

# 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 Evaluation of Query Processing Using Stride Data Access in a Main Memory Database

宮崎 純<sup>♡</sup> 府川 智治<sup>◇</sup> 田中 清史<sup>▲</sup>

Jun MIYAZAKI Tomoharu FUKAWA  
Kiyofumi TANAKA

本論文では、我々の提案している、主記憶関係データベース向きのメモリアクセスを可能とする高機能メモリコントローラを使用することにより、不連続なデータアクセスが高速化され、その結果主記憶データベースの問合せ処理が効率的に行なわれることを示す。

**In this paper, we show that query processing in a main memory database can efficiently be executed by using a highly functional memory controller we have proposed, which enables faster memory access of discontinuously allocated data.**

## 1. はじめに

近年、データベースに蓄積される情報は飛躍的に増加しており、それに伴い、データベースへの問い合わせに要する時間も長くなりつつある。単純な問い合わせだけでなく、蓄積された大量のデータから有意な相関関係を解析したり、意思決定支援等の複雑かつアドホックな問い合わせは、インデックスが利用できない場合も多く、このような問い合わせの高速化が強く望まれている。

一方、近年の微細加工技術により、半導体メモリ、特に計算機の主記憶に使われる DRAM は急速にそのビット単価が下がってきている。このため、今までディスク上でしか格納できなかった巨大なデータベースを、主記憶上に全て格納する主記憶データベース [4] が現実なものとなりつつある。しかし、最近の著しいプロセッサの速度向上により、プロセッサとメモリのアクセス速度のギャップは開く一方である。つまり、主記憶にデータをアクセスすることが非常にコストのかかる処理であり、いわゆるメモリの壁の問題として知られる。

メモリの壁に対応するため、キャッシュを意識したデータ構造やプリフェッチ命令の効率的な利用等が提案されている。しかし、大規模なデータベースに対してはキャッシュが有効に働かず、また、連続アドレスのデータを先読みするプリフェッチ命令は、テーブル中のある属性のみをアクセスするといったデータベース処理

<sup>♡</sup> 正会員 奈良先端科学技術大学院大学情報科学研究科 / 科学技術振興機構さきがけ「情報基盤と利用環境」  
miyazaki@is.naist.jp

<sup>◇</sup> 非会員 NEC マイクロシステム株式会社 システム事業部  
tomo.fukawa@nms.necel.com

<sup>▲</sup> 非会員 北陸先端科学技術大学院大学情報科学研究科 / 科学技術振興機構さきがけ「機能と構成」  
kiyofumi@jaist.ac.jp

特有のメモリアクセスパターンには適さず、メモリアクセスのレイテンシの問題の解決とはならない。

これらの問題を解決するために、我々は主記憶関係データベース向きのメモリアクセスを可能とする高機能メモリコントローラを提案している [9, 10]。本論文では、ウィスコンシンベンチマークを利用して、この高機能メモリコントローラの使用が、現実的な問い合わせに対してどの程度有効であるかを明らかにする。

## 2. 主記憶データベース

### 2.1 主記憶データベースの特徴

主記憶データベースは、既存のディスクにデータを格納するデータベースとは異なり、全てのデータが主記憶中に格納される。このため、高速なデータアクセスが可能であり、実時間のデータベース処理が要求される応用に適している。これまでは、主記憶に利用される DRAM が小容量かつ高価であったため、主記憶データベースは非常に限られた分野でしか利用されていなかった。しかし、近年の DRAM の大容量化ならびにコスト低下により、PC 等の安価な計算機による高速な主記憶データベース処理が期待されている。

主記憶データベースは、処理速度に関してだけでなく、データベースの物理設計に関して一利ある。例えば、既存の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テーブルは図 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、各タブ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 32bit アーキテクチャの場合は 32bit ワード) のデータ型の場合はワード単位で扱い、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、そのオブジェクトへのポインタを格納する。これによりタブ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、テーブル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ことによりデータ領域の効率化が可能となる。それだけでなく、問い合わせ時のオブジェクトのマッチングもポインタの比較によりその等価性を容易に判定できる。

account#	balance	type
129459	2240585	●
571652	4059	●
134578	110597	●
387216	919327	●

● → saving  
● → checking

図 1: テーブルのデータ構造例

Fig. 1: A data structure for a database table

### 2.2 メモリの壁と主記憶データベース

主記憶を構成する DRAM のアクセス速度は、最近の著しいプロセッサの速度向上と比較すれば非常に低速であり、プロセッサとメモリ間のアクセスギャップは大きい。この問題は、メモリの壁として認識されており、解決法としてキャッシュ指向のデータ構造やプリフェッチ命令の利用が提案されている [7, 2]。しかし、アドホックな問い合わせのような時間局所性のほとんどない問い合わせの場合、キャッシュは有効に働かない。また、選択演算や結合演算時の属性値のスキャンは、既存のアーキテクチャのキャッシュライン指向のデータアクセスには適合しない。なぜなら、キャッシュライン指向では演算に不要なデータが多くアクセスされ、無効なデータ転送のためにシステムバスが占有されるからである (図 2)。システムバスは、CPU サイクルよりも低速で動作するため、無効なデータ転送は大きなペナルティとなる。

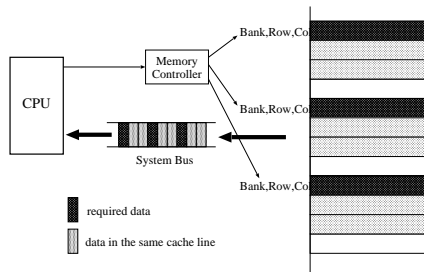


図 2: メモリアクセスとシステムバス  
Fig. 2: Ordinary memory access and system bus

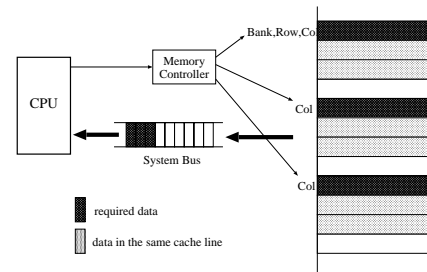


図 4: SDT アクセス時のシステムバス  
Fig. 4: System bus under the SDT access

### 3. 主記憶データベース向けメモリコントローラ

#### 3.1 DRAM のアクセス特性

既存の DRAM は、図 3 のように複数のメモリアレイのバンクから構成される。メモリアクセス時には、まずバンク (Bank) アドレスでバンクを指定する。次に、指定されたバンクの行 (Row) アドレスを与えた後、列 (Col) アドレスを指定し、CAS レイテンシ後にデータが読み出される。しかし、同一バンクの同一行に属するデータは、原理的に Col アドレスを指定するだけで良く、複数の Col アドレスを指定して、同一行内の任意のデータを 1 バスクロックごとに読み出すことが可能である。このようなアクセス方法はスタティックカラムモードとして知られている。

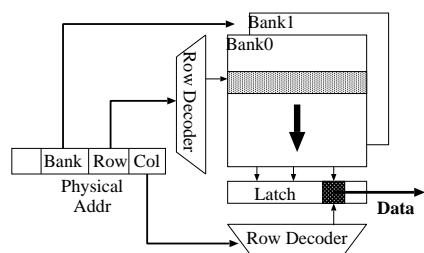


図 3: DRAM の構造  
Fig. 3: A structure of DRAM

#### 3.2 ストライドデータ転送 (SDT)

データベース処理では、次に必要なデータは必ずしも隣接したメモリアドレスにあるとは限らない。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み出しをパイプライン的に行なうために、我々の提案しているメモリコントローラはストライドデータ転送 (SDT) 方式を備えている [9, 10]。SDT は基本的にスタティックカラムモードを利用し、メモリコントローラは最初のデータの読み出しには Bank, Row アドレスを指定する必要があるが、それ以降は参照するデータが同一バンクの同一の行に存在する限り、メモリコントローラが次のデータの Col アドレスを自動計算し、その Col アドレスを DRAM に与えることで、等ストライド間隔のデータをパイプライン的に読み出す。

この SDT により、Bank および Row アドレスの再設定を行なうことなしに、図 4 のように必要とするデータのみがシステムバスに連続的に載るため、効率の良いメモリアクセスが行なえる。しかし、その反面、従来のキャッシュライン指向のアクセスとは異なるため、CPU に転送されたデータは通常のキャッシュに入れることが困難であるが、スキャンされる属性値は時間局所性が低いいため、キャッシュに入れるよりはむしろ FIFO を CPU に設け、FIFO でデータを受け取る方が適当である。この FIFO は、メモリコントローラが自動的にアドレスを計算してプリフェッチするデータを受け取るバッファとして動作する。

#### 3.3 SDT 時のメモリコントローラの動作

SDT 転送を行なう際の CPU とメモリコントローラ間のプロトコルを以下に示す。

- (1) SDT 開始前に、メモリコントローラのレジスタにパラメータ (転送データ数およびストライド長) を書き込む。
- (2) CPU が最初のストライドデータロード命令 (`sdt_ld`) を発行すれば、メモリコントローラが自動的にデータのアドレスを計算しメモリ上のデータを CPU の FIFO に転送する。
- (3) CPU はストライドデータロード命令を発行するたびに、FIFO のデータを指定されたレジスタに格納する。もしデータが FIFO に未到着であれば、データの到着を待つ。
- (4) (3) を指定された転送データ数の分だけ繰り返す

SDT で問題となるのは、同一の Bank, Row アドレスの境界を越えた場合や、SDT 転送中に通常のメモリアクセス命令が発行された場合である。同一の Bank, Row アドレス境界を越えた場合は、SDT 転送を一時中断し、次のストライドデータロード命令で再び SDT を開始する。SDT 転送中に通常のメモリアクセス命令が発行された場合も、SDT を一時中断し通常のメモリアクセスを優先させる。次のストライドデータロード命令が発行されれば SDT を再開する。

### 4. 性能評価

前節のメモリコントローラを用いて、ウィスコンシンベンチマーク [5] で主記憶データベース処理の性能向上を評価する。評価には SPARC プロセッサのシミュレータを用い、命令をトレースすることにより問い合わせ実行に要する CPU クロック数を計測することにより行なう。

#### 4.1 実験パラメータ

CPU は、それぞれ 8KB の 2 ウェイ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。なお、キャッシュの容量を公平とするため、SDT 転送時は、8KB のデータキャッシュ容量のうち 4KB を通常のデータキャッシュ、残りの 4KB を FIFO に利用するものとした。

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  $r$  を  $r = 10$  と設定した。この時、メモリアクセスのために必要なクロック数は、

- データの read/write ヒット時のレイテンシ: 1
- データの read ミス時のレイテンシ:  $12r$
- キャッシュラインの write 時間:  $14r$
- メモリコントローラへのパラメータの設定:  $4r$
- SDT 転送の開始ならびに再開時のレイテンシ:  $12r$
- FIFO 内のデータの read: ヒット時 1, ミス時  $2 \sim r$

と設定した。なお、FIFO 内データの read 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。

## 4.2 データベースと問合わせ

ウィスコンシンベンチマーク [5] のうち、データの挿入、更新、削除以外の問合わせを利用した。ウィスコンシンベンチマークのテーブルは、unique1(重複なし 4 バイト整数)、unique2(重複なし 4 バイト整数)、それ以外に 11 個の 4 バイト整数属性と、三つの 52 バイトの文字列属性を持つ。1 タプル 208 バイトであるが、図 1 のデータ構造を利用したとき、文字列はテーブルとは別の領域に格納されるため、1 タプル 60 バイトの規則的なテーブルとなる。テーブルは、10,000 個のタプルからなる tenk1、1,000 個のタプルからなる onek と Bprime から構成される。なお、本実験ではインデクスが利用できないアドホック問合わせを想定し、インデクスは利用しない。

実験に使用した問合わせは、以下の Q1~Q7 である。

```
(Q1) select *
      from tenk1
      where (unique2 > 301)
            and (unique2 < 402)

(Q2) select *
      from tenk1
      where (unique1 > 647)
            and (unique1 < 1648)

(Q3) select *
      from tenk1
      where unique2 = 2001

(Q4) select *
      from tenk1 t1, tenk1 t2
      where (t1.unique2 = t2.unique2)
            and (t2.unique2 < 1000)

(Q5) select *
      from tenk1 t, Bprime B
      where t.unique2 = B.unique2

(Q6) select t1.*, o.*
      from onek o, tenk1 t1, tenk1 t2
      where (o.unique2 = t1.unique2)
            and (t1.unique2 = t2.unique2)
            and (t1.unique2 < 1000)
            and (t2.unique2 < 1000)

(Q7) select MIN(unique2)
      from tenk1
```

Q1 は 1%、Q2 は 10%、Q3 は 0.1% の選択率となる選択演算である。Q4 は選択演算と結合演算の組み合わせ、Q5 は 2 way の結合演算、Q6 は 3 way の結合演算、Q7 は集約演算である。

結合演算には、通常主記憶データベースで使用される入れ子ループアルゴリズムを利用した。各演算結果の格納は、フィルタされたタプルへのポインタの配列として実現している。SDT によるデータアクセスでは、選択演算のループならびに結合演算の内側ループに SDT を適用することにより、データアクセスが効率よく行なわれるようにした。

## 4.3 実験結果

Q1~Q7 の問合わせを、通常のメモリアccess (Normal Access) と SDT によるメモリアccess (SDT Access) を用いたときの、実行完了までの CPU クロックサイクル数で比較したものを図 5 と図 6 に示す。

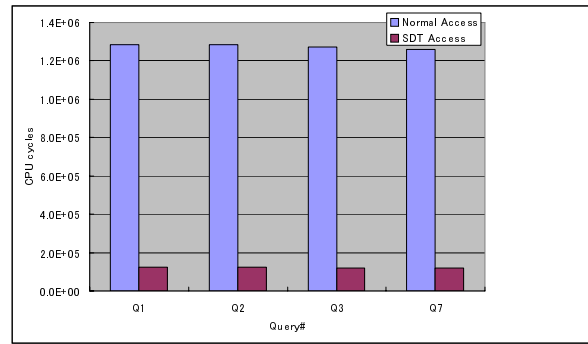


図 5: Q1,2,3,7 の CPU サイクル数  
Fig. 5: CPU cycles of Q1, 2, 3, and 7

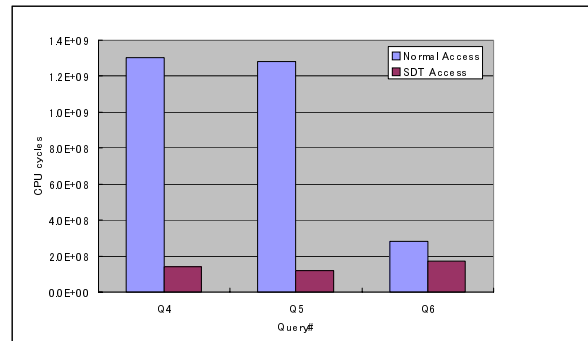


図 6: Q4,5,6 の CPU サイクル数  
Fig. 6: CPU cycles of Q4, 5, and 6

SDT の導入により、問合わせ処理速度が 8.9 倍 (Q2) ~ 10.7 倍 (Q3) 高速化された。例えば Q4 の CPU サイクル数の内訳 (図 7) を調べると、通常のデータアクセスでは、ほとんどがデータキャッシュのミスによるストールに時間が費やされている。一方、SDT によるデータアクセスでは、ストールする時間は全体の 1/4 以下となっている。これは、SDT により不連続データアクセスがパイプライン的に行なわれ、メモリアccessレイテンシが減少したことに加え、時間局所性の低いデータがノンキャッシュブルの FIFO に送られることにより、時間局所性の高いデータがキャッシュに高い確率で残ることが大きな要因である。

一方、Q6 は SDT が通常のメモリアccessに比べて 1.6 倍程度の高速化に留まっている。Q6 の CPU サイクル数の内訳を示す図 8 より、通常のデータアクセス、SDT によるデータアクセスの

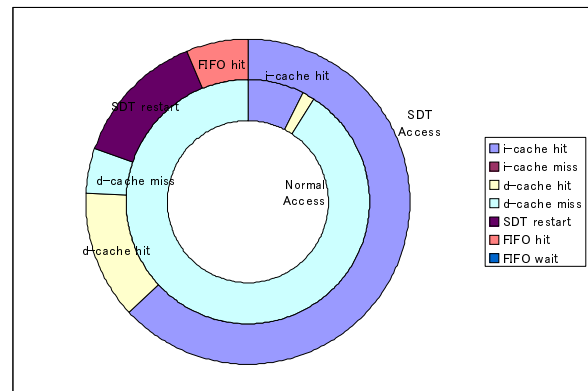


図 7: Q4 の CPU サイクル数の内訳  
Fig. 7: A breakdown of CPU cycles of Q4

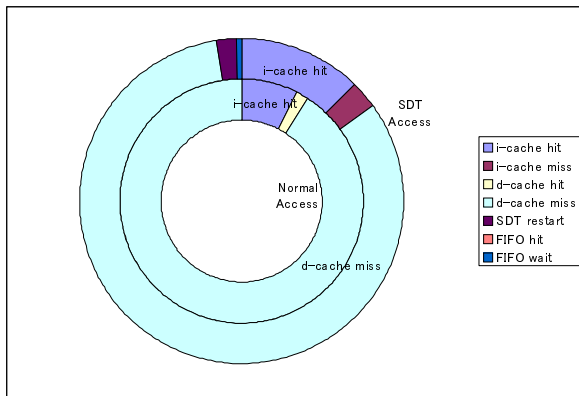


図 8: Q6 の CPU サイクル数の内訳

Fig. 8: A breakdown of CPU cycles of Q6

どちらもデータキャッシュのミスが全体の CPU サイクル数を支配している。この原因は、演算結果がタプルへのポインタの配列として表現されるため、タプルへのアクセスが間接参照となるが、SDT が間接参照に対して無効であることによる。この欠点を補うための手法として、例えば、選択演算の結果をポインタだけでなく、後の結合演算に必要なキーとともに連続メモリ空間にコピーしておくことが挙げられる。これにより、結合演算時にこのキーを SDT で読み出すことができる。しかし、キー値のコピーはメモリへの書き込みを伴うため、トレードオフとなる。

## 5. 関連研究

主記憶データベースシステムには、1980 年代に実験的に作成された IBM の OBE や、Wisconsin 大学の MM-DBMS、Princeton 大学の TPK や System M 等が存在するが、並行処理制御方式の工夫やデータ保護のためのログ処理等に重点が置かれており [4]、当時はメモリの壁の問題が無かったため、メモリ読み出しの効率化は考慮されていなかった。

一方、メモリの壁に対応するため、キャッシュを意識した索引の研究が AT&T の Rao らによってなされたが [6, 7]、キャッシュを意識したデータ構成でもプリフェッチを行わなければメモリレイテンシのために性能が改善されないことが Chen らによって指摘された [2]。これは、各データに近い将来読まれるデータへのポインタを持たせ、このポインタに基づいてプリフェッチを行うものである。しかし、アドレスバスにプリフェッチするデータのアドレス情報を載せる必要があり、データフェッチごとにアクセスレイテンシが印加される。2.2 節で述べたように、属性値のスキャンは空間局所性が低く、キャッシュライン指向には適合しないため、このプリフェッチによる効率化は困難である。

一方、ハードウェアの観点からの試みとして、Impulse[1]、SMC[3] ならびに田邊らの研究 [8] がある。Impulse[1] は不連続データをエイリアスを利用して仮想的に連続アクセスとして扱うが、多段アドレス変換によるオーバーヘッドは大きい。SMC[3] は、DRAM の Col アドレスを連続発行して同一の行データを連続して取り出すスタティックカラムモードを利用している点で本研究の DRAM アクセス機構と類似している。しかし本研究のように FIFO を用いて時間局所性の無いデータをうまく扱うことはできない。田邊らのプリフェッチ機能付きのメモリモジュール [8] はキャッシュブルアクセスを基本としており、SMC と同様の理由で主記憶データベース処理に適合しない。

## 6. おわりに

本稿では、主記憶データベースにおけるメモリの壁の問題を解決するために、不連続なデータアクセスをパイプライン的に転送する機能を持たせた高機能メモリコントローラを用いて、ウィス

コンシンベンチマークにて問合わせ処理の評価を行なった。

その結果、通常のメモリアクセスと比較して、3 way の結合演算の場合を除き、8.9 倍 ~ 10.7 倍の高速な問合わせ処理が可能であることを明らかにした。本手法は基本的にテーブルのフルスキャンの高速化であり、インデクスを利用できないアドホックな問合わせに高い効果を示す。

今後の課題として、マルチウェイの結合演算を高速化するためのデータ構造の改良、ならびに主記憶データベース向けのインデクスを利用した場合のインデクススキャンと SDT の組み合わせの最適化が挙げられる。

## 【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 (さががけ)「情報基盤と利用環境」、科学研究費補助金 (課題番号: 15700090) の支援による。ここに記して謝意を表す。

## 【文献】

- [1] J. Carter and W. Hsieh et al. Impulse: Building a Smarter Memory Controller. In *Proceedings of the 5th HPCA*, pp. 70–79, 1999.
- [2] 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.
- [3] S. McKee et al. Design and Evaluation of Dynamic Access Ordering Hardware. In *Proceedings of the 10th ICS*, pp. 125–132, 1996.
- [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] Jun Rao and Kenneth A. Ross. Cache Conscious Indexing for Decision-Support in Main Memory. In *Proceedings of VLDB '99*, pp. 78–89, 1999.
- [7] Jun Rao and Kenneth A. Ross. Making B+-tree Cache Conscious in Main Memory. In *Proceedings of ACM SIGMOD Conf. 2000*, pp. 475–486, 2000.
- [8] 田邊昇, 中武正繁, 箱崎博孝, 土肥康孝, 中條拓伯, 天野英晴. プリフェッチ機能付きメモリモジュールによる不連続アクセスの連続化. 情報処理学会 HOKKE-2004, pp. 139–144, 2004.
- [9] 府川智治, 田中清史, 宮崎純. 主記憶データベース向け高機能メモリコントローラの実現方式. 情報研報 ARC Vol.2002, No.112, pp. 77–82, 2002.
- [10] 府川智治, 田中清史, 宮崎純. 主記憶データベース向け高機能メモリコントローラの性能評価. 情報処理学会 HOKKE-2003, 2003.

宮崎 純 Jun MIYAZAKI

奈良先端科学技術大学院大学情報科学研究科助教授。データベースシステムの研究に従事。情報処理学会、電子情報通信学会、日本データベース学会、ACM SIGMOD、IEEE CS 各会員。

府川 智治 Tomoharu FUKAWA

北陸先端科学技術大学院大学情報科学研究科修了。現在、NEC マイクロシステム株式会社システム事業部に勤務。

田中 清史 Kiyofumi TANAKA

北陸先端科学技術大学院大学情報科学研究科助教授。計算機アーキテクチャの研究に従事。情報処理学会会員。