社団法人　電子情報通信学会
THE INSTITUTE OF ELECTRONICS,
INFORMATION AND COMMUNICATION ENGINEERS

信学技報
TECHNICAL REPORT OF IEICE.

# 高機能メモリシステムを利用した主記憶データベースにおける索引の利用について

宮崎　　純†

† 奈良先端科学技術大学院大学 情報科学研究科
〒 630–0192 けいはんな学研都市
E-mail: †miyazaki@is.naist.jp

**あらまし**　本論文では，高機能メモリシステムを持つ主記憶データベースにおいて，索引の利用とその問題点について明らかにする．近年，プロセッサとメモリの間のアクセスギャップが大きくなり，これが主記憶上でのデータベース処理に大きく影響を及ぼしている．この問題に対処するため，我々は高機能メモリシステムを利用した主記憶データベースとその問合せ処理に関して研究を進めてきた．本稿では，本システム上での索引に関して，問合せ処理における性質とその限界について述べる．

**キーワード**　主記憶データベース，索引，高機能メモリシステム，問合せ処理

# On Use of Indexes for Main Memory Databases Using a Functional Memory System

Jun MIYAZAKI†

† Graduate School of Information Science, NAIST
Kansai Science City, Nara 630–0192 Japan
E-mail: †miyazaki@is.naist.jp

**Abstract**　We discuss advantage and disadvantage in use of indexes in a main memory database using a functional memory system. Recently, the access gap between a processor and memory becomes larger. This gap greatly affects database processing in main memory. To cope with the problem, we have proposed a functional memory system that helps query processing of main memory databases. In this paper, we show the characteristics of some indexes and their applicability and limitation to query processing.

**Key words**　main memory database, index, functional memory system, query processing

## 1. まえがき

既存のサーバ型計算機を利用するデータベースの多くは，大量のデータを扱う必要性から，従来からビット当たりのコストが低いディスクにデータを格納してきた．ディスクに対する入出力はメモリアクセス時間と比較して非常に低速であるため，従来のデータベースシステムはこの入出力を最適化することが最重要課題であり，CPU サイクルを有効に利用することに関しては，ほとんど注目されてこなかった．特に，PC や組込システム等の小規模計算機では，そもそもディスクを装備していないものあり，また，主記憶のバンド幅も非常に小さい．このような小規模計算機でのデータベース処理において，CPU サイクルの有効利用は性能に大きく影響を与える．

一方，計算機の主記憶に使われる DRAM は，急速に容量が増加すると同時にそのコストも下がってきており，主記憶上にデータベース全体を格納する主記憶データベース [4] が現実的なものとなりつつある．しかし，DRAM のアクセス速度は，近年の著しいプロセッサの速度向上と比較すれば，この 20 年間ほとんど改善が無かったに等しく，プロセッサのメモリへのアクセスが高コストとなる，いわゆる**メモリの壁**が問題となっている．これは，データベース処理においても顕著化している [1], [2].

キャッシュの利用はメモリの壁を解決する一手段であるが，データベースは扱うデータ量が多いため，キャッシュが有効に働かない場合が多い [1]．そのため，主記憶，すなわちメモリへのアクセスレイテンシを短縮することが重要となる．我々は，このような小規模計算機を利用したデータベース処理においてもメモリの壁の問題を解決するため，主記憶関係データベース向きのメモリアクセス方式として，固定ストライド間隔で配

置されたデータを効率よくアクセスする Stride Data Transfer (SDT) 方式，不定ストライドのデータでも効率良くアクセスする Bitmap-based Data Transfer (BDT) 方式，さらにメモリシステム内に比較器を導入し，比較結果のビットベクトルだけを CPU に返す Comparator-based Data Transfer (CMP) 方式を提案してきた [9], [14], [15]．しかしながら，これらのメモリアクセス方式は基本的にデータベーススキャンを効率化するものであり，一般のデータベース処理には索引が不可欠である．

本論文では，まず主記憶データベースにおける索引の比較を行う．その後，SDT，BDT ならびに CMP を利用したスキャンに基づく問合せ処理と，索引を利用した問合せ処理との関係について述べ，索引の利用範囲について明らかにする．

## 2. 主記憶データベースとデータアクセス

### 2.1 主記憶データベースと表の構成

主記憶データベースは，性能面だけでなくデータベースの物理設計に関しても利点がある．例えば，ディスクの特性に合わせたファイル編成よりも柔軟なデータ構造が可能であり，関係データベースの表は図 1 に示すようなデータ構造で編成可能である．このデータ構造では，各タプルの属性はプロセッサのアクセス単位であるデータサイズ (例えば 32 ビットアーキテクチャの場合は 32 ビットのワード) で扱い，1 ワードよりも大きい文字列等のオブジェクトの場合は，表中にはそのオブジェクトへのポインタを格納する．これによりタプルの各属性のデータサイズは計算機のアクセス単位に一致し，効率よくデータにアクセスできる．一方，文字列のようなオブジェクトは，表とは別の領域に格納する．重複するオブジェクトはポインタで同一のオブジェクト実体を指すようにすれば，データ領域の効率化ができる．

関係データベースでは，情報表現の単位がタプルであるため，各タプルを順にアドレス空間上に配置する N-ary Storage Model (NSM) [6] が一般的であり，本稿でも NSM を前提とする．例えば，図 1 では，上のタプルの左側の属性から順にアドレス空間上に格納される．

| account# | balance | type |
|---|---|---|
| 129459 | 2240585 | • → saving |
| 571652 | 4059 | • → checking |
| 134578 | 110597 | • → checking |
| 387216 | 919327 | • → saving |

Fig. 1　表のデータ構造例

### 2.2 主記憶データベースにおけるメモリの壁

メモリの壁の原因を考えると，メモリ自体のレイテンシだけでなく，CPU とメモリ間の低速なシステムバスを経由してデータが転送されることも一つの要因である．既存のキャッシュ指向のメモリアクセスは，アクセス要求のあったデータだけでなく，そのデータに近接するデータも含めてキャッシュラインサイズ分を一括して CPU に転送し，キャッシュに入れる．しかし，同一キャッシュライン中に計算に不要なデータも含まれる可能性があり，この計算に不要なデータを転送するために，低速なシステムバスを利用することは効率的であるとは言えない．データベース処理の場合，一般に空間局所性は高くはなく，キャッシュ指向のメモリアクセスは有効とは限らない．

そこで，メモリシステムを高機能化し，計算に不要なデータを低速なシステムバスに極力載せないデータアクセス方式である SDT，BDT および CMP を提案し，そのデータベース処理への応用に関して明らかにしてきた [9], [14], [15]．

### 2.3 主記憶データベース向けのメモリアクセス方式

DRAM のメモリアレイは，図 2 のように複数のバンクから構成される．データ読出し時には，バンク (Bank) アドレスならびにそのバンクの行 (Row) アドレスを与えた後，列 (Col) アドレスを指定すれば，データが読出される．一行のデータ全体は I/O バッファに入れられるため，内部では一行全体のデータを一括して扱うことが可能であり，内部のバンド幅は大きい．そのため，同一行に属する I/O バッファ中のデータは，原理的に Col アドレスを指定するだけで 1 クロックごとに連続して読出すことができる．

DRAM は通常，キャッシュ指向のデータアクセスに最適化されており，同一行内の連続する固定長 (バースト長) のデータを転送するバーストモードにより，キャッシュラインを埋めるためのまとまったデータをパイプライン的に転送する．

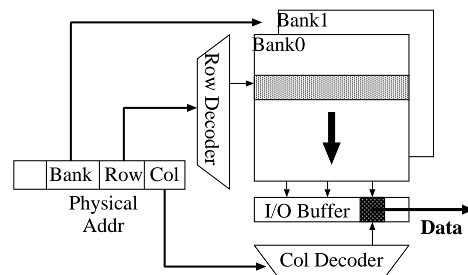


Fig. 2　DRAM の構造
Fig. 2　Structure of DRAM.

以降ではシステムバスの有効利用の観点から，I/O バッファ中から計算に必要なデータのみを CPU に転送するモデルを挙げ，そのデータベース処理への適用について述べる．

#### 2.3.1 Stride Data Transfer

通常利用される NSM モデルのデータベースでは，必要なデータが隣接したアドレスに存在しない場合が多い．図 1 のデータ構造の場合，特定の属性値を読み出す際は，固定ストライド間隔で格納されたワード単位のデータを参照することとなる．このような固定ストライド間隔のデータ読出しをパイプライン的に行なう方式を *Stride Data Transfer* (SDT) と呼ぶ [16](図 3 参照)．

SDT は，アクセスするデータの先頭アドレスとストライド間隔からメモリ内の物理アドレスをメモリコントローラで計算し，パイプライン的にメモリに与えることで，固定ストライド間隔のデータをパイプライン的に読出す方式であり，キャッシュ指向のメモリアクセスで問題となる不要なデータの読出しを避け

ることができる．SDT は，メモリの同一行内，つまり I/O バッファ内に存在するデータのみに有効であるが，近年の DRAM の行サイズは大きく，4KB のものも存在するため，SDT のパラメタ再設定は頻繁には起らない．

SDT により順次読出されたデータは，そのアドレスが連続ではないため，CPU に転送されたデータを通常のキャッシュに入れることは困難である．そこで，FIFO を CPU に設け，FIFO でデータを受け取る方式を採ることにする．このような非キャッシュ指向のデータを受け取るためのバッファは幾つか提案されており [13], [17]，多くの研究者に受け入れられているアイデアである．

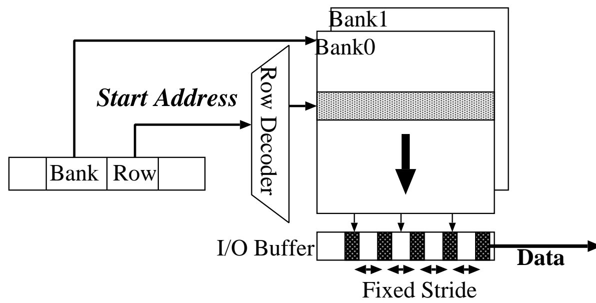


Fig. 3　SDT 方式

**2. 3. 2**　Bitmap-based Data Transfer

SDT はタプルが密に格納されており，ある一つの属性値を読出す場合に有効であるが，例えば，タプルごとに二つ以上の属性をスキャンする場合，削除されたタプルの空き空間が存在する場合，選択演算の後に結合演算を行う場合など，必ずしも固定ストライド間隔でデータをアクセスできるとは限らず，むしろストライド間隔が不定のアクセスパターンの方が一般的である．

このような不定ストライド間隔のアクセスパターンに対応するために，図 4 に示すように，I/O バッファ中の読出したいデータの位置をビットマップで指定し，ビットマップで指定されたデータのみをアクセスする方式を *Bitmap-based Data Transfer* (BDT) と呼ぶ [15]．例えば，32 ビットプロセッサを使用し，DRAM の 1 行が 4KB，すなわち 1024 ワードの場合，1024 ビットのビットマップを与えて I/O バッファ中から任意の位置のワードを連続して読出すモデルである．

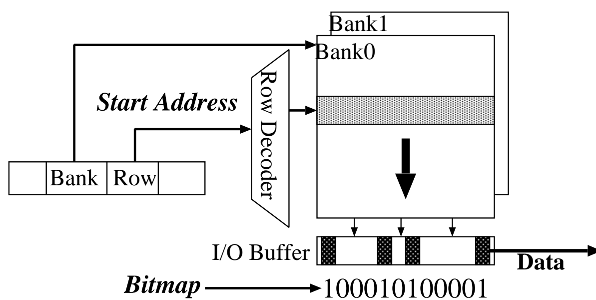


Fig. 4　BDT 方式

**2. 4　Comparator-based Data Transfer**

SDT ならびに BDT はいずれも，CPU がメモリからデータを読出す際に，メモリの壁の一原因である低速なシステムバスに，計算に必要な実データのみを載せることにより，データ転送の効率化を図るものである．しかし，問合せ処理で中心となる選択演算や結合演算は，多数の簡単な比較演算からなり，実データそのものよりも比較演算の結果の真偽値の方が重要である．

これらの演算を行うには，メモリ中に格納されている実データを CPU へ転送し，CPU 内での比較演算により 1 ビットで表現できる真偽値を決定する．通常，比較の対象となるデータはワードもしくはそれ以上の大きさからなり，メモリからこのようなデータを低速なシステムバスを通じて読出し，CPU 内での比較演算により 1 ビットに変換するのは，システムバスの有効利用という観点からすれば効率が悪い．SDT や BDT は，システムバスを効率よく利用して実データを CPU へ転送しているが，システムバス上で転送されるデータ量という観点からは効率的であるとは言えない．

メモリ中の I/O バッファは，ある一行のデータすべてが格納されるため，この I/O バッファに比較器を設けることにより，ある定数データと行内の全ての実データとの間で同時に比較演算を行うことができる．どのデータを比較の対象とするかを BDT と同様にビットマップで指定し，比較演算結果である真偽値からなるビットベクトルを CPU に転送する方式を *Comparator-based Data Transfer* (CMP) 方式と呼ぶ [14](図 5 参照)．

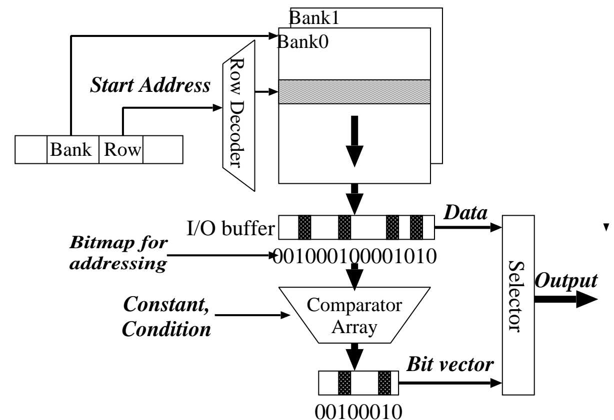


Fig. 5　CMP 方式

## 3. 主記憶データベースと索引

一般にデータベースには索引は不可欠である．主記憶データベースにおいても索引の利用により問合せ処理を効率化することが可能である．しかしながら，SDT，BDT，CMP のスキャンに基づく問合せ処理と比較して，索引がどの程度有効であるのかを明らかにする必要がある．そこで，比較器を有するメモリシステムを前提として，どのような索引が適切であるかについて，提案ハードウェアを利用した索引処理も含めて比較検討する．

主記憶データベース向けの索引は，いくつかの方式が提案されている [3], [7], [8], [10]〜[12]．その多くは B-tree に基づくものであり，本稿でも B-tree を前提とする．主記憶データベース中で B-tree アクセスを効率化するためには，キャッシュを意識した構成が有効である．

そのなかで，ABC-tree [12] と呼ばれる，キャッシュラインサイズのノードを一要素とする配列を利用した B-tree の構成方式がある．ABC-tree は，CSB+-tree [11] 等で利用される B-tree 中の子ノードを辿るためのポインタを廃止し，その代りに子ノードへのアクセスを配列の添字の計算に置き換えたものである．これにより，各ノードのファンアウトが増加し，木の高さを低くできるため，キャッシュの利用効率が高まり，CSB+-tree よりも B-tree アクセスが高速化される．

### 3.1 CMP を利用した B-tree

一般に，B-tree はファンアウトを大きくし，木の高さを低く抑えられれば，効率的なアクセスが可能である．キャッシュを意識した B-tree は，いずれもノードサイズが小さいため，ファンアウトも小さい．そこで，ノードサイズを大きくし，ファンアウトを大きくしつつ，それに伴うキーの比較の効率の低下を，CMP を利用してノード内のキーの一括比較により補う方式の検討を行う．

B-tree のノードは，キーと子ノードへのポインタの集合から構成される．検索キーはノード中の複数のキーと比較され，探索条件に合うキーを探し出し，そのキーに対応するポインタを辿り，子ノードを再帰的に探索する．この検索キーとノード中のキーとの比較は，CMP を用いて一括して行い効率化できる可能性がある．

B-tree のノード中の $n$ 番目のキー ($Key_n$) と $n+1$ 番目のキー ($Key_{n+1}$)，ならびに $n$ 番目のポインタ ($Ptr_n$) の子孫の任意キー値 ($\kappa$) との関係が，$Key_n \leqq \kappa < Key_{n+1}$ である時，B-tree の探索処理は以下の手順で行うことができる．

（1） CMP を利用して，ノード中の比較対象となるキーに対応する部分を “1”，それ以外を “0” とする入力ビットマップおよび検索キーから，検索キー以上の値を持つキーをノード中から探す．

（2） 出力ビットマップにおいて，最初に出現する “1” のビットを探すことにより，ノード中の条件を満す最小のキー位置を特定し，そのキーに対応するポインタを求める．

（3） (2) で得られたポインタを辿り，葉ノードに到着するまで再帰的に探索を行う．

図 6 を利用して以上の手順を例示すると，入力としてノード中のキーに対応するビットマップ ($\cdots 010101010 \cdots$) を与え，CMP により検索キーと比較した結果，出力ビットマップ ($\cdots 000001010 \cdots$) が得られたとき，ビット “1” に対応するキーは検索キー以上の値を持つことになる．検索キー値以上である条件を満す最小のキーは $Key_{n+2}$ となので，$Key_{n+2}$ に対応するポインタ $Ptr_{n+2}$ を利用して子ノードを辿る．

## 4. 性能評価

まず，キャッシュ指向のメモリアクセスによる ABC-tree と

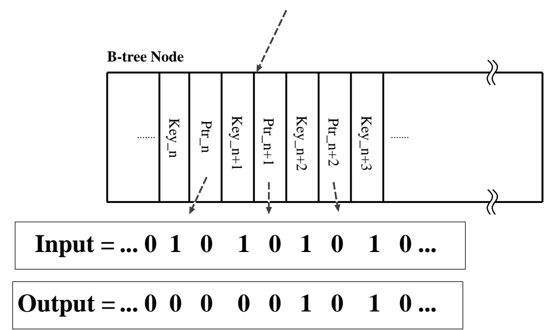


Fig. 6 CMP を利用した B-tree の探索

B-tree，ならびに CMP を利用した B-tree(以降 CMP-Btree と略す) の性能比較を行う．その後，索引を利用した問合せ処理の性質を明らかにするために，SDT，BDT，CMP を利用したスキャンベースの選択演算と，索引利用による選択演算の性能比較を行う．

システム構成として，図 7 で示すシングルチャネルのメモリを持つを仮定して性能を評価する．性能を計測するため，32 ビットの SPARC プロセッサシミュレータを利用し，実行に要した CPU サイクル数を計測することで評価した．利用可能なメモリアクセス方式は，キャッシュ指向の通常のメモリアクセス (Normal)，SDT，BDT ならびに CMP である．キャッシュ指向の Normal には DDR-SDRAM 相当のアクセスレイテンシ，それ以外の場合は SDR-SDRAM 相当のレイテンシを仮定する．なお，DRAM の I/O バッファサイズは 4KB に設定した．

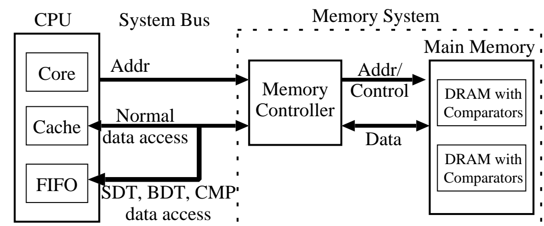


Fig. 7 システム構成

簡単化のため，CPU のモデルとして，小容量の一次キャッシュを持つものを仮定する．CPU には 8KB の 2-way セットアソシアティブのライトバック方式の命令キャッシュとデータキャッシュを備え，ラインサイズは 32 バイトとした．CPU サイクルとバスサイクルに関して，CPU の 1 命令実行を 1CPU クロックサイクルとし，バスクロックサイクルと CPU クロックサイクルとの比を $r$ とする．この時，メモリアクセスに必要な CPU クロック数を，

- キャッシュヒット時のレイテンシ: 1
- キャッシュミス時のレイテンシ: $10r$(SDR)，$8r$(DDR)
- キャッシュラインのメモリへの書込み時間: $14r$(SDR)，$10r$(DDR)
- メモリコントローラへのアクセスパラメタの設定: ワード当り $4r$
- BDT，SDT，CMP 転送の中断からの復帰のレイテンシ: $12r$

- FIFO 内のデータの読出し: ヒット時 1，ミス時 2 ∼ $r$

と設定した．なお，FIFO 内のデータの読出しミス時の最大レイテンシは，バスサイクルと CPU サイクルの比 $r$ で決定される．以降の実験では $r = 10$ とした．

キャッシュラインサイズは 32 バイトであるので，ABC-tree のノードサイズは 32 バイトとした．キーを 32 ビット整数とするとき，各ノードには最大で 8 個のキーを格納でき，ファンアウトは最大 9 となるが，多くの B-tree の研究ではノードの利用率として $\log 2$ がよく使われるため，各ノードには最大で 6 個のキーを格納した．

一方，CMP-Btree の場合は，CMP の効率を最大限に利用するため，ノードサイズを 4KB とした．B-tree も CMP-Btree に準じ，ノードサイズを 4KB とした．この時，ノードヘッダを除いた部分がデータ格納に利用可能であり，最大キー数は 510 個である．ノードの利用率を考慮して，各ノードには最大 391 個のキーを格納した．いずれの索引も完全一致問合せだけでなく範囲問合せも効率良く行えるよう，葉ノードには B+-tree と同様にサイドリンクを設けた．

### 4.1 索引の比較

ABC-tree，B-tree ならびに CMP-Btree を，完全一致問合せを利用して比較する．実験には，整数のデータからなる 10,000 個と 128,000 個の二種類のデータセットを用意した．前者のデータセットの場合，ABC-tree では木の高さは 4，B-tree，CMP-Btree では 2，後者の場合，ABC-tree が 6，B-tree，CMP-Btree では 3 である．データセットの 10%をランダムに選んだものをキーとして探索し，全ての探索が終了するまでに要した CPU サイクル数を計測した．

図 8 はデータ数が 10,000 個の場合，図 9 は 128,000 個の場合の結果である．両図から明らかなように，ABC-tree が最も性能が良いことが分かる．これは，本来の目標であるデータキャッシュのミスが少ないことによる．

一方，CMP-Btree は最も性能が悪く，通常のメモリアクセスによる B-tree よりも劣る．この原因は，ビットマップの処理のオーバヘッドである．子ノードを探すためのビットマップの処理コストは，ノード内のキー数 $k$ 対して $O(k)$ となる．具体的には，各ノードのキーに対応するビットマップの作成，ビットマップのメモリシステムへの設定，結果ビットマップの読出し，対応ポインタの特定の合計四つの処理のためのオーバヘッドが，全体の処理コストに対して支配的となっている．一方，通常のメモリアクセスを利用した B-tree では，ノード内のキーはソートされているためキーの探索に二分探索が利用でき，そのコストは $O(\log k)$ となる．351 個のキーでも高々9 回の比較で各ノードの探索が完了する．

### 4.2 範囲問合せにおける索引の効果

前節の実験結果より，ABC-tree が索引の候補の中で最適であることが分かった．本節では，現実的な問合せである範囲問合せを利用して，ABC-tree による問合せ処理と，スキャンベースの SDT，BDT，CMP による問合せ処理の比較を行う．

実験に利用したデータは，ウィスコンシンベンチマーク [5] で使用されるデータを利用するが，タプルサイズを 8 ワードとなるよう属性の一部を省略し，データの総数は 128,000 タプルとした．また SDT が利用可能なよう，タプルはメモリ空間上に密に格納した．

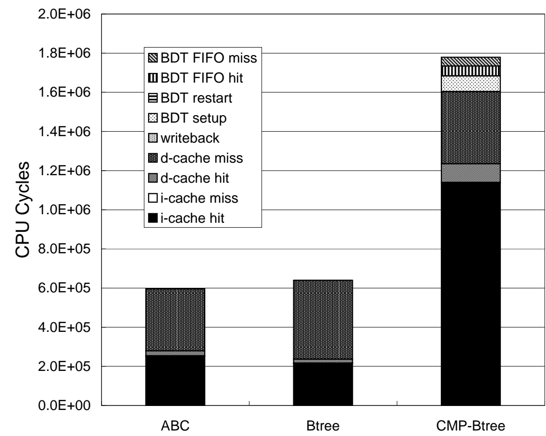


Fig. 8 索引の比較 (データ数: 10,000)

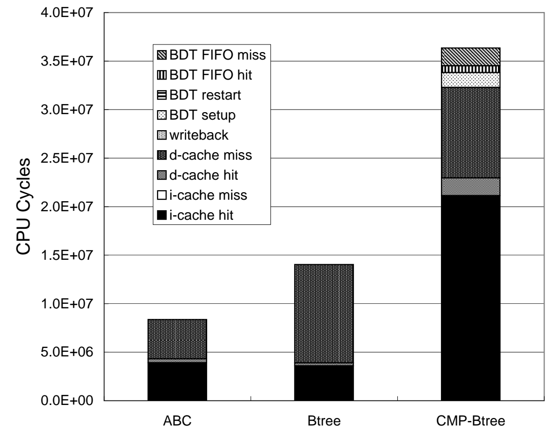


Fig. 9 索引の比較 (データ数: 128,000)

評価に利用した範囲問合せは以下の通りである．

```
SELECT *
FROM   R
WHERE  unique1 < CONSTANT
```

定数 CONSTANT により，問合せ結果の選択率を変化させて評価を行った．図 10 にその結果を示す．

一般に問合せ処理では，索引の利用は完全一致問合せだけでなく範囲問合せにおいても，多くの場合スキャンによる問合せよりも高速であるが，図 10 から明らかなように，索引が有効な範囲は，選択率が 10%未満の場合である．さらに注目すべきは，選択率が 75%を越えると，通常のキャッシュ指向のメモリアクセスで表をスキャンした方が効率が良い点である．以上より，SDT，BDT，CMP 等のメモリからのデータ読出しを効率化するハードウェアを備える主記憶データベースにおいても，索引は選択率のかなり小さな範囲であれば有効であることが明らかとなった．

CMP は B-tree の処理には向かなかったが，その原因である

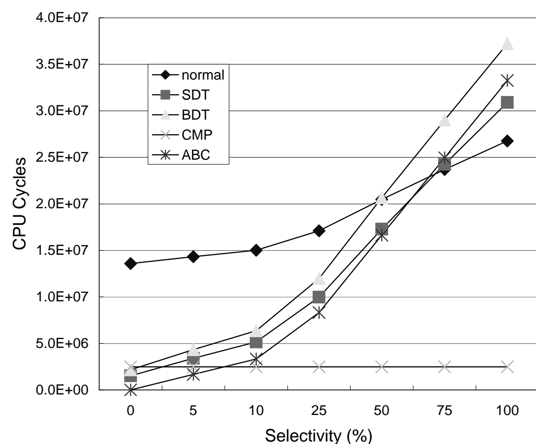


Fig. 10　範囲問合せによる問合せ処理の比較

子ノードへのポインタを探す処理の際に，CPU 内でのビットマップ計算ではなくメモリシステム内でのハードウェア処理により実現することにより，改善可能であると考えられる．一方，CMP を利用したスキャンによる問合せ処理では，比較演算の一括処理によるシステムバスの有効利用だけでなく，メモリからの読出しデータサイズの小ささから，問合せ結果の書込み時にキャッシュミスを起こしにくく，範囲問合せに非常に有効であることが明らかとなった．

## 5. ま と め

本論文では，小規模計算機を利用した主記憶データベースにおいて，索引の導入とその問合せ処理への適用範囲，ならびにスキャンによる問合せ処理との比較に関して議論した．

まず，B-tree，CMP を利用した B-tree，ならびにキャッシュを意識した索引である ABC-tree の比較を行い，それぞれの特性を明らかにし，ABC-tree が優れていることを確認した．しかし，CMP を利用した B-tree の処理は，子ノードのポインタの探索に高コストな CPU 上での線形探索ではなく，メモリシステム内でのハードウェア処理により実現可能と考えられる．

次に，範囲問合せ処理における ABC-tree，SDT，BDT，CMP の比較を行った．その結果，ハードウェアによる効率的なデータ読出し機構があっても，選択率が小さい場合には，やはり索引が有効であることを示すとともに，CMP がそのデータ圧縮効果によりデータ読出しと結果書込みの両方に効果を及ぼし，範囲問合せに非常に有効であることを明らかにした．

今後の課題として，関係データベースにだけでなく，XML データベースの主記憶上の処理とそのハードウェアサポートに関して検討していく予定である．

## 謝　　辞

本研究の一部は，科学技術振興機構戦略的創造研究推進事業 (さきがけ)「情報基盤と利用環境」による．ここに記して謝意を表す．

### References

[1] Anastassia Ailamaki, Divid J. DeWitt, Mark D. Hill, and David A. Wood. DBMSs on a Modern Processor: Where Does Time Go? In *Proceedings of VLDB '99*, pp. 266–277, 1999.

[2] Peter Boncz, Stefan Manegold, and Martin Kersten. Database Architecture Optimized for the new Bottleneck: Memory Access. In *Proceedings of VLDB '99*, pp. 54–65, 1999.

[3] Shimin Chen, Phillip B. Gibbons, and Todd C. Mowry. Improving Index Performance through Prefetching. In *Proceedings of ACM SIGMOD Conf. 2001*, pp. 235–246, 2001.

[4] Hector Garcia-Molina and Kenneth Salem. Main Memory Database Systems: An Overview. *IEEE Trans. Knowl. Data Eng.*, Vol. 4, No. 6, pp. 509–516, 1992.

[5] Jim Gray. *The Benchmark Handbook*. Morgan Kaufmann, 1993.

[6] Jim Gray, Andreas Reuter, 喜連川優監訳. トランザクション処理 –概念と技法– (下巻). 日経 BP 社, 2001.

[7] Kihong Kim, Sang Kyun Cha, and Keunjoo Kwon. Optimizing Multidimensional Index Trees for Main Memory Access. In *Proceedings of SIGMOD 2001*, pp. 139–150, 2001.

[8] Tobin J. Lehman and Michael J. Carey. A Study of Index Structures for Main Memory Database Management Systems. In *Proceedings of VLDB '86*, pp. 294–303, 1986.

[9] Jun Miyazaki. Query Optimization for Main Memory Databases Using a Memory System with a Comparator Array. In *Proceedings of DBWeb 2006*, pp. 281–288, 2006.

[10] Jun Rao and Kenneth A. Ross. Cache Conscious Indexing for Decision-Support in Main Memory. In *Proceedings of VLDB '99*, pp. 78–89, 1999.

[11] Jun Rao and Kenneth A. Ross. Making B+-tree Cache Consious in Main Memory. In *Proceedings of ACM SIGMOD Conf. 2000*, pp. 475–486, 2000.

[12] Hidehisa Takamizawa, Kazuyuki Nakajima, and Masayoshi Aritsugi. Array-based Cache Conscious Trees. 情報処理学会論文誌, Vol. 47, No. 1, pp. 217–230, 2006.

[13] Lixin Zhang, Zhen Fang, Mike Parker, Binu K. Mathew, Lambert Schaelicke, John B. Carter, Wilson C. Hsieh, and Sally A. McKee. The Impulse Memory Controller. *IEEE Trans. Computers*, Vol. 50, No. 11, pp. 1117–1132, 2001.

[14] 宮崎純. コンパレータアレイを有するメモリサブシステムを利用した主記憶データベース. 信学技法 DE2005-62, pp. 37–42, 2005.

[15] 宮崎純. ビットマップアクセスを利用した主記憶データベース処理. 情処研報 2005-DBS-137(I), pp. 245–252, 2005.

[16] 宮崎純, 府川智治, 田中清史. ストライドデータアクセスによる主記憶データベースの問合せ処理 の評価. 日本データベース学会 Letters, Vol. 3, No. 2, pp. 41–44, 2004.

[17] 近藤正, 中村宏, 朴泰祐. ハイパフォーマンスコンピューティング向けアーキテクチャ SCIMA. 情報処理学会論文誌, Vol. 41, No. SIG5(HPS1), pp. 15–27, 2001.