利用率で実現している．平均と同様に，「ランダム」に対して，39.6 ポイントの効果がある．

(3) 標本点逐次追加型性能パラメータ推定法の探索の振舞いを分析し，局所最適化でなく，大域的探索を実施していることを示した．

(4) 標本点のとりうる値の分布に対して，標本点逐次追加型性能パラメータ推定法による推定値が最適値にどの程度近いかどうかの評価尺度を定義し，どの程度正しく推定できたかを定量的に示した．

(5) 標本点逐次追加型性能パラメータ推定法を実行時自動チューニングへ適用するための手順を示した．そこで，新たに必要となる d-Spline の生成時間を実測し，疎行列の行列ベクトル計算処理時間と比較し，平均的に 1/1000 となること，最小値と比べても 2 桁の違いがあることから，実行時ソフトウェア自動チューニングにおいて，d-Spline を用いた性能パラメータ推定の計算量は無視することができることを確認した．

今後の課題として，今回提示した方式を実際のソフトウェア自動チューニングツールに組み込み，実用化を図っていく．

謝辞 本研究の一部は，文部科学省科学研究費補助金基盤研究（C）（課題番号：18500018）「数値計算と組み込みシステムのための自動チューニング方式」，および，京都大学学術情報メディアセンター，スーパーコンピュータ共同研究制度（若手研究者奨励枠），「スーパーコンピュータ環境におけるソフトウエア自

動チューニング技術に関する研究」の支援により行われた.

参 考 文 献

1) Bilmes, J., Asanovic, K., Chin, C. and Demmel, J.: Optimizing Matrix Multiply using PHiPAC: A Portable High Performance, ANSI C Coding Methodology, *Proc. ICS 97*, pp.340–347 (1997).
2) Whaley, R., Petitet, A. and Dongarra, J.: Automated Empirical Optimizations of Software and the ATLAS project, *Parallel Computing*, Vol.27, pp.3–35 (2001).
3) 片桐孝洋：ソフトウェア自動チューニング：数値計算ソフトウエアへの適用とその可能性，慧文社 (2004).
4) Barrett, R., Berry, M., Chan, T., Demmel, J., Donato, J., Dongarra, J., Eijkhout, V., Pozo, R., Romine, C. and Vorst, H.: *Templates for the Solution of Linear Systems: Bullding Blocks for Iterative Methods*. 長谷川里美，長谷川秀彦，藤野清次（訳）：反復法 Tamplates，朝倉書店 (1996).
5) 田中輝雄，片桐孝洋，弓場敏嗣：ソフトウェア自動チューニングにおける d-Spline を用いた性能パラメタ推定法，先端的計算基盤システムシンポジウム，SACSIS2006, pp.159–166 (2006.5).
6) Tanaka, T., Katagiri, T. and Yuba, T.: d-Spline Based Incremental Parameter Estimation in Automatic Performance Tuning, *Workshop on State-of-the-Art in Scientific and Parallel Computing* (*PARA'06*), CP4-2 (2006.6).
7) 田中輝雄，片桐孝洋，弓場敏嗣：ソフトウェア自動チューニングにおける標本点逐次追加型性能パラメタ推定法，電子情報通信学会論文誌，Vol.J90-A, pp.281–291 (2007).
8) Im, E.-J.: Optimizing the performance of sparse matrix-vector multiplication, Ph.D. dissertation, UCB (2000, 5).
9) Davis, T.: UF Sparse Matrix Collection. www.cisu.ufi.edu/research/sparse/matricies
10) Vuduc, R.: Automatic performance tuning of sparse matrix kernels, Ph.D. dissertation, UCB (2003, 12).
11) 片桐孝洋，黒田久泰，大澤清，金田康正：ILIB：自動チューニング機構付き並列数値計算ライブラリとその性能評価，並列処理シンポジウム，JSPP2000, pp.27–34 (2000).

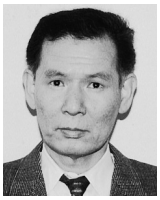
(平成 19 年 1 月 22 日受付)
(平成 19 年 5 月 23 日採録)

田中 輝雄（正会員）

1981 年電気通信大学情報数理工学科卒業．1983 年同大学院情報数理工学研究科修士課程修了．1981 年 4 月から 1983 年 3 月まで文部省統計数理研究所（現在，独立行政法人統計数理研究所）特別研究生．2007 年電気通信大学大学院情報システム学研究科博士課程修了．博士（工学）．1983 年（株）日立製作所中央研究所入所．2007 年 4 月から（株）日立超 LSI システムズ所属．1997 年 10 月から 2005 年 3 月まで電気通信大学情報工学科非常勤講師．専門は，大規模数値計算アルゴリズム，並列処理アーキテクチャ，ソフトウェア自動チューニング．日本応用数理学会会員．

片桐 孝洋（正会員）

東京大学情報基盤センター特任准教授．1994 年豊田工業高等専門学校情報工学科卒業．1996 年京都大学工学部情報工学科卒業．2001 年東京大学大学院理学系研究科情報科学専攻博士課程修了．博士（理学）．2001 年 4 月日本学術振興会特別研究員 PD，2001 年 12 月科学技術振興機構研究者，2002 年 6 月電気通信大学大学院情報システム学研究科助手，2005 年 3 月から 2006 年 1 月米国カリフォルニア大学バークレー校コンピュータサイエンス学科訪問学者を経て，2007 年 4 月より現職．並列計算機を用いた効率の良い行列計算アルゴリズムの研究，およびソフトウェア自動チューニングの研究に従事．2002 情報処理学会山下記念研究賞受賞．著書「ソフトウエア自動チューニング：数値計算ソフトウエアへの適用とその可能性」，慧文社（2004）．日本ソフトウェア科学会，日本応用数理学会，ACM，IEEE-CS，SIAM 等各会員．



弓場　敏嗣（フェロー）

1966年神戸大学大学院工学研究科修士課程修了．（株）野村総合研究所を経て，1967年通商産業省工業技術院電子技術総合研究所（現在，独立行政法人産業技術総合研究所）に入所．以来，計算機のオペレーティングシステム，見出し探索アルゴリズム，データベースマシン，データ駆動型並列計算機等の研究開発に従事．その間，計算機方式研究室長，知能システム部長，情報アーキテクチャ部長等を歴任．1993〜2007年電気通信大学大学院情報システム学研究科教授．2007年同名誉教授．高性能計算の科学技術一般に興味を持つ．工学博士．電子情報通信学会フェロー．ACM，IEEE Computer Society 各会員．