計算機アーキテクチャ 152－15
（２００３．３．１０）

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

府川智治[†] 田中清史[†,††] 宮崎純[†]

近年，プロセッサと主記憶間の性能格差によるボトルネックの解消および大規模データに対するデータベースの高速な質問処理が要求されている．本稿では高速大規模データ転送方式として，主記憶データベースのリレーション構成を利用した方式と DRAM のハードウェア構造を利用した方式を提案する．これらを質問処理に適用してシミュレーションによる評価を行う．

Evaluation of Highly Functional Memory Controller for Main Memory Database

Tomoharu Fukawa,[†] Kiyofumi Tanaka[†,††] and Jun Miyazaki[†]

It is required to alleviate a bottleneck caused by a speed gap between a CPU and main memory and respond promptly to queries for large data in database systems. In this paper we propose the fast and large scale data transfer methods that take advantage of relational data structure and characteristics of DRAM, and evaluate them in simulations for query processing.

1. はじめに

近年データの大規模化により，データベースの応答時間が増大してきており，質問処理の高速化が重要になっている．一方，半導体技術の発達により主記憶装置である DRAM が大容量化，低価格化し，従来はディスク上に格納していた大量のデータを主記憶内に格納する主記憶データベース（Main Memory Database System，MMDB）[1] の実現が可能になってきた．MMDB はディスク格納型データベースに比べて高速なデータアクセスが可能であり，高リアルタイム性のデータベースシステムに使用される．

本研究ではリレーショナルデータベースシステムを高速化するために質問処理時間を短縮化する手法に着目する．質問処理時間にはプロセッサのデータ参照時間とデータベース演算時間が含まれる．データ参照時間は与えられた質問に対してプロセッサがメモリからタプルを読み出す時間であり，データベース演算時間は主に条件に一致したデータを抽出するための実行時間である．大規模なリレーションで特定の属性を走査する場合，メモリ内では空間的・時間的局所性が共に存在しない．すなわち，プロセッサが有するキャッシュを有効に機能させることが困難である．また，MMDB 方式ではメモリ使用量を削減するためにプロセッサはポインタを介したデータアクセスを行う．しかしこの際，データの実体と共にポインタもキャッシュに格納されるため，キャッシュのヒット率が低下し，メモリアクセスのオーバヘッドが増大する．

本稿ではデータ参照時間の削減のために，メモリ空間に一定間隔で存在するデータをメモリからプロセッサに連続して転送する方式と，ポインタを介した二段階のアクセスに対してプロセッサが見かけ上一回のメモリアクセスでデータを取得する方式の 2 つの方式を提案する．また，これらを実現する機構をメモリコントローラ（以下，MC）に組み込み，データ参照時間を最小限に抑えることを示す．

2. 主記憶データベース（MMDB）

MMDB は主記憶内にリレーションを格納するため，従来のディスクを使用したデータベースシステムより高速なデータアクセスが可能である．また，主記憶内にデータを格納することでランダムアクセスが発生した場合でもアクセス時間を低下させることなく，一定時間内に大量データを処理できるため高速リアルタイム処理に適している☆．

図 1 に MMDB 方式におけるリレーション構成を示す．メモリ内に展開されたリレーション内の各セルには実体へのアドレス（ポインタ）が格納され☆☆，セル間で共通の実体は一つのみ存在する．サイズの大きい

† 北陸先端科学技術大学院大学 情報科学研究科
School of Information Science, Japan Advanced Institute of Science and Technology
†† 科学技術振興事業団，さきがけ研究 21,「機能と構成」領域
“Information and Systems”, PRESTO, Japan Science and Technology Corporation(JST)

☆ ただし主記憶に使用される DRAM は揮発性であるため，障害によるデータやトランザクションの消失を防ぐために，定期的にディスクなどの補助記憶装置にバックアップをとる必要がある．
☆☆ ただしポインタサイズ以下のデータはリレーションに直接格納する．

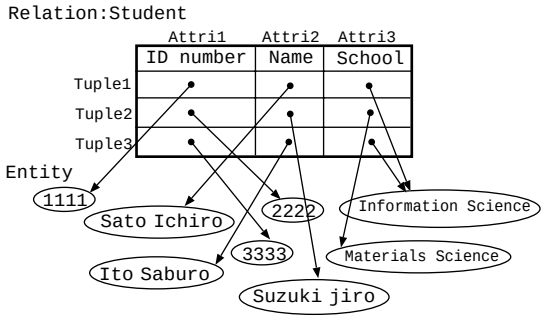

図 1 MMDB 方式のリレーション構成

共通データがリレーション内に頻出する場合，各セルにその共通アドレスを格納することでメモリ資源の効率化が可能である．このことから，質問処理を実行する場合，ポインタ比較によってデータの一致／不一致を判定することが可能であるが，データの大小関係などは実体同士の比較によってのみ得られるため，実体へのポインタを取得しそのポインタを使って実体へアクセスする必要がある．

3. データ転送方式

本節ではデータアクセス時間の短縮のために DRAM のハードウェア特性を考慮した高速データ転送方式である Stride Data Tranfser 方式と，MMDB によるポインタを介した二段階アクセスを高速化する Two-Phase Data Transfer 方式を提案する．

3.1 Stride Data Tranfser（SDT）

リレーションを格納する主記憶装置としては DRAM の使用が主流となっている．現在の DRAM のメモリアレイは複数のバンクから構成されており，それぞれのバンクは独立して動作可能である．DRAM にアクセスする際，メモリアドレスのバンク（Bank）アドレスで一つのバンクを選択し，そのバンクに対し行（Row）アドレスを与え，続いて列（Col）アドレスを与えることによって CAS レイテンシ後にデータが読み出される．ここで，Bank，Row アドレスを指定した状態で複数の Col アドレスを連続して与えることにより該当するデータが連続して出力される．しかし MMDB において同一 Bank，Row アドレス内に一定間隔で存在する複数の不連続なデータをアクセスする際，従来の MC では各データを含むキャッシュブロック単位でアクセスし，それぞれのブロックに対して Bank，Row アドレスを毎回指定するため効率が悪い．

Stride Data Transfer（SDT）方式はメモリ空間に一定間隔毎に存在するデータを連続して転送する機構である．リレーションをタプルの集合で管理する場合，ある属性に対して条件探索する際に一定間隔（タプルサイズ）毎のメモリアクセスが発生する．このとき，MC は最初の属性データを Bank，Row および Col アドレスを与えることにより読み出すが，それ以降の同一 Bank，Row アドレスに存在する各属性データは対応する Col アドレスのみの指定で読み出す．すなわち，MC が間隔値を加算して Col アドレスを自動生成することにより，同一 Bank，Row に存在するデータに対して連続アクセスが可能となる．

従来の MC と SDT のデータ転送方式の比較を**図 2** に示す．SDT により，MC と DRAM 間での Bank，Row アドレスの再入力の除去によるデータの連続読み出し，およびプロセッサと MC 間のメモリリクエストの一括化によりデータ転送効率を向上させる．

3.2 Two-Phase Data Tranfser（TPDT）

MMDB ではプロセッサが主記憶上の実体にアクセスするとき，メモリ資源の効率化のためにポインタを介した二段階のメモリアクセスを行う．プロセッサとメモリの速度差が大きくなっている現在の計算機システムではメモリアクセス時間を短縮することが重要課題である．またプロセッサは実体だけでなくその実体へのポインタもキャッシュに格納するためヒット率が低下し，その結果メモリアクセスが増大する．

Two-Phase Data Transfer（TPDT）方式はこの二段階アクセスを回避するために，プロセッサからは見かけ上，一回のメモリアクセスでデータを取得するデータ転送方式である．実体同士の比較を行う場合，あるいはポインタテーブルを介した条件探索を行う場合で使用する．ポインタテーブルとは，一つの質問処理で複数の探索条件を指定する場合，ある探索条件で一致したタプルのアドレスを格納する一時的なリレーションである．

従来の MC を介する方式と TPDT 方式の比較を**図 3** に示す．TPDT を実現する MC は第一段階のメモリアクセスでポインタを取得し，その値を使用して第二段階のメモリアクセスを行い，読み出した実体データをプロセッサに転送する．この間，ポインタはプロセッサに転送されない．これによりデータアクセスレイテンシが減少し，ポインタがキャッシュに格納されないためキャッシュの使用効率が向上する．

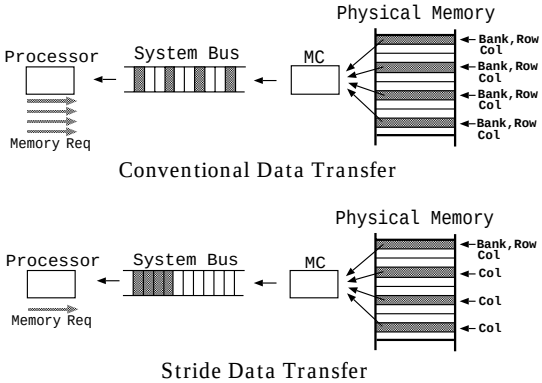

図 2 SDT 方式によるデータ転送

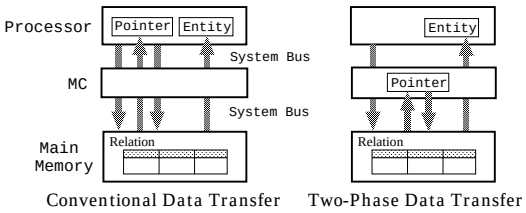

図 3 TPDT 方式によるデータ転送

4. メモリコントローラの実装

4.1 概　　要

提案するデータ転送方式を実現する MC を設計した．そのブロック図を**図 4** に示す． Command Interface，Address Generator，Command Generator，Data Path の 4 つのモジュールから構成される．

- Command Interface
  プロセッサからメモリリクエストとアドレスを受け取り，DRAM の動作コマンドとアクセスモードを決定し，Command Generator に送る．
- Address Generator
  Stride Address Generator（SDT AG）と Pointer Address Generator（TPDT AG）を内蔵し，それぞれ SDT および TPDT を実現する．詳細は 4.3 節で説明する．
- Command Generator
  Command Interface からのコマンドとアクセスモード，および Address Generator で生成されたアドレスを受け取り，DRAM への制御信号を生成する．またメモリアクセスの結果，Address Generator に Acknowledge（ACK）信号を返す．
- Data Path
  プロセッサと DRAM 間のデータ転送および TPDT における転送の終了判定を行う（4.3 節）．

Command Interface，Command Generator および Data Path は従来の MC に存在するモジュールであり，Address Generator モジュールを追加し SDT および TPDT を実現する．

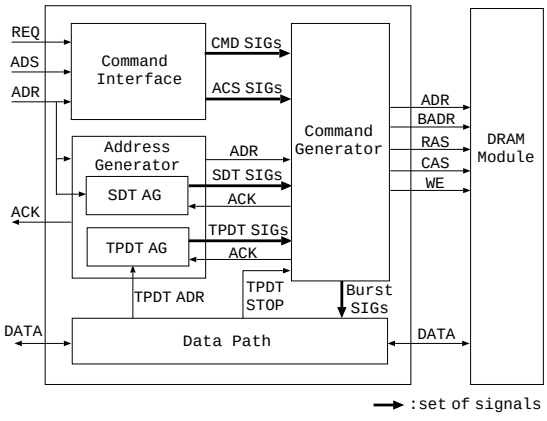

図 4 メモリコントローラのブロック図

4.2 メモリアドレス形式

SDT および TPDT のメモリアクセス要求を検出するために，MC は物理アドレスのフィールドを使用する．通常，メモリアドレスのフィールドは Byte offset を除いた下位ビットから Col，Row，Bank アドレスで構成され，残りの上位ビットはシステム依存である．ここでは，メモリアドレスの最上位 2 ビットをメモリアクセスモードの指定フィールド（AMODE）とする．AMODE の値はコンパイラあるいはリンカのアドレスリロケーションおよび，仮想記憶機構が協調することにより指定される．**図 5** にメモリアドレス形式，**表 1** に AMODE フィールド情報を示す．

4.3 プロセッサと MC の協調動作

4.3.1 SDT 方式の動作

プロセッサは SDT 開始前に SDT AG のレジスタにマップされたアドレスに書き込みを行うことにより，転送するデータ数およびデータ間隔値を格納する☆．プロセッサは AMODE フィールドを “01” とするアドレスによりメモリリクエストを発行する．MC はアドレスの AMODE フィールドをデコードし，開始アドレスとして SDT AG 内のレジスタに記憶し，DRAM アクセスを行いデータをプロセッサへ転送する．以下，データ間隔値を加算することにより次アドレスを生成し，Col アドレスの連続指定によって一定間隔毎のデータを読み出し，プロセッサへ転送する．プロセッサ内の再構成可能なキャッシュの一部を FIFO バッファとして利用し，転送されたデータが順次格納される[3]．データ転送の終了条件は，MC のレジスタにセットされた転送データ数に達した場合，間隔値の加算によって Bank，Row アドレスが変化した場合，あるいはページ境界のいずれかである．ここで，ページ境界での終了は仮想アドレスに空間に対する物理アドレスの連続性が保証されないためである．

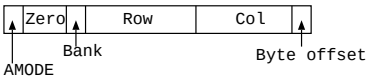

図 5 メモリアドレス形式

表 1 AMODE フィールド

アクセスの種類	AMODE
通常のメモリアクセス	00
SDT 方式のメモリアクセス	01
TPDT 方式のメモリアクセス	10
Reserved	11

☆ 更にデータサイズを格納することにより任意のデータサイズの SDT が可能となるが，本設計では簡単化のためこれを省略し，SDT のデータをポインタ（ワード）サイズに限定した．

SDT 転送中にキャッシュミスやライトバックによるメモリアクセスが発生した場合，SDT 転送を中断して応答する．その後プロセッサは FIFO 内のデータを読み出し，やがて FIFO が空になった時点で再リクエストを発行し SDT 転送が再開される．

4.3.2 TPDT 方式の動作

本 MC では，実体として文字列を扱う TPDT を設計した．プロセッサから AMODE が “10” のリクエストが発行された場合，MC はメモリ内にある実体へのポインタを読み出す．読み出されたポインタは Data Path を通じて TPDT AG 内のレジスタに記憶される．ポインタの値は仮想アドレスであるため，TPDT AG 内の MMU で物理アドレスに変換され，Command Generator を介して DRAM に送られ，文字列の実体が読み出される．実体は Data Path を通じてプロセッサに転送される．Data Path で NULL 文字を検出した場合，NULL 文字を転送後処理を終了する．また文字列の比較途中に条件が一致しないとわかった場合は次の文字列への TPDT リクエストが発行されるが，MC は前回の TPDT を停止して新たな TPDT を開始する．この際プロセッサ内の FIFO はクリアされ，TPDT の第一データの到着までは外部からのデータ挿入は禁止される．TPDT 転送中に他のメモリリクエストが発生した場合，あるいは MC 内の MMU でページフォルトが発生した場合に TPDT を一時中断する．その後の処理は SDT と同様である．

5. 性能評価

5.1 シミュレーション環境

C 言語で記述された質問処理をコンパイルして生成されたアセンブリコード（SPARC 命令セット）を入力とするシミュレータを作成した．1 命令実行を 1 プロセッサクロックサイクルとし，総実行サイクル数を求める．命令，データキャッシュはそれぞれ 8KB（2 ウェイセットアソシアティブ）である．キャッシュヒット時は 1 サイクルで読み書き可能であるが，ミス時のペナルティは 120 サイクルとした☆．転送方式に関する所要サイクル数として，実際に VHDL で設計した MC における動作サイクル数を使用した．これらを以下に示す．

- SDT 転送の前処理（転送数と間隔値の格納）に必要なサイクル数 ··· 各 40 プロセッササイクル
- SDT 転送の開始または再開時における初回データの読み出しサイクル数 ··· 120 プロセッササイクル
- TPDT 転送における初回の実体データの読み出しサイクル数 ··· 200 プロセッササイクル
- FIFO バッファ内データの読み出しサイクル数（ヒット時）··· 1 プロセッササイクル
- FIFO バッファ内データの読み出しサイクル数（ミス時）··· 1〜10 プロセッササイクル

TPDT 転送における 200 サイクルは最初の 4 文字を転送するまでのサイクル数である☆☆．FIFO ミス時のサイクル数（ストール時間）に関しては，データアクセス命令の実行タイミングと SDT の動作状況により値が決まり，最大でプロセッサとメモリバスの周波数比の 10 サイクルとなる．

5.2 リレーションと質問

評価対象のリレーションは Wisconsin Benchmark[2] の一部を MMDB 用に変更したものを使用する．一つのタプルは 15 個の属性を持ち，13 個の属性は整数型の実体を，残り 2 個の属性は文字列型の実体へのポインタを格納する．属性が持つ値の範囲は決まっており，例えば属性 “four” は 0，1，2，3 の値を持ち，リレーション内に均等な回数で出現する．すなわち「属性 “four”=1」で条件探索を行う場合，タプルの選択率は 25%となる．

このリレーションを用いて以下のような質問処理を実行する．括弧内は適用する転送方式を示す．

(a) 選択質問（SDT・TPDT）

（1）
```
SELECT * FROM R WHERE two = 1
```
（2）
```
SELECT * FROM R
    WHERE string1 ≦ 'xxxxx'
```
（1）は選択率 50%の属性 two でタプルを検索する．属性 two は直接タプルに属性値が格納されているため SDT を適用する．（2）は任意の文字列型データ ‘xxxxx’ でタプルを検索する．属性 string1 は文字列型の実体へのポインタが格納されてるため TPDT を適用する．

(b) 結合質問（SDT・TPDT）

（1）
```
SELECT * FROM R, S
    WHERE R.unique1 = S.unique1
```
（2）
```
SELECT * FROM R, S
    WHERE R.unique2 = S.unique2
    AND R.two = 1 AND S.two = 1
```
（1）はリレーション R, S の結合を属性 unique1 で行う．この属性は直接タプルに属性値が格納されているため SDT を適用する．（2）は R，S に対してそれぞれの選択処理で得たポインタテーブルによる結合処理のため TPDT を適用する．

(c) DISTNCT 質問（SDT）
```
SELECT DISTINCT four FROM R
```
この質問はリレーション R から属性 four が重複しているタプルを除去する．

(d) 集約質問（SDT）
```
SELECT four, COUNT(*) FROM R
```

☆ 1 メモリアクセス時間が 12 バスクロックサイクルであり，これを 10 倍の動作周波数を仮定したプロセッサのクロックサイクルで換算した．

☆☆ 本 MC は 32 ビットデータバスを使用しているため 4 文字（1 文字 8 ビット）を同時に転送する．

表 2 選択質問の実行サイクル数（SDT） -1-

タプル数	方式	属性 two	属性 four	属性 ten
100	normal	14,058	13,623	13,338
	sdt	3,368	2,636	2,273
1K	normal	145,195	136,256	130,955
	sdt	26,939	19,840	15,309
10K	normal	1,465,628	1,368,372	1,309,764
	sdt	337,036	218,622	146,662
100K	normal	14,671,896	13,688,616	13,096,228
	sdt	3,540,907	2,343,212	1,630,325

表 4 結合質問の実行サイクル数（SDT）

タプル数	アクセス方式	リードミスのサイクル数	総サイクル数	性能向上率（%）
R=1K S=1K	normal	1,201.2	1,274.3	-
	sdt（S に適用）	1.2	121.2	90.5
	sdt（R に適用）	1,200.0	1,273.4	0.1

（単位:10^5 サイクル）

```
GROUP BY four
```

属性 four の値ごとにその出現回数をカウントする COUNT 関数を用いた集約を実行する．

5.3 評価結果

表 2，**表 3** に従来方式と SDT 方式で（a）選択質問の（1）を実行したときのサイクル数を示す．表 2 は WHERE 句で指定する属性とリレーションのタプル数を変更したときの結果である．従来方式（normal）と比較し実行サイクル数の 76.0〜90.0%を削減することができた．従来方式では総サイクル数の 82〜93%がキャッシュミスに伴うメモリアクセスに費やされていた．これは 1 タプルのサイズがキャッシュのブロックサイズよりも大きいため空間的局所性が活かされず，タプル数分のメモリアクセスが発生していることに起因する．これに対し SDT 方式では従来方式と同数のメモリアクセスが発生するが，プロセッサの命令実行と並行してメモリから属性値を読み出し FIFO バッファに格納するため，見かけ上のメモリリクエスト回数およびメモリアクセス時間を削減している．

表 3 は，タプル数を 10K に固定して WHERE 句で指定する探索条件の属性数を増やしたときの実行結果である．1 つの属性に対してタプルを検索する場合に SDT 転送の効果は大きいが，2 つ以上の属性に対しては効果が小さくなる．これは SDT 転送のメモリアクセスは 1 つの属性に対してのみ適用可能であり，残りの属性に対しては通常のメモリアクセスを行う必要があるためである．しかし 4 つの属性検索の場合に 1 つの属性に対してのみ SDT 転送を適用することで実行時間の 22.7%を削減する結果が得られた．

表 4 は（b）結合質問の（1）に関する実行結果である．結合質問処理は 2 つのリレーションを同時に参照するため二重ループ構造になる．内側ループをリレーション S，外側ループをリレーション R とし，SDT

表 5 DISTNCT・集約質問の実行サイクル数（SDT）

タプル数	アクセス方式	DISTNCT 質問の総サイクル数	集約質問の総サイクル数
100	normal	16,529	15,522
	sdt	5,071	4,184
1K	normal	154,236	147,254
	sdt	37,465	30,363
10K	normal	1,531,484	1,486,127
	sdt	359,301	313,824

表 6 選択質問の実行サイクル数（TPDT） -1-

タプル数	文字列の種類	normal	tpdt	性能向上率（%）
1K	10	165,224	238,133	-44.1
	100	201,208	214,953	-6.8
	200	245,766	213,923	13.0
	500	252,858	213,105	15.7

表 7 選択質問の実行サイクル数（TPDT） -2-

タプル数	文字列の長さ	normal	tpdt	性能向上率（%）
1K	15	367,420	292,305	20.4
	30	532,168	372,289	30.0
	60	840,004	532,553	36.6

表 8 結合質問の実行サイクル数（TPDT）

タプル数	R の探索属性	S の探索属性	normal	tpdt	性能向上率（%）
R=10K S=1K	two	twenty	45.7	46.7	-2.3
		ten	135.1	132.8	1.7
		four	1261.7	1255.2	0.5
		two	3483.1	3479.6	0.1

（単位:10^5 サイクル）

をどちらか一方に対して適用する．従来方式では内側および外側のループでの属性読み出しが全てミスとなる．これは本来内側ループは参照回数が多いためキャッシュを有効利用できるが，ここではリレーション S のサイズはキャッシュ容量より大きく，キャッシュ内で追い出しが発生するためである．リレーション S に対して SDT 転送を実行した結果，外側での属性読み出しが全てミスになるが，内側での属性読み出しは FIFO バッファによって見かけ上のメモリアクセス時間を削減した。そのため実行時間の 90.5%を削減する結果が得られた．一方リレーション R に対して SDT 転送を適用した場合，内側に対して外側の参照回数の比率が小さいためその効果はほとんど得られなかった．

表 5 は（c）DISTNCT 質問，（d）集約質問の実行結果である．DISTNCT 質問はタプルの参照毎にその属性値がユニークであるかを判定するため，その時点での結果のリレーションを走査する必要があり，また集約質問ではタプルの参照毎に結果リレーションにアクセスし，その属性の出現回数をカウントする必要がある．従来方式では結果リレーションへのアクセスを

表 3　選択質問の実行サイクル数（SDT）-2-

タプル数	方 式	属性数:1	属性数:2	属性数:3	属性数:4
10K	normal	1,465,628	1,561,101	1,655,193	1,750,708
	sdt	337,036	1,159,992	1,255,497	1,353,489

含めたメモリアクセス時間はそれぞれ実行サイクル数の約 77〜82%，73〜78%を占めていた．SDT 方式ではそれらのメモリアクセス時間を削減し，(c) では約 73〜80%，(d) では 70〜77%の実行時間を削減した．

表 6，**表 7** は（a）選択質問の（2）に関して従来方式と TPDT 方式で実行した結果である．表 6 はタプル数を 1K に固定し文字列の種類を変更して TPDT 方式を適用したときの結果である．文字列の長さを 30 文字とした．従来方式では文字列の種類が少ない場合，ポインタと実体データがキャッシュで保持されるため文字列読み出しで発生するメモリアクセスの比率は 2.0%，33.0%（10，100 文字の順）であった．一方，文字列の種類が多い場合，それらはキャッシュから追い出され，さらにポインタ読み出しによる実体データの追い出しが発生するためその比率は 35.4%，47.5%（200，500 文字の順）であった．TPDT 方式では一度読み出された文字列はキャッシュで保持しないが，属性読み出しで発生するメモリアクセス率が高い場合に有効である．

表 7 は 500 種類の文字列で固定し文字列の長さを変更して実行した結果である．各文字列は最低 10，20，40 文字を比較するように設定した．従来方式においてメモリアクセスはポインタ読み出しのほかに文字列読み出しでも発生する．またキャッシュの 1 ブロックに収まらない文字列はその比較途中でメモリアクセスが必要になる．TPDT 方式では FIFO を使うことで文字列の長さに関係なく属性読み出しが可能である．

表 8 は（b）結合質問の（2）に関して従来方式と TPDT 方式で実行した結果である．内側ループをポインタテーブル S，外側ループをポインタテーブル R とした．両方式においてポインタテーブル S へのアクセスはほぼ同じキャッシュヒット率を示していたため TPDT 転送はリレーション R のポインタテーブルに適用した．従来方式によるポインタテーブルへのアクセスはキャッシュでヒットするが，TPDT 方式ではアクセス毎にメモリアクセスが発生するため TPDT による性能向上が見られなかった．

以上の結果から，これらの質問処理で SDT および TPDT 方式を適用した場合の質問処理時間の高速化を確認した．

6. 関 連 研 究

従来の MC に新たな機能を付け加えることで高速データ転送を実現する方式として Impulse[4] と SMC[5] がある．Impulse は空間的局所性のないデータをプログラム中に用意したエイリアスを用いて必要なデータのみを転送する．これはプログラマへの負担が大きく，データアクセス毎の多段のアドレス変換によるオーバヘッドが大きい．SMC は DRAM の同一 Row 内のデータを Col アドレスの連続指定して MC 内のバッファに格納する．しかしデータアクセス毎に発生するプロセッサと MC 間の転送がボトルネックになっている．また，これらの転送方式は時間的局所性の保証されないデータがキャッシュに格納されることになる．本研究では FIFO バッファを使うことでその問題を解消し，さらに必要なデータのみをプリフェッチするためメモリアクセスのオーバヘッドを削減している．

7. おわりに

本稿では主記憶データベースにおけるデータ参照時間削減手法として，メモリ空間に一定間隔で存在するデータをメモリからプロセッサに連続して転送する SDT 方式と，ポインタを介した二段階のアクセスに対してプロセッサが見かけ上一回のメモリアクセスでデータを取得する TPDT 方式を提案した．これらの方式と従来方式に関してシミュレーションにより質問処理時間を評価した．今後は複雑な質問処理文における提案した転送方式の評価を行い，更に MC にデータベース演算機構を追加することでプロセッサの演算負荷を分散し，スループットの向上を図る．

謝辞

本研究において，Synopsys 社と Model Technology 社の University Program を用いた．深く感謝します．

参 考 文 献

1) H. Garcia-Molina, K. Salem, “Main Memory Database Systems: An Overview.” IEEE Trans. on Knowledge and Data Engineering, Vol.4, No.6, pp.509–516, 1992.
2) DeWitt. D. J, “The Wisconsin Benchmark:Past, Present, and Future.” The Benchmark Handbook, pp.269–316, J.Gray ed., Morgan Kaufmann, 1993.
3) Khairuddin bin Khalid and Kiyofumi Tanaka, “Implementation of FIFO Buffer Using Cache Memory.” IPSJ SIG Notes, ARC, Vol.2002, No.112, pp.83–88, 2002.
4) J. Carter, W. Hsieh et al. “Impulse: Building a smarter memory controller.” In Proc. of the 5th HPCA, pp. 70–79, 1999.
5) S. McKee et al. “Design and evaluation of dynamic access ordering hardware.” In Proc. of the 10th ICS, pp.125–132, 1996.