MOLAP用多次元配列の圧縮方式とその並列化

一色　淳夫†　　都司　達夫††　　宝珍　輝尚††　　樋口　　健††

† 福井大学大学院工学研究科　〒 910-8507 福井県福井市文京 3 丁目 9 番 1 号
†† 福井大学工学部　〒 910-8507 福井県福井市文京 3 丁目 9 番 1 号
E-mail: †{ishiki,tsuji,hochin,higuchi}@pear.fuis.fukui-u.ac.jp

あらまし　MOLAP において使用される多次元配列の実装は，一般に (1) 疎配列となる，(2) 集約演算や検索の速度が配列の次元に依存する，という問題点がある．特に (2) は配列スライスに対する集約演算や順次検索において，スライスする次元により応答性にばらつきが生じる．ここでは，配列全体を chunk と呼ばれる論理ブロックの集合として管理し，chunk を圧縮してディスクページへの詰め合わせを行う．ところが詰め合わせの方法によっては，さらなる次元依存性が発生し得る．本論文ではこの新たな次元依存性による (2) の欠点を軽減するための詰め合わせアルゴリズムの提案とその並列化について述べて，評価する．

キーワード　多次元 DB, OLAP, データウェアハウス

Compression Schemes of Multidimensional Arrays for MOLAP and Their Parallelization

Atsuo ISSHIKI†, Tatsuo TSUJI††, Teruhisa HOCHIN††, and Ken HIGUCHI††

† Graduate School of Engineering, Fukui University　Bunkyou 3-9-1, Fukui, 910-8507 Japan
†† Department of Information Science, Fukui University　Bunkyou 3-9-1, Fukui, 910-8507 Japan
E-mail: †{ishiki,tsuji,hochin,higuchi}@pear.fuis.fukui-u.ac.jp

Abstract　The implementation of multidimensional arrays used in MOLAP suffers from the problems; (1) the number of nonempty elements tends to be so small, (2) the speed of the aggregation operations or elementns retrieval heavily depends on the dimension along which the elements are accessed. The latter is especially important in the aggregation on *sliced* elements frequently required in MOLAP. This problem would be alleviated by dividing the whole array into the set of the same sized smaller subarrays called *chunks*. Here, these chunks are packed in a container (i.e., a disk page), but further dimension dependency may arise depending on the selected packing method. We propose several algorithms for compressing chunks and packing them in a container that would not cause this further dimension dependency in accessing array elements. We also describe some schemes of parallelizing these algorithms.

Key words　Multidimensional DB, OLAP, Data warehousing

1. はじめに

現在, 基幹系システムからデータのスナップショットがとられ大規模なデータベースを作成して, 意志決定支援に利用されている．ユーザは, このデータベース中の任意の属性の組合せについて様々な集約結果を求め多次元分析を行う. これをサポートするためのデータキューブ演算が提案され [2] , データキューブを効率良く計算する手法の研究が行われている [10]〜 [13] . システムはユーザからの問い合わせを, 思考を妨げないようオンラインで答えられる速度が必要であり, この要求を満たすシステムが OLAP と呼ばれる [1] [14]. OLAP はバックエンドの構成により ROLAP, MOLAP に分類できる.

ROLAP システムは詳細なソースリレーションを格納した関係データベースをバックエンドに持つ. このため, ユーザのどのような問い合わせに対しても答えることができる. しかし, 基本的には問い合わせの度に集約演算を行うため応答時間が長くなり, 問い合わせの対象カラム数によってばらつきが生じる. この問題を解消するため, 頻繁に利用される集約演算は事前に計算を行うことが多い. この次元計算された集約演算結果は, 他の集約演算の計算にも利用でき, このことを考慮にいれ事前計算すべきビューの選択手法も提案されている [3], [4] . また，効果的な索引を用意することが難しく, [4] では，ビューとそれに対す

る複合索引の選択がターゲットとなっている.

OLAP が提供する多次元分析機能をよりダイレクトにサポートするのが MOLAP（Multi-dimensional OLAP) である. MOLAP において使用されるデータは，多次元データベースが管理する大規模な配列に格納される．このような大規模配列の記憶装置上での実装方式は数多く研究されている [5] [7] [9]. データの属性は配列の次元に対応している. データに n 個の属性が存在し，i 番目の属性の値の数を c_i とすると，この配列は，サイズが $(c_1, c_2, ..., c_n)$ の n 次元配列である．データまたはそれへの参照はその各属性の値に応じて，この n 次元配列の 1 つの要素に格納される．ところが，各属性値のすべての組合せについてデータが存在する場合は少なく，通常は配列の要素数に比較してデータの数はかなり少ない．したがって，データの充填率は低く無駄な記憶領域を消費することとなる．これは，配列要素の高速アクセスのために，アドレス関数により配列要素を線形に配置するためである．データを RDB のテーブルのレコードとして管理する ROLAP（Relational OLAP）の場合は，このような充填率の低下の問題は存在しない.

このような充填率の低い疎領域は一般的に属性の数（配列次元数）が増すにつれて，増大する．また，格納するデータが集約演算による階層構造の下位であるほど (詳細な区分のデータであるほど)，増大すると考えられる．この疎配列に対する対策として，chunk [5] [8] 単位でのデータ圧縮を行う．データの存在する配列要素のみを格納し，データベースサイズの低減を行う．多次元配列はメモリ上では，通常よく知られているように，あらかじめ定められた次元の順序（以後この順序を単に「次元順」と呼ぶ）に連続した領域として確保されている．この領域をそのままの順で 2 次記憶に格納したとする．この次元順に沿った検索を行う場合は，2 次記憶へのアクセス回数は最小限に抑えられるが，異なる次元順に検索をする場合，一回のページ読み込みで得られる有効データが少なく，多くのページアクセス回数が必要となる．このような検索時の次元依存性は，MOLAP では大きな問題である．MOLAP の主要な目的は，スライス&ダイス操作 [14] をはじめとするような多次元分析を対話的に行える環境の提供であり，ユーザからの要求に対して応答性を損なわないことが期待される．ユーザがオンラインで行う分析・スライス要求は，前もって特定できない．ダイス操作により分析視点を変更することで，スライスを行う次元軸が変更されるが，これによりレスポンスが大幅に悪化するようなシステムは好ましくない．すべての視点からのスライス要求に対し，一定範囲内の時間で応答を返すことが望まれる.

2. chunk 圧縮と次元依存性に対する対策

[5] では検索時の次元依存性を解消するため，管理する大規模な配列は “chunk” という単位に分割して取り扱っている. chunk もまた n 次元の配列であり，その大きさは 2 次記憶の 1 ページ以内を目安とする．大きさを 1 ページ以内とすることで，一回の読み込みで 1 chunk を取り出すことができる (図 1). 扱う単位を chunk とすることで，どの次元軸のスライス要求に対しても一回のページ読み込みで複数の隣接データを得ることができ，ページアクセス数の平均化を図ることができる.

一方，疎配列の問題を解決するための方法として，chunk 中の有効データのみを格納することにより圧縮を行うことが考えられる．このための圧縮方式として, chunk offset 方式 [5], bitmap 方式 [6] などが考えられるが，以下では圧縮方式を問わない.

ここでは，chunk データの圧縮を行いながら，複数の chunk を 2 次記憶の 1 ページに格納することが考える．ところが，chunk を格納する順序によっては，さらに新たな次元依存性が発生し得る．ここではこのように，疎配列問題を解消しつつ，この新たな次元依存性を極力解消するための方策を提案する.

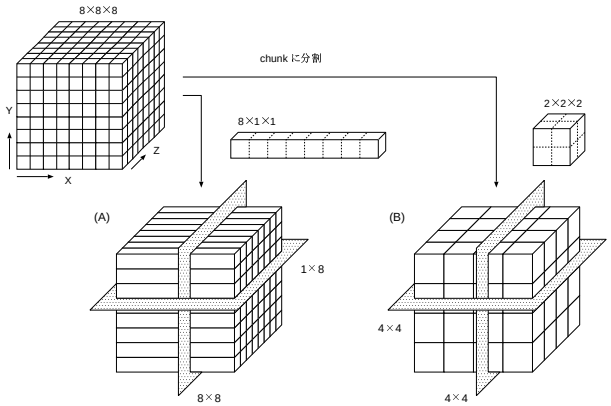

図 1　chunk

2.1 コンテナ化

chunk は 2 次記憶の 1 ページ分の論理サイズを占めるものとする．chunk が疎である場合，圧縮された chunk は 1 ページより小さくなるが，ここでは一つのページに複数の圧縮された chunk を配置するものとする．これにより，一度の読み込みで複数の chunk を読み込むことができる．この複数の圧縮 chunk を収容するための 2 次記憶の 1 ページを “chunk コンテナ”（以後，単にコンテナ）と呼ぶ．chunk は論理的な大きさの単位であるのに対して，コンテナは 2 次記憶中の物理的な大きさの単位である.

ここで，どの chunk を組合わせてコンテナに格納するかという問題が生じる．配列要素を通常どおり，次元順に線形に記憶領域に配置するのと同様に，圧縮 chunk を次元順に格納すると，新たな次元依存性が発生する．この次元依存性は，配列スライスに対する集約演算時に応答時間の標準偏差を悪化させる原因である．すなわち，ダイス操作による分析視点の変更により，スライスする次元に依存して応答時間にばらつきを生じ得る．この問題を解決するには，コンテナを構成する際に次元依存性を軽減するように圧縮 chunk を選ぶ必要がある.

コンテナの充填率を 100% に近づけると全体のコンテナ数が減少し，したがって，ディスクページの総数が減少する．MOLAP ではスライスとダイス操作が処理の中心となり，スライスでは，隣接した chunk 内の配列要素を必要とする．コンテナ化にあたっては，より近い chunk 同士を同一コンテナに格納することにより，スライスに対する集約演算のための検索を行うとき，ディスクページの読み込み回数が減少すると考えられる．したがって，なるべく隣接した chunk を選ぶと同時にコンテ

ナの充填率を上げるという２つの要求を満足する必要がある．

2.2 コンテナ化方式

[15] では chunk 集合としての配列に対していくつかの種類のコンテナ化方式を検討しシミュレーションを行っているコンテナ化方式の評価基準は，すべての配列スライスに対して，平均のディスクページ読み込み回数とその読み込み回数の標準偏差としている．以下ではそれらの中で双方ともに優れている方式を２つ記す．１回のコンテナ化の対象となり得る chunk は，コンテナ化ドメイン (以後単にドメイン) と呼ばれる各次元方向で一定個数の範囲内とする (図 2)．１回のコンテナ化が終われば，そのドメイン内の残りの chunk のコンテナ化はそこで，放棄し，ドメインを 1 chunk 分，定められた次元順に移動する．ここで，コンテナ化を放棄されたドメイン内 chunk はドメインの移動により，後のドメインのいずれかでコンテナ化されるので，最終的にすべての chunk がコンテナ化されることが保証される．

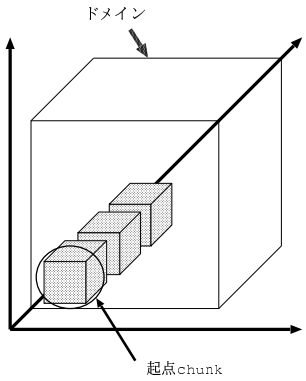

図 2　ドメイン

[A] Dynamic order(d-order)　コンテナ化の度に chunk を選択する際の次元の優先順位をコンテナ化の度にランダムに決定し，最優先次元方向の chunk から優先的にコンテナに詰める．次元の優先順位をチャンク化の度に変更することで，複数の chunk を１つのコンテナに詰めた場合の次元依存性の解消を図る．

優先次元方向へ chunk をコンテナに付け加えていった時，(a) ドメインのその次元での終端に至った場合，(b) 対象 chunk がすでにコンテナ化済みであり，他のコンテナに含まれる場合，(c) 対象 chunk を付け加えるとコンテナバッファの充填率が 100%を越える場合 は次の優先度の次元方向へ 1 つ分移動し同様の処理を行う．(c) の場合においても，その時点でコンテナ化を行うことはしない．最も優先度の低い次元について，処理が終わった時点，すなわちドメイン内のすべての chunk のコンテナ化可能性をチェックした時点で，コンテナ化を行う．続いて，ドメインの起点移動が行われ，同じアルゴリズムにより，新たなコンテナ化を進める．

例えば，図 3 のように横が最も優先度が高い次元で縦，高さと続く場合，まず横に進みながら chunk をコンテナに格納することになり，一列目の様になる．その次元で対象範囲内まで格納したので１つ拡張して二列目の処理になる．ここで，図のように途中の chunk がコンテナ化済あるいはその chunk を格納するとコンテナ充填率が 100%を越えるときにはすぐに次の列に拡張を行う．もし，列の最初の chunk がコンテナに格納出来ないときには，三番目に優先度が高い次元に拡張を行い，新たな面の原点から続けることになる．同じように面の原点でコンテナ化出来なければ，さらに次の次元に拡張して新たな立方体の原点から続けられ，さらに立方体の原点がコンテナ化出来なければ，その次の４次元目に拡張して新たな４次元の超立方体の原点から続けることになる．これを配列全体を構成する次元いっぱいまで行う．

このようにコンテナ化を行う理由はコンテナ化せずに残す chunk 集合を出来るだけ近接した直方形にすることにある．例えば，図 3 の上方半分は長方形の形で chunk が残ることになるし，面の原点でコンテナ化出来ず，次の立方体に処理が移るときでは今までの立方体でそこから上の部分，つまり直方体型の chunk 集合が残されることになる．

もし，ドメイン内の chunk の充填率が低く，ドメインのすべての chunk がコンテナ化された場合には，充填率の低いコンテナが生成されることとなる．逆に，ドメイン内の chunk の充填率が高い場合には，100%近い充填率のコンテナが生成されるが，ドメイン内の他の chunk はコンテナ化されないままである．ただし，ドメインの最後の chunk がコンテナに収まりきらない場合は，その chunk についてはコンテナ化を行わないこのためにコンテナの充填率が低くなる場合もある．この場合，これらの chunk は起点移動による新たなドメインに対するコンテナ化でカバーされることになる．

[B] 超立方体 (cube)　コンテナ化の度にランダムに優先次元を選択するのは d-order と同様である．ただし，現在の形 (n 次元の超立方体) から優先次元方向へ 1 chunk 分ずつ拡張してゆく．この時，各次元方向の chunk 数の差が 1 以下の範囲内で拡張する．現在の優先次元方向への拡張ができたら，次の優先方向へ超立方体の形状を保つように同様に拡張を行う．コンテナ化済みの chunk に出会ったときや充填率が高すぎてコンテナに格納するとコンテナの充填率が 100%を越える場合には，次の優先次元に切り換える．各次元方向の chunk 数の差が 1 以下になるので，コンテナの形状は超立方体に近い形に整えられる．これにより d-order と同様にコンテナ化せずに残す chunk を出来るだけ正方な形にし，コンテナ間の干渉を小さく抑える．

例えば，図 4 のように 2 次元配列の場合，まず 0 とある 起点 chunk を調べ，これがコンテナに格納可能ならば拡張が始まる．まず，優先次元方向に 1 chunk 拡張し 1 までの chunk がコンテナに格納される．つぎに２番めに高い優先次元に拡張して２までがコンテナに格納される．ここでは 2 次元配列なのでこれが最後の次元となり，次にもとの次元に最も高い優先度の次元に戻り３の chunk がコンテナに格納される．このような順番によってコンテナ化されていくが，もし７の様に拡張しようとした chunk 集合 (n 次元配列ならば $n-1$ 次元をなす超平面) の chunk すべてがコンテナに格納できない場合，この chunk 集合 (図では 7) をすべてコンテナには詰めず，この次元方向はこれ以上拡張しないことにする．他の次元方向についてはさら

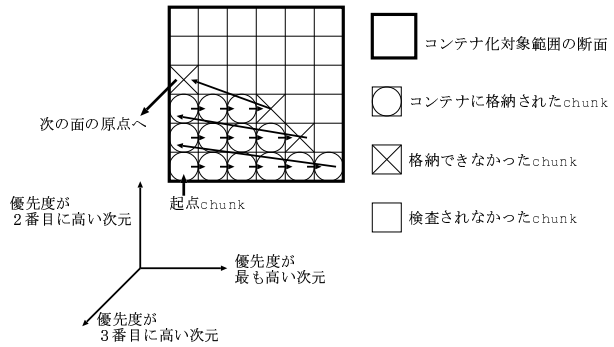

図 3　コンテナ化アルゴリズム d-order

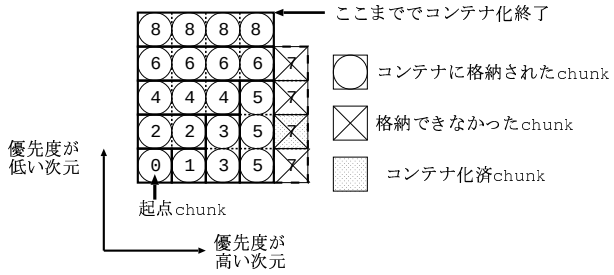

図 4　コンテナ化アルゴリズム cube

に拡張できるがそれもコンテナ化済の chunk によって拡張をしなくなった次元方向の chunk 数より 1 つ分大きい所 (図では 8) までで 1 つのコンテナを作成する. そのあと次のドメインに移動する.

3.　シミュレーションによる評価

前節では chunk 配列に対するコンテナ化方式を提案したが, ここではその有効性をシミュレーションにより評価する.

1 つの次元方向に 36 個の chunk を持つ 5 次元配列を用意する. したがって, 全体の chunk 数は, $36 \times 36 \times 36 \times 36 \times 36 = 60,466,176$ 個である. この 5 次元の chunk 配列に対し前述の 2 つのアルゴリズムによりコンテナ化を行い, i 次元スライス時 $(i = 1, \ldots, 4)$ のコンテナ平均読み込み回数とその標準偏差を測定する. ここで, n 次元配列の i 次元スライスとは, $n-i$ 個の次元の値を固定して i 次元分を可変にした時にできる i 次元超平面である. コンテナ平均読み込み回数はこのような超平面に含まれるコンテナの数をすべての i 次元超平面について平均したものである. 固定分のバリエーションが ${}_nC_i \times 36^{(5-n)}$ できるので, このような超平面の数は ${}_nC_i \times 36^{(5-n)}$ である. また, ドメインサイズは各次元 2 で $2^5 = 32$ chunk サイズとしている. 標準偏差値が低いほど次元依存性が解消されていることを示している.

シミュレータの入力として, 各 chunk のデータ充填率を格納した float 型の 1 次元配列を用いた. 各 chunk はすでに chunk offset や bit-map などの圧縮方式にて圧縮されているとし, シミュレーションでは chunk を処理単位とした. また, 出力として, コンテナ化の結果, chunk が格納されるコンテナ番号を整数型の 1 次元配列に chunk 毎に記録した.

3.1　実験結果

[15] では, chunk のデータ充填率が一様な場合を仮定した. ここでは, chunk のデータ充填率が一様でない場合においてスライス時のコンテナの平均読み込み回数とそれらの標準偏差を調べる. ある次元方向に対して, 正規分布によるデータ充填率の偏りがあると仮定する. これは年齢や時間に関するような, 連続した前後の関係がある次元を想定している.

3.1.1　1 つの次元に沿って偏りがある場合

5 つの次元のうち 1 つに対して (第 3 次元) 偏りを与え, 他の 4 つの次元については一様分布であるとする. x をその次元における chunk の位置としたとき, その chunk のデータ充填率 $y(\%)$ が $y = y_0 + ae^{-\frac{(x-\mu)^2}{2\sigma^2}}$ (y_0 はグラフの山の裾の高さ, μ は最も山が高くなる x の中央値, σ は山のなだらかさを表す標準偏差, a は裾から見た山の高さ) という正規分布式で与えられるような場合である. ただし, $y_0 = 10(\%)$ とした上で, 全ての chunk のデータ充填率の平均, $\bar{y} = 10(\%)$ となるよう y をスケーリングした. また, 一様でないことの影響をみるため, $\sigma = 1$, $a = 100(\%)$ と比較的, 急峻にした. さらに, 各次元方向の chunk 数が 36 なので, $\mu = 18$ とした. 図 5 と図 6 が 4 次元スライス時のコンテナの平均読み込み回数とその標準偏差を一様分布と正規分布について求めた結果である.

コンテナ読み込み回数の平均には大きな変化が認められない. これは一様かそうでないかということに関わらず比較対象にするために平均の充填率をどちらも 10%にしているためと思われ, 意図通りの結果といえる. 充填率の高い第 3 次元の中央付近ではコンテナ読み込み回数が増えるが, 周辺では一様な充填率の場合より読み込み回数は減少しているため, 平均すれば一様な場合とほぼ同じくらいになる.

一方, 標準偏差の方には, d-order と cube アルゴリズムにおいては, 一様な場合と違いが認められる. これは先に述べた通り第 3 次元の中央と周辺ではコンテナの読み込み回数に差が現れるためと, ドメインの位置によってはその中に含まれる chunk の充填率が場所により急激に変化するためと考えられる.

次元順の場合に標準偏差にあまり変化が認められないのは, 次元順では第 1 次元にそって chunk をコンテナに格納するため, 第 1 次元を固定してスライスするときにはほぼ常にコンテナ内の 1 つの chunk しかスライス断面に存在しないことになる. これが最悪のスライス断面の取り方であり, 平均読み込み回数はこの最悪の場合に大きな影響を受ける. この最悪のスライス断面の取り方では, データ充填率が一様な場合にくらべ, そうでない場合では, 読み込み回数が悪化することはない. 他のスライス断面の取り方では一様でない分布があると悪化するが, その変化は平均に向かって移動することになる. しかし, 最悪の場合の影響が支配的であるために, この変化が目立たないためだと考えられる.

双方の指標とも d-order と cube はデータ充填率に分布があっても, 次元順より圧倒的に優れていることを示している.

3.1.2　2 つの次元に沿って偏りがある場合

5 つの次元のうち第 2 次元と第 4 次元に偏りを与え, 他の

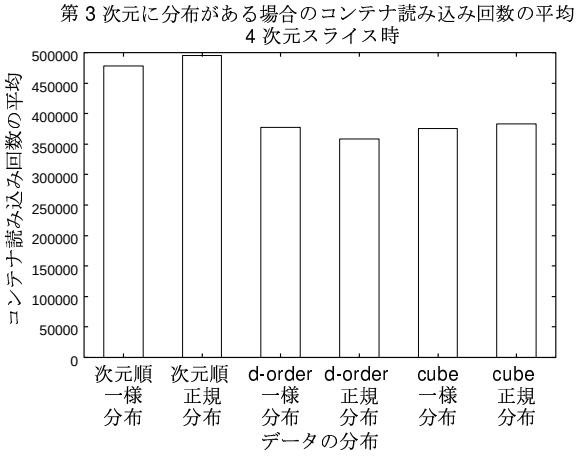

図 5　分布がある場合のコンテナ読み込み回数の平均

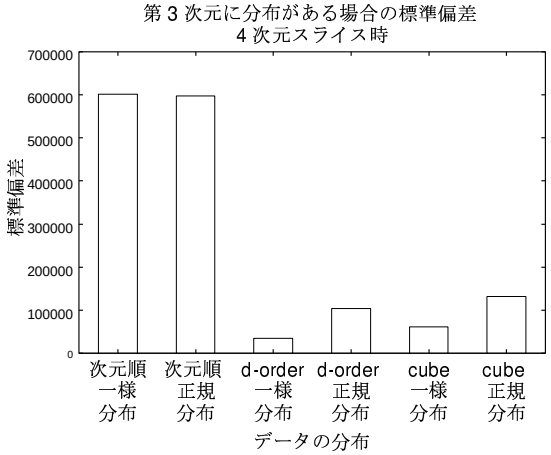

図 6　分布がある場合の標準偏差

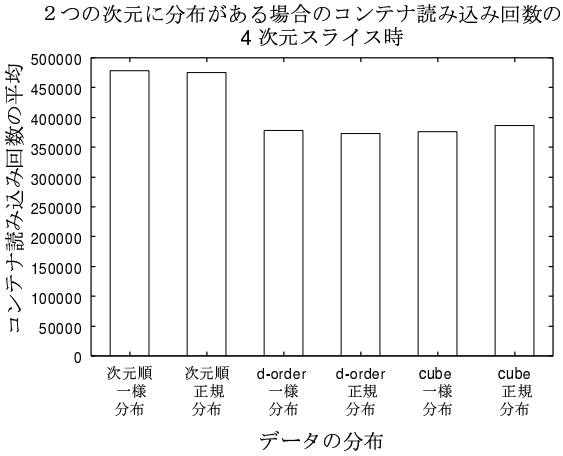

図 7　2 つの次元に分布がある場合のコンテナ読み込み回数の平均

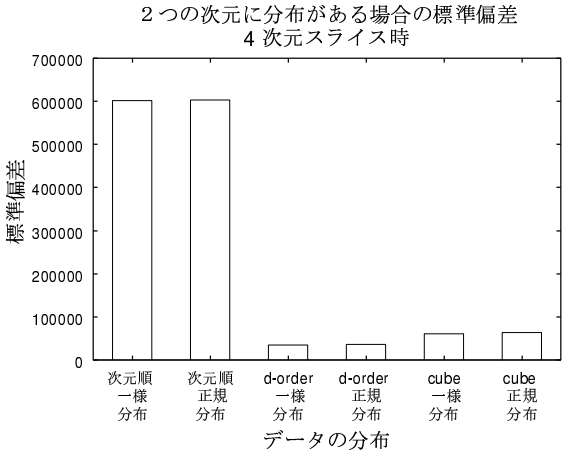

図 8　2 つの次元に分布がある場合の標準偏差

3 つの次元については一様分布であるとする．x_2 を第 2 次元における chunk の位置，x_4 を第 4 次元における chunk の位置としたとき，その chunk のデータ充填率 $y(\%)$ が $y = y_0 + ae^{-\frac{(x_2-\mu)^2+(x_4-\mu)^2}{2\sigma^2}}$ (y_0 はグラフの山の裾の高さ，μ は最も山が高くなる x_2 と x_4 の中央値，σ は山のなだらかさを表す標準偏差，a は裾から見た山の高さ) という正規分布式で与えられるような場合である．式の形から分かるように第 2 次元と第 4 次元の軸で構成される 2 次元平面上で円状にデータ密度が高い部分をもつ様な分布の偏りとなる．ただし，$y_0 = 10(\%)$ とした上で，全ての chunk のデータ充填率の平均，$\bar{y} = 10(\%)$ となるよう y をスケーリングした．また，一様でないことの影響をみるため，$\sigma = 1$，$a = 90(\%)$ と比較的，急峻にした．さらに，各次元方向の chunk 数が 36 なので，$\mu = 18$ とした．

コンテナ読み込み回数の平均と標準偏差のいずれの指標においても大きな差が認められない．これは，急峻であるために大きく充填率が変化する領域が 1 つの次元に分布がある場合よりも小さくなったためと思われる．d-order と cube は 2 つの次元にについてデータ充填率に分布があっても，次元順より圧倒的に優れていることを示している．

4. コンテナ作成の並列化

MOLAP の場合には大規模な多次元配列が想定されており，コンテナ作成には大きなコストがかかる．コンテナ化のアルゴリズムを並列化することによってこのコストは軽減される．ここでは，前節の 2 つのコンテナ化アルゴリズム d-order と cube の並列化を考える．前章のシミュレータを並列化するため，一般に実システムの並列化とは処理時間の減少の度合は異なる．コンテナ化はコンテナに格納する chunk を選ぶ部分と実際にコンテナを作成して 2 次記憶に格納する部分の 2 つに分かれる．本並列化シミュレータでは後者をシミュレートしていない．実際にはディスクキャッシュの量など，使用するシステムの I/O ストラテジーによっても変化するが，1 つのプロセッサがコンテナ化した chunk の数はそのプロセッサの入力 I/O の量にほぼ対応する．また，作成したコンテナの数はそのプロセッサの出力 I/O の量にほぼ対応すると考えられる．よって 2 次記憶の I/O が並列動作すると仮定し，並列化したシミュレータにおけるそれぞれのプロセッサのコンテナ化した chunk 数と処理したコンテナ数がプロセッサ間で均等でコンテナ総数が並列化前とあまり変化していないなら，実システムの I/O の処理時間は並列度にしたがって減少すると考えられる．

並列化するにあたってドメインがコンテナ化とともに chunk

配列内を定められた次元順に chunk 1つ分ずつ進んでいくことから，ある起点をもつドメイン R と次の位置を起点とするドメイン R' を処理するプロセッサ P, P' を別に用意し，P によりコンテナ化された chunk のマップを P' に順次送るパイプライン並列処理を行うことが考えられる. しかし，(1) コンテナ化のたびに次元優先順が動的に変化するために，P と P' の次元優先順は一般に異なること，(2) コンテナ化対象の chunk 集合は 1chunk 分の断面を除いて R と R' の共通部分であること，のために P' は次にコンテナに格納出来るかどうか調べたい chunk を P が決定するまで停止することになり，並列性が向上しない.

d-order と cube アルゴリズムにおいて，図 9 のように第 1 次元以外の軸に沿って両端から中心方向へと衝突するまでドメインを次元の順で移動・逆移動する方法によって並列化する. このようにするのは，あるドメインによるコンテナの出来方によってその近くの chunk を起点とするコンテナの作成が影響されるため，各プロセッサが処理する chunk の位置が互いに離れている方が並列性が阻害されず，またコンテナ化結果が並列化しない場合に近くなるからである. プロセッサ数が十分多い場合には，n 次元配列に対して $2^{(n-1)}$ 個の並列度で処理される．この方法では，配列内のデータ分布に偏りがあったとしても負荷分散が容易に行われるという利点がある.

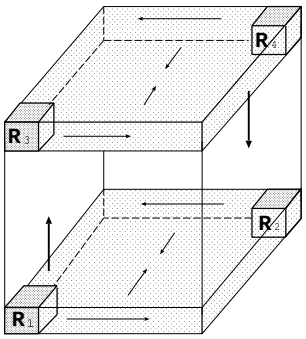

図 9　コンテナ作成の並列化

並列に処理するにあたって 1 つの chunk は 1 つのコンテナに格納されるようロッキングが必要になる. 並列化前と必ずしも同じコンテナ化結果となる必要がないとすればドメイン全体のロッキングを必ずしも必要としない. そこで，ロックの粒度として (1) 個々の chunk (2) 第 1 次元のドメインの幅の列の chunk 集合が考えられる (図 10).

図 10 において chunk 単位ロックの図で複数個同じ模様で表されているのはコンテナ化済 chunk である．ドメイン幅列単位ロックの図では論理的にロックされる chunk の範囲を表している．図の様に chunk 単位の場合，ロックする chunk はある時点においてプロセッサ毎にドメイン内の 1 つのみである. また列単位の場合実際にロック処理を行うのは列の先頭の chunk のみでよい. これで列全体の chunk が列同士で排他的になり，目的のロックが行われることになる.

(1) の場合には 2 つのドメインの起点が衝突するまでコンテナ化を chunk で構成される面でおこなうのでコンテナ化の負荷分散の精度が良い．並列性が高い反面，ロックする回数が多い分オーバヘッドを要する．逆に，(2) の場合ロック回数は少ないが，ロック粒度が大きいので負荷分散の精度は高くない.

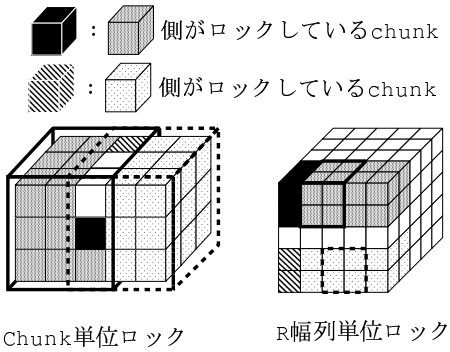

図 10　並列コンテナ化のロック粒度

実プロセッサの数には制限があり，したがって，一般に実並列度には上限があるが，コンテナ化したい多次元配列が n 次元で並列度が $2^{(n-1)}$ に満たない並列環境の場合には，上記の方法では chunk の面に 2 つのプロセッサを割り当てていたが，それを 1 つにすることで必要なプロセッサが半分になり，また，chunk の直方体に対して 1 つにすることで 4 分の 1 となる．この時，次元の低い方からプロセッサを減らしていけば，高い次元で対称な位置にあるドメインは配列内での距離が大きいため互いに干渉しにくくなる.

コンテナ化アルゴリズムの並列化の実装には共有メモリ型並列計算機（Sun Enterprise 5500(14CPU))) におけるマルチスレッドを用いた.

図 11 から図 14 に，2 種類のロック方法とデータ充填率が一様な場合と第 3 次元に分布がある場合の組合せににについて，d-order および cube の並列化アルゴリズムによるコンテナ化の実行時間を示す．いずれのロック方法にてもスレッド数の台数効果がよく表れている．また，ドメイン列単位のロッキングのオーバヘッドは chunk 単位のロッキングのものより少ないのでコンテナ化時間がより少ないことが観察できる.

5. おわりに

MOLAP において使用される多次元配列における疎配列と次元依存性を低減するためのアルゴリズムと，その並列化の方式を提案した．これらの問題を解決するために，配列を”chunk”と呼ばれる小さなブロックで管理し，その chunk 単位で圧縮を行なった．さらに，圧縮された chunk を 1 ページに収まるように複数まとめ，“chunk コンテナ” として 1 回の読み込みで複数の chunk を得られるようにした上で並列化を行った．それに際して疎配列と次元依存性の問題を再び悪化させることなく並列化すること，負荷分散がうまく行えることなど考慮した上で並列化方式を選択した.

[15] では，chunk のデータ充填率が一様な場合を仮定したがここではまず並列化前に，chunk のデータ充填率が一様でない場合においてスライス時のコンテナの平均読み込み回数とそれらの標準偏差を調べ，その様な場合においてもコンテナ化アル

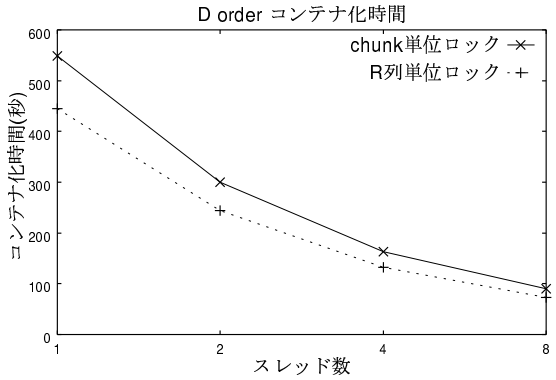

図 11　d-order のコンテナ化時間

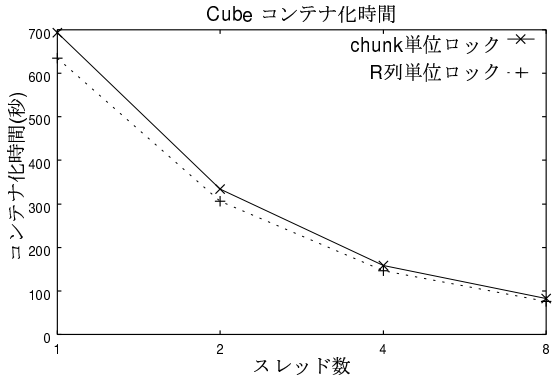

図 12　cube のコンテナ化時間

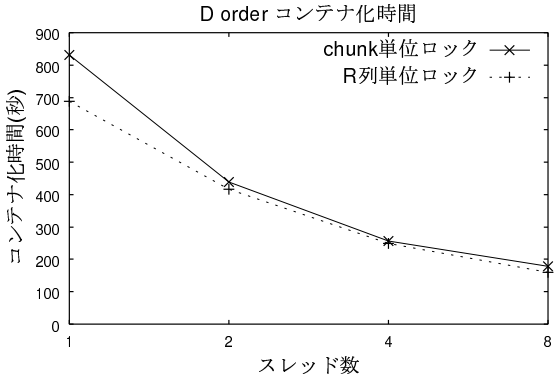

図 13　データ充填率に分布がある場合の d-order のコンテナ化時間

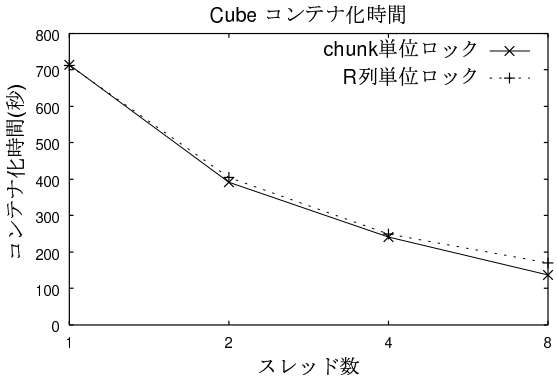

図 14　データ充填率に分布がある場合の cube のコンテナ化時間

ゴリズム d-order と cube は有効であることを確かめた．そして，提案した並列化手法においてはデータ充填率に分布があってもコンテナ化時間を削減できることを示した．

今後の課題として，コンテナ集合を複数のマシンに分散配置したときの並列処理の問題，多次元配列に対するビューの実体化の高速処理の問題などが上げられる．

文　献

[1] E.F.Codd, S.B.Codd, C.T.Salley, *"Beyond decision support"*, Computerworld, vol.27, no.30, July 1993.

[2] J.Gray, A.Bosworth, A.Layman, and H.Pirahesh, *"Data cube: A relational aggregation operator generalizing group-by, cross-tab, and sub-totals"*, Technical Report, Microsoft, Nov. 1995.

[3] V.Harinarayan, A.Rajaraman, and J.D.Ullman, *"Implementing data cubes efficiently"*, Proc. of the ACM SIGMOD Conference on Management of Data, Montreal, Canada, pp.205-216, June 1996.

[4] H.Gupta, V.Harinarayan, A.Rajaraman, and J.D.Ullman, *"Index Selection for OLAP"*, 13th ICDE, pp.208-219, Manchester, UK, April 1997.

[5] S. Sarawagi, M. Stonebraker, *"Efficient Organization of Large Multidimensional Arrays"*, Proc. of ICDE, pp.328–336, 1994.

[6] M.C. Wu, A.P. Buchmann, *"Encoded Bitmap Indexing for Data Warehouses"*, Proc. of ICDE, pp.220–230, 1998.

[7] Kent E. Seamons, Marianne Winslett, *"Physical Schemas for Large Multidimensional Arrays in Scientific Computing Applications"*, 1994 IEEE

[8] P.M.Deshpande, K.Ramasamy, A.Shukla, and J.F.Naughton, *"Caching multidimensional queries using chunks"*, ACM SIGMOD, p.259-270, Seattle, WA, june 1998.

[9] Yihong Zhao, Prasad M.Deshpande, Jeffry F.Naughton, *"An Array-Based Algorithm for Simultaneous Multidimensional Aggregates"*, SIGMOD '97 AZ,USA p.159-170

[10] 松澤 裕史, 福田 剛志, "複数の集約演算のための並列アルゴリズム", 電子情報通信学会論文誌 D-1 Vol. J82-D-1 No.1 pp98-110, January 1999.

[11] Sameet Agrawal, Rakesh Agrawel, Prasad M. Deshpande, Ashish Gupta, Jeffrey F. Naughton, Raghu Ramakrishnan, and Sunita Sarawagi, "On the Computation of Multidimensional Aggregates", Proc. of the 22nd VLDB Conference, Mumbai(Bombay), India, Sept. 1996.

[12] Sunita Sarawagi, Rakesh Agrawal, Ashish Gupta, *"On Computing the Data Cube"*, Research Report RH10026, IBM Almaden Research Center, 1996.

[13] Theodore Johnson, Dennis Shasha, "Hierarchically Split Cube Forests for Decision Support: description and tuned desgn", Working Paper, 1996.

[14] M.Abbey et.al., *"SQL Server 7 Data Warehousing"*, McGraw-Hill, 1999.

[15] T.Tsuji, A.Isshiki, T.Hochin, K.Higuchi, "An Implementation Scheme of Multidimensional Arrays for MOLAP", Proceedings of the 13th International Workshop on Database and Expert Systems Applications, pp.773-778, 2002.