# 挿入拡張・中抜き縮小可能な多次元配列

## Multidimensional Arrays Enabling Insert Extensions and Delete Reductions

熊切 正和* 李 蓓* 都司 達夫**
樋口 健**

Masakazu KUMAKIRI LI Bei Tatsuo TSUJI
Ken HIGUCHI

**最近，MOLAP 等，多次元データの格納データ構造として多次元配列の有用性が再認識されている．通常，データベースで用いられる多次元配列は，要素に高速にランダムアクセスできるために，全ての次元のサイズが固定である．そのため，配列データの再配置無しに拡張や縮小など配列のサイズ変更を行えない．拡張可能配列では，データの再配置を必要とせずに，どの次元方向に対しても拡張が行える．しかし，拡張可能配列は，配列の外縁に対してのみしか拡張ができないという制限がある．本稿では，中抜き縮小や挿入拡張を導入することで，柔軟にサイズの動的変更が可能な拡張可能多次元配列について述べる．**

**Multidimensional arrays are becoming important for storing multidimensional data. To be benefited by the fast random accessing capability, size of multidimensional arrays should be fixed in every dimension. While such a fixed size array cannot extend or reduce without relocating all of the elements, an extendible array can extend its size along any directions without any relocation. However the existing extendible arrays can always extend only at the surrounding. We propose a new flexible extendible array organization, in which a subarray can be inserted or removed even in the midst of the array.**

.

## 1. はじめに

大規模な科学技術データを効率よく取り扱うため，多次元配列の構造化技法やそれらの実装技術の研究が行われてきた(例えば[1])．また近年，多次元配列を用いたMOLAP（Multidimensional On-Line Analytical Processing）の研究(例えば[2][5])が盛んに行われるようになり，大量データの統計的な分析を基に経営や販売戦略の意思決定に使われている．MOLAPシステムでは，多次元配列に対する問合わせを迅速に処理するために，配列の高速参照が要求される．

要素参照の速度に加えて，多次元配列に対する効率よい演算･操作機能はこれらのアプリケーションにとって重要な要求であり，参照速度を劣化させないために，対象とする配列高速なアドレス関数を使用する．しかし，新たなカラム値を持つレコードが挿入されたときには，その次元方向により大きい新たな固定サイズ配列を確保して，新たなアドレス関数を使って，元の配列要素すべてを再配置する必要があるために，著しくリアルタイム性が阻害される．

拡張可能配列は実行時に動的に任意の次元方向にそのサイズが拡張できる配列である[3]～[6]．拡張可能配列では拡張部分のみが動的に割り付けられ，拡張前の配列要素のデータは再配置することなくそのまま利用される．上記アプリケーションをはじめ，配列サイズが実行環境の変化に応じて動的に変化しうるような様々なアプリケーション分野において有利となる．拡張可能配列の実装上のポイントは，配列要素のアクセス性能や記憶域の利用率を犠牲にせずに，拡張可能性を実現することである．

[3]ではハッシュ関数を使用した拡張可能配列の実現技法が提案されており，[4][5][6]では補助テーブルを使用した実現技法が論じられ，[4]ではハッシュ関数による方法に対する優位性が述べられている．これらは，[6]以外，すべて配列の外縁部からの拡張のみ可能である．[6]では領域の縮小をとしているが，やはり，外縁部からの縮小のみであり，応用上，問題となっている．本論文では拡張可能配列の内部に対してサブ配列を挿入して配列を拡張（挿入拡張）したり，内部のサブ配列を除去して配列を縮小（中抜き縮小）することを可能にするための方式を提案し，実装・評価する．

## 2. 拡張可能配列の実現モデル

n 次元拡張可能配列は次元毎に 3 つの補助テーブルと経歴値カウンタを有している．これらは配列要素のアドレス計算に使用される．補助テーブルの 1 つは経歴値テーブルと呼ばれ，配列拡張が行われるたびに経歴値カウンタの値は 1 インクリメントされ，それが記録される．他の 1 つはアドレステーブルと呼ばれ拡張時に動的に連続領域に割り当てられるサブ配列の先頭アドレスを保持する．このサブ配列は n-1 次元であり，拡張時の配列形状によってその各次元サイズが定まる通常の固定サイズ配列である．図 1 は 2 次元拡張可能配列の例である．3 次元以上の拡張可能配列の場合，n-1 次元のサブ配列内要素のアドレス計算のための 1 次関数について，その n-2 個の係数をサブ配列毎に記録する係数テーブルを必要とする．たとえば，n=4 でサイズ[p,q,r]の 3 次元サブ配列の要素<i,j,k>に対するアドレス関数の例は qri+rj+k であり，係数テーブルには(qr,r)が格納される．

図 1 において，1 次元方向および 2 次元方向の経歴値テーブルをそれぞれ$H_1$,$H_2$，またアドレステーブルをそれぞれ，$L_1$,$L_2$とする．例えば，配列要素<3,4>のアドレス計算は次のように行われる．$H_1(3) < H_2(4)$であるから，要素<3,4>を含むサブ配列Sは経歴値$H_2(4)=7$ の時に割り付けられたサブ配列であり，その先頭アドレスは$L_2(4)=60$ である．要素<3,4>のSにおける先頭アドレスからのオフセットは3であるので，求めるアドレスは 63 となる．

一般に n 次元の拡張可能配列の場合，サブ配列は n-1 次元固定配列であり，その要素のアドレス関数は，n 次元固定配列のアドレス関数より，項数が少ないので掛け算回数が減少する（ただし，n 個の経歴値を比較して，最大値を決定するコストを要する）．シンプルな計算により，要素アドレスが決定すること，補助テーブルのサイズは配列本体に比べてわずかであり，拡張可能性はこのわずかな記憶領域のみを余分に必要とすることで達成されていることに注目されたい．

*学生会員 福井大学大学院工学研究科博士前期課程
{m_kuma,libei}@pear.fuis.fukui-u.ac.jp
**正会員 福井大学大学院工学研究科
{tsuji,higuchi}@pear.fuis.fukui-u.ac.jp

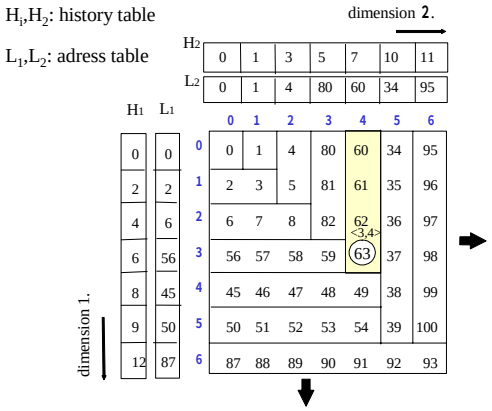



図 1 拡張可能配列の実現モデル
Fig.1 Realization Model of Extendible Arrays

## 3. 挿入拡張，中抜き縮小を許す拡張可能配列

挿入拡張や中抜き縮小は既に配列に格納されている要素の論理的な位置，即ち添字の組へ影響を与える．図 2 の拡張例では，次元 1 へ新たなサブ配列を挿入拡張することで，要素 A の論理的な位置は<1,1>から<2,1>となる．また，図 3 の縮小例において，縮小の際には，要素 B は<2,2>から<1,2 >に変わる．実際に要素が含まれているサブ配列や物理アドレスに変化は無いが，上記のように論理的位置がずれることで，サブ配列中の正しいオフセットの計算ができず，誤ったアドレスを導いてしまう．

例えば，図 2(a)の配列において，要素 A の位置<1,1>は次元 2 の添字 1 のサブ配列中に存在し，次元 1 の添字が 1 であるためオフセットは 1 となる．要素 A の論理位置は次元 1 の 1 列目へ挿入拡張を行った後の図 2(b)では<2,1>となり，2 節の計算手順によれば，拡張前と変わらずに次元 2，添字 1 のサブ配列に含まれることがわかる．しかし，要素 A の次元 1 における添字は挿入拡張前の値と異なりオフセットは 2 となるので，挿入拡張前後で同じアドレスが導けず，正しい結果は得られない．縮小においても同様の問題が生じる．なお，以後，縮小によって開放された領域を斜線で図示したレイアウトを補正レイアウト，縮小した領域を詰めたユーザの視点からのレイアウトを論理レイアウトと呼ぶ．

### 3.1 補正のための構造

挿入拡張・縮小で生じた上記のオフセットのずれを補正し，正しい配列要素の位置を求めることができる構造を提案する．要素eのユーザ指定の論理位置を$<i_1,i_2,\cdots,i_n>$とすると，eを含むサブ配列は 2 節で述べた手順に従って求まり，その次元をpとする．このサブ配列を以後，eの主サブ配列という．補正が必要な次元はp以外の次元であり，次元p以外の添字に対応するサブ配列をeの副サブ配列と呼ぶ．補正値の算出はbitmapを用いて行う．bitmapは次元毎に作成され，挿入拡張によるオフセットのずれを補正する拡張補正値と縮小によるオフセットのずれを補正する縮小補正値をそれぞれ個別に求める．したがって，それぞれの次元毎に拡張補正bitmap(図 4(b))，縮小補正bitmap(図 5(b))を保持する．次元k(k≠p)の補正値$\Delta_k$は，

$\Delta_k$＝縮小補正値－拡張補正値

となり，補正後のeの正しい論理位置$<i_1',i_2',\cdots,i_n'>$は$i_k' = i_k + \Delta_k$ となる．それぞれのbitmapはbit列と補正値計算に用いるbit列選択のための経歴値の組の集合で構成される．この経歴値は組になるbit列が生成されたタイミングを表す．また，本方式では図 4(a)のように，新たに論理レイアウトの添字に対応する補正レイアウトにおける補正添字テーブル(revised subscript table)を追加する．bit 列中の bit 位置はその次元の補正添字テーブルの添字を表し，1 の場合には補正が必要なことを表す．次元 k に対する，要素 e の 2 種類の補正値は次のように求まる．

(1) e の主サブ配列の経歴値を h とし，h を超える最小の経歴値を bitmap 中で引き当てる．
(2) $r_k$を副サブ配列に対応する補正添字テーブルの添字であるとし，(1)で引き当てたbit列中bit $r_k$までの 1 の総数をカウントし，これを次元kの補正値とする．

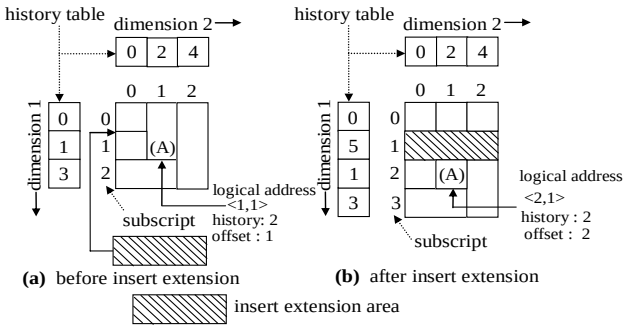



図 2 サブ配列の挿入拡張
Fig.2 Insert extension of a subarray

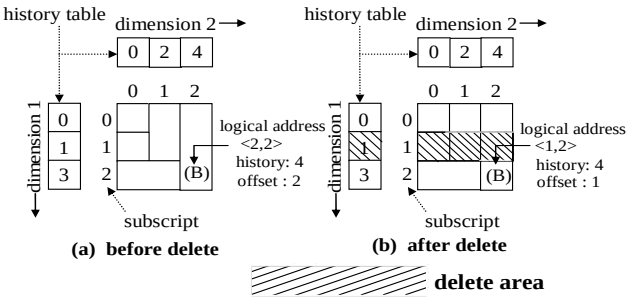



図 3 中抜き縮小
Fig.2 Delete reduction

### 3.2 拡張補正例

図 4(b)は図 4(a)の 2 次元目の bitmap を示している．ここで，要素 A<3,5>の 2 次元拡張補正値を求める手順を示す．要素 A の主サブ配列は次元 1 の経歴値 6 のサブ配列である．したがって，次元 2 の補正値を求める必要があり，次元 2 の指定添字 5 が補正対象となる．

(1) 図 4(b)の bitmap において補正に使われる bit 列を求める．これには，経歴値 6 を超える最小の経歴値を bitmap 中で引き当てる．経歴値 8 が該当し，それに対応する bit 列 0010100 が補正に使われる．
(2) 要素 A の次元 2 の副サブ配列の補正添字は 5 であり，(1)の bit 列中 bit 5(最左 bit が bit 0)までの 1 の総数は 2 である．従って拡張補正値は 2 となる．

要素 B<4,5>と C<3,1>は共に要素 A の近隣にあるが，次元 2 の補正値は A のものとは異なる．B に対しては，bit 列 0000100 が選ばれ，bit 5 までの 1 の bit 総数は 1 であるので，補正値は 1 である．同様にして，C の補正値は 0 となる．

### 3.3 縮小補正例

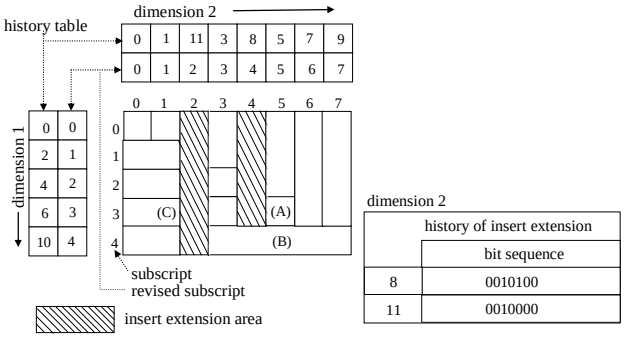
図 5(a)は中抜き縮小された拡張可能配列の例であり，図 5(c)は図 5(a)の論理レイアウトを示す．また，図 5(b)は図 5(a)

の 2 次元目の bitmap を示している．要素 A<4,2>の 2 次元縮小補正値を求める手順を示す. 要素 A の主サブ配列は次元 1 の経歴値 11 のサブ配列である．従って，次元 2 の補正値を求める必要があり，次元 2 の添字 2 が補正対象となる．

(1) 図 5(b)の bitmap において補正に使われる bit 列を求める. これには, 経歴値 11 を超える最小の経歴値を bitmap 中で引き当てる．経歴値 12 に対応する bit 列 00010110 が補正に使われる．

(2) 要素 A の次元 2 の副サブ配列の補正添字は 4 であり, (1) の bit 列中 bit 4 までの 1 の総数は 1 である．従って縮小補正値は 1 となる．

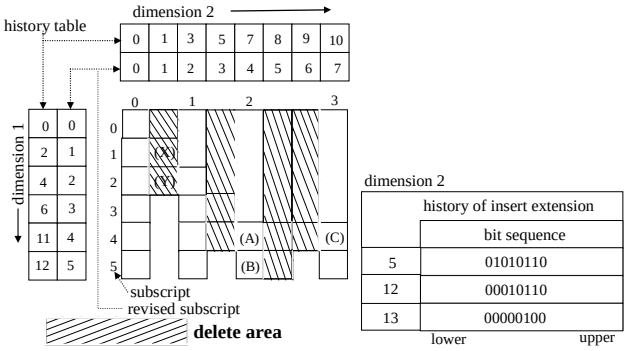
要素 B<5,2>と C<4,3>は共に要素 A に隣接しているが, 次元 2 の補正値は A のものとは異なる．B に対しては，bit 列 00010000 が選ばれ，bit 4 までの 1 の bit 総数は 0 であるので, 補正値は 0 である. 同様にして, C の補正値は 3 である．



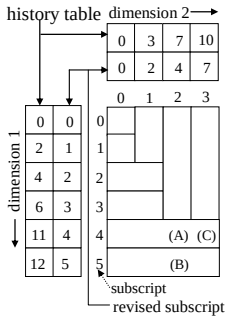

(a) 補正レイアウト (b) 次元 2 の bitmap

図 4 拡張補正のためのデータ構造

Fig.4 Compensatory data structures for insert extension




(a) 補正レイアウト (b) 次元 2 の bitmap




(c) 論理レイアウト

図 5 縮小補正のためのデータ構造

Fig.5 Compensatory data structures for delete reduction

### 3.4 挿入拡張，中抜き縮小時の動作

**(1) 補助テーブルに対する操作**

**［挿入拡張時］** 次元 i の添字 j に新たな値のサブ配列を挿入拡張する場合，まず次元 i の補助テーブルの j 番目以降のテーブル情報をすべて後ろへ 1 シフトする．このとき j 番目以降のテーブル情報の補正添字をシフト時に 1 インクリメントしておく．空いた添字 j のテーブルには挿入拡張により追加されるサブ配列の情報を格納する．挿入サブ配列の補正添字は添字 j-1 のサブ配列の補正添字に 1 加えた値とする．

**［中抜き縮小時］** 次元 i の添字 j の値を中抜き縮小する場合，次元 i，添字 j のサブ配列本体を解放する．ただし，このとき実際に解放できる領域は次元 i 添字 j 番目の主サブ配列本体のみであり，その他の縮小により使用されなくなる領域，例えば図 5(a)の例では X や Y の要素等は開放されない．これらの要素はそれが属する主サブ配列が解放されたときに初めて解放される．

次に次元 i の補助テーブルの，添字が j 以降のサブ配列情報をそれぞれ一つ前へシフトして，テーブルを詰める．この時，補正添字テーブルの j 以降の要素は更新されない．

**(2) 補正 bitmap に対する操作**

**［挿入拡張時］** 次元 i，添字 j に新たな値のサブ配列を挿入拡張する場合, 次元 i の拡張補正 bitmap, 縮小補正 bitmap 双方の全ての bit 列に対し，bit j 以降を 1 右シフトし，bit 列の bit j は, 拡張補正 bitmap 中 bit 列は 1, 縮小補正 bitmap 中 bit 列は 0 に変更する．

**［中抜き縮小時］** 次元 i，添字 j の値を中抜き縮小する場合は，次元 i の縮小補正 bitmap 中全ての bit 列に対し，bit j を 1 とする．

**(3) 補正 bitmap へ bit 列を追加するタイミング**

補正 bitmap 中の bit 列は挿入拡張・中抜き縮小のタイミングにより新たに bit 列を追加する必要がある．追加される bit 列の初期値は全 bit 0 である．

**［拡張補正bitmapの追加］** 次元iの拡張補正bitmapは次元iの拡張$E_i$と，$E_i$の直前に次元iにて行われた拡張$E_i$'との間に，i以外の次元jで拡張$E_j$が行われていた場合に新たにbit列を追加する．bit列は拡張$E_i$が行われるタイミングで追加し，追加したbit列の挿入拡張時経歴値は拡張$E_i$が行われたときの拡張可能配列の経歴値カウンタの値となる．

**［縮小補正bitmapの追加］** 次元iの縮小補正bitmapは次元iの縮小$D_i$と，$D_i$の直前に次元iにて行われた縮小$D_i$'との間に，i以外の次元jで拡張$E_j$が行われていた場合に新たにbit列を追加する．bit列は縮小$D_i$が行われるタイミングで追加し，追加したbit列の中抜縮小時経歴値は縮小$D_i$が行われたときの拡張可能配列の経歴値カウンタの値となる．

## 4. 評価

従来の拡張可能配列システム（CSと表記）と本方式の挿入拡張・中抜き縮小を可能とする拡張可能配列システム(FSと表記)を比較し，評価する．両方式とも拡張可能配列はすべて主メモリ上に構築している．FSにおいて縮小，挿入拡張の回数に依るアクセス速度への影響を調べるために，

(1) 挿入拡張と中抜き縮小が共に行われていない状態: FS(a),
(2) 中抜き縮小のみ何度か行われている状態: FS(b),
(3) 挿入拡張のみ何度か行われている状態: FS(c),
(4) 挿入拡張と中抜き縮小が共に行われた状態: FS(d)

の4 種類の状態で測定を行う．また，拡張可能配列の次元数

は3,4,5,6とし，拡張や縮小が行われた後の各次元サイズは等しく，次元数順にそれぞれ400,90,35,20としている．

### 4.1 アクセスコストの比較

**(1)ランダムアクセスコスト**

図 6 は多次元拡張可能配列に対し，その全要素数の 1/10 回ランダムに要素参照した際に要素アクセスに要した時間である．図 6 より FS によるアクセス時間は，CS の時間の約 2.8 倍であることが確認できる．この差は，補正値演算時の bitmap 操作と新たなテーブルの表引き操作に依るものである．また，両方式共に次元数が大きくなるにつれ，アクセス時間が増加している．配列の次元数が大きくなれば，サブ配列の次元数も大きくなり，アドレス計算のための乗算が増え，また FS においては補正個所が増えるためである．なお，FS における挿入拡張，中抜き縮小の有無に依る時間変化はほとんど見られず，それらのよるアクセス速度低下もほとんど見られない．

**(2)範囲アクセスコスト**

MOLAPをはじめ多くのアプリケーションでは単一要素へのアクセスコストよりもスライス検索等の範囲アクセスコストが重要である．範囲アクセスではCS,FS共に範囲内の要素を含むサブ配列毎に要素へのアクセスを行う．このため係数ベクトルを用いたサブ配列内オフセットの計算は，サブ配列単位で再帰的に行うことができ，乗算回数を大幅に減らすことができる．これにより両方式共に範囲アクセス時間は大幅に短縮できる．また，同一サブ配列内では補正値を共用できるためFSにおける補正値計算回数は大幅に減らすことができ，CSとの速度差は減少する．

アクセス範囲を多次元拡張可能配列の全範囲であるとして，すべてのサブ配列を順次アクセスしながら，サブ配列の中では上述のように要素を再帰的にアクセスした．このような範囲アクセスの速度比較として,CSとFSについて，配列の全要素を参照したときのアクセスに要する時間を測定した結果を図7 に示す．FSにおけるアクセス時間は，CSより5%程度しか悪化していないことが確認できる．また，CS,FSに配列の次元数の増加による速度低下もほとんど見られない．なお，ランダムアクセスの場合と同様，FSにおける挿入拡張，中抜き縮小の有無による時間変化はほとんど見られず，それらの回数に依るアクセス速度低下もほとんど見られない．

### 4.2 記憶コストの比較

FS でも CS と同様に，サブ配列本体の領域，および各種補助テーブルを次元毎に保持するが，これらは配列本体のサイズに比べていずれも無視できる程度である．FS ではこれらに加えて要素の論理的な位置を記す補正添字テーブルおよび補正 bitmap を次元毎に保持する．挿入拡張や中抜き縮小の回数が増加するにつれて補正添字テーブルと補正 bitmap のサイズは増大するが，本体サイズに比べて，やはり無視できる．紙面の都合で詳しい評価は省略する．中抜き縮小により，使用されないサブ配列の要素(例えば図 5(a)の X や Y)が増加することは問題であり，ある時点で，配列全体を再構成する必要がある．

## 5. まとめと今後の課題

従来の拡張可能配列に，挿入拡張，中抜き縮小操作を導入することで，より柔軟なサイズ変更が可能な多次元配列の実現方式を提案して，評価した．これにより，MOLAP 等で重要なスライス検索などの範囲アクセスを，従来方式とほぼ同等の速度で行うことできる．今後，MOLAPのための，従来の多次元配列より柔軟性の高い基本データ構造としての応用を検討する予定である．

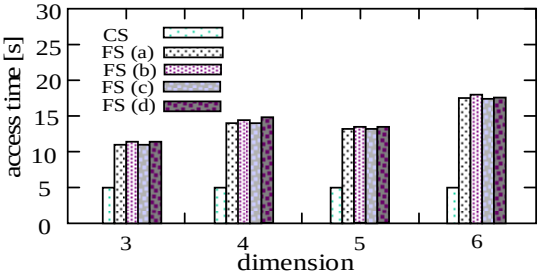



**図6 ランダムアクセス時間**
**Fig.6 Random access time**

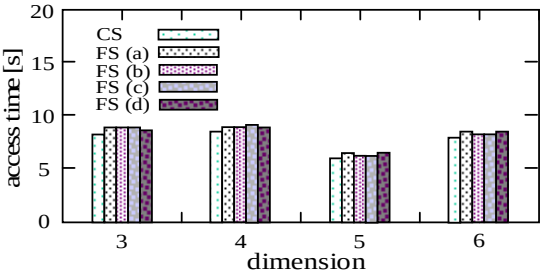



**図 7 範囲アクセス時間**
**Fig.7 Range access time**

## ［文献］

[1] Sarawagi S., Stonebraker M.: “Efficient Organization of Large Multidimendional Arrays”, Proc. of ICDE, pp. 328-336 (1994).

[2] Zhao Y., Deshpande P.M., Naughton J.F.: "An Array-Based Algorithm for Simultaneous Multidimensional Aggregates", Proc. of SIGMOD, pp.159-169 (1997).

[3] Rosenberg A.L., Stockmeyer L.J.: "Hashing Schemes for Extendible Arrays", JACM, Vol.24, pp.199-221 (1977).

[4] Otoo E.J., Merrett T. H. : "A Storage Scheme for Extendible Arrays", Computing, Vol.31, pp.1-9（1983.

[5] Rotem D., Zhao J.L., "Extendible Arrays for Statistical Databases and OLAP Applications", Proc. of SSDBM, pp.108-117(1996).

[6] 都司達夫，水野剛，宝珍輝尚，樋口健：拡張可能配列の遅延割付け方式，電子情報通信学会論文誌 D-I, Vol.J86-D-I, No.5, pp.351-356 (2003).

**熊切 正和 Masakazu KUMAKIRI** 2006 年福井大学大学院工学研究科博士前期課程修了．現在，(株)NEC アクセステクニカ．在学中，拡張可能配列に関する研究に従事．IPSJ，日本データベース学会各学生会員．

**李 蓓 LI Bei** 福井大学大学院工学研究科博士前期課程在学中．拡張可能配列，MOLAP 等の研究に従事．IPSJ，日本データベース学会各学生会員．

**都司 達夫 Tatsuo TSUJI** 福井大学大学院工学研究科教授．1978 年大阪大学大学院基礎工学研究科博士課程修了．IEICE,IPSJ,日本データベース学会,IEEE 各会員．

**樋口 健 Ken HIGUCHI** 福井大学大学院工学研究科助教授．1997 年電気通信大学大学院工学研究科博士課程修了．IEICE,IPSJ,日本データベース学会,ACM 各会員．