計算機アーキテクチャ 150－15
（２００２．１１．２８）

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

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の拡大および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，質問処理の高速化が重要となっている．本稿では主記憶データベースにおけるデータ構造と DRAM のハードウェア特性を利用したデータ転送方式を提案し，シミュレーションによりそれらの評価を行う．

Highly Functional Memory Controller for Main Memory DataBase

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

The response time in database systems is getting large because of the growing gap between speed of a CPU and that of a memory, and the increase in data size. it is thus important to accelerate query processing. In the paper, we propose the data transfer methods which take advantage of the data structure in main memory database systems and the characteristic of DRAM, and evaluate them in simulations.

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，高いリアルタイム性が要求されるデータベースシステムに使用される．

データベースシステムを高速化するためには質問処理時間，障害時のログファイルによるデータ回復時間，トランザクションの安全性を確保するために定期的にとるバックアップ時間などの改善が必要であるが，本研究では質問処理時間の短縮化について取り組む．

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に対して特定の属性を走査する場合，メモリ内では空間的局所性，および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，データの実体だけでなく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と比較し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，一定時間内に大量のデータを処理することができるため高速リアルタイム処理に適している．☆

† 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

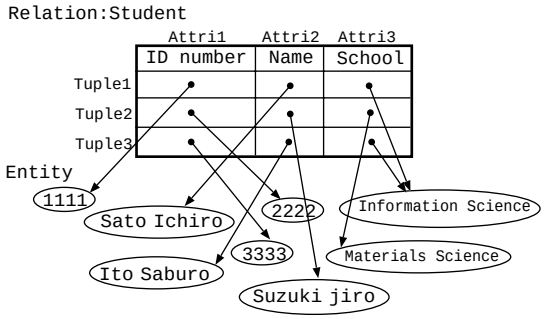

図 1 MMDB 方式のリレーション構成

図 1 に MMDB 方式のリレーション構成を示す．リレーション内の各セルには実体へのアドレスが格納され，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．なお，メモリ内で実体はリレーションとは別の領域に格納される．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，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データをアクセスする際，従来の MC では各データに対して 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，一つのタプルのサイズが一定間隔に相当し，ある属性に対して複数のタプルを条件探索する際に一定間隔毎のメモリアクセスが発生する．このとき，MC は最初のタプルの属性データを 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 が一定間隔値を加算して Col アドレスを自動生成することにより，同一 Bank，Row アドレス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を図るために実体へのポインタを介した二段階のメモリアクセスを実行する．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において，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）方式はこの二段階のメモリ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，あるいはポインタ比較による条件探索で得られた結果を出力する場合などで使用される．

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

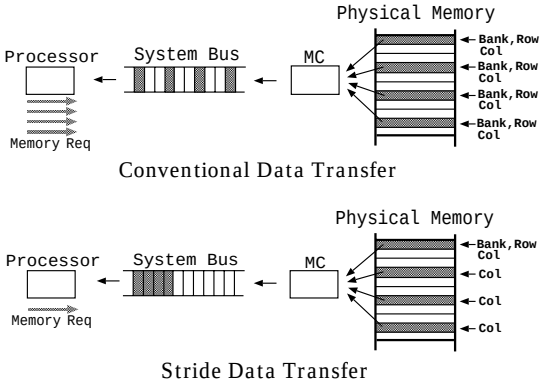

図 2 SDT 方式によるデータ転送

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．

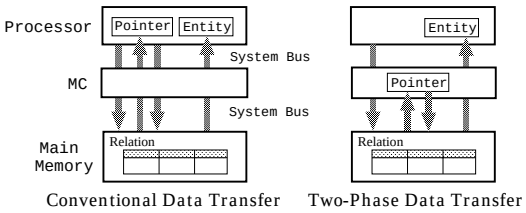

図 3 TPDT 方式によるデータ転送

されないためキャッシュの使用効率が向上する．

4. メモリコントローラの実装

4.1 概　　要

本稿で提案したデータ転送方式を実現する MC を設計した．ブロック図を**図 4** に示す． 図において MC は 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）．

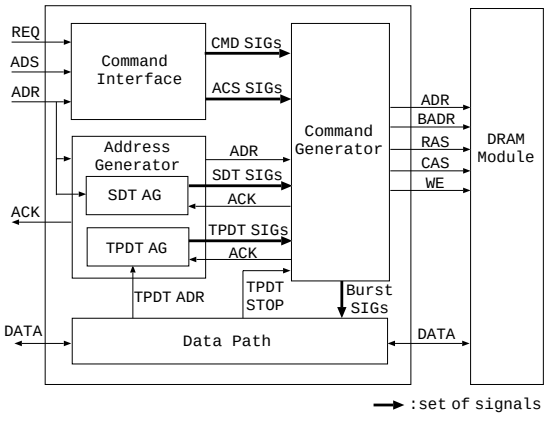

図 4 メモリコントローラのブロック図

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る．すなわち，Address Generator モジュールの追加によって SDT および TPDT が実現される．

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 フィールド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（アドレスの変換）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

SDT および TPDT 方式におけるプロセッサと MC の動作を述べる．

4.3.1 SDT 方式の動作

プロセッサは 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 は受け取ったアドレスの AMODE フィールドをデコードし，SDT の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，記憶された開始アドレスに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]．

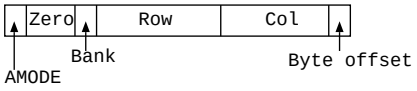

図 5 メモリアドレス形式

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 でアクセスするデータをポインタ（ワード）サイズに限定する．

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．

SDT 転送中にキャッシュミスやライトバックのメモリアクセスが発生した場合，SDT 転送を停止して応答する．プロセッサ内の FIFO が空になり，再リクエストが発行されて SDT 転送が再開する．

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” のアドレスでメモリリクエストが発行された場合，MC はメモリ内にあるリレーションから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，文字列型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．

5. 性能評価

本節では質問処理における SDT 方式の効果をシミュレーションにより示す．

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 クロックサイクルとし，総実行サイクル数を求める．命令，データキャッシュそれぞれのサイズは 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☆．SDT 処理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．

- 前処理（転送数と間隔値の格納）に必要なサイクル数 ··· 各 40 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 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 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 サイクル

☆ 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．

最後の 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を**表 2** に示す．これは Wisconsin Benchmark[2] の一部であり，タプルの 15 個の属性のうち 13 個（unique1 から oddOnePercent）が整数型の実体，2 個（string1 と string2）が文字列型の実体へのポインタである．各属性の情報として左から属性名，値の範囲，順番，選択率を表す．順番とは各属性の値（例えば属性 “four” において 0，1，2，3 の値）のそれぞれが出現する順番であり，random はランダム，sequential は昇順，cyclic は文字列が周期的に繰り返されることを意味する．選択率とは各属性の値が出現する割合である．すなわち後の評価において，全タプル数に対して質問処理で選択抽出されるタプル数の割合を表す．なお，MAXTUPLS は全タプル数を示す．

このリレーションを用いて以下のような選択，結合，射影，集約の 4 つの単純な質問を実行する．

(a) 選択質問

```
SELECT * FROM R WHERE two = 1
```

この例では選択率 50%の属性 two でタプルを検索する．10 万のタプル数で実行した場合は 5 万のタプルが選択される．

(b) 結合質問

```
SELECT * FROM R, S
WHERE R.unique1 = S.unique1
```

リレーション R，S の結合を行う．リレーション S のタプル数を R の 1/10 とした場合，結合属性 unique1 はユニークであるため結果のタプル数は S と同じ数になる．

(c) 射影質問

```
SELECT DISTINCT four FROM R
```

この射影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

```
SELECT four, COUNT(*) FROM R
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．

上記のうち選択質問を実行するコードを例に挙げる．

従来型 MC の質問処理コード

```
for( i = 0; i < MAXTUPLES; i++ ){
  if(relationR[i].two == 1){
    rslt_ptr[j++] = &relationR[i];
  }
}
```

表 2 リレーションの仕様

Attribute Name	Range of Values	Order	Selectivity
unique1	0-(MAXTUPLES-1)	random	unique
unique2	0-(MAXTUPLES-1)	sequential	unique
two	0-1	random	50%
four	0-3	random	25%
ten	0-9	random	10%
twenty	0-19	random	5%
onePercent	0-99	random	1%
tenPercent	0-9	random	10%
twentyPercent	0-4	random	20%
fiftyPercent	0-1	random	50%
unique3	0-(MAXTUPLES-1)	sequential	unique
evenOnePercent	0,2,4,...,198	random	1%
oddOnePercent	1,3,5,...,199	random	1%
string1	-	cyclic	20%
string2	-	cyclic	20%

属性をメンバとする構造体でタプルを表し，その構造体の配列型で定義されたリレーション（relationR）を全タプル数（MAXTUPLES）回繰り返すループで走査する．属性 two が条件を満たすタプルの先頭アドレスを結果の配列（rslt_ptr）に格納する．

SDT 方式の質問処理コード

```
SET_TIMES(100);  /* 転送数をレジスタに格納 */
SET_STRIDE(60);  /* 間隔値をレジスタに格納 */
for( i = 0; i < MAXTUPLES; i++ ){
  if(*(&relationR[i].two|0x40000000) == 1){
    rslt_ptr[j++] = &relationR[i];
  }
}
```

SDT 方式で実行する場合，前処理として MC の内部レジスタにデータ転送数および間隔値（タプルサイズの 60 バイト）を格納する．SDT 転送を属性 two への参照で使用するために，上位 2 ビットに “01” を加えた仮想アドレス（ここでは 32 ビット）を指定する．

5.3 評価結果

表 3，**表 4** に従来 MC 方式と SDT 方式で選択質問を実行したときのサイクル数を示す．表 3 は（a）選択質問の 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て実行したときの総サイクル数を示している．従来方式と比べて約 4.1〜8.8 倍の性能向上が得られている．

従来方式では総サイクル数の 90〜93%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読み出された属性値が FIFO バッファに格納されるため，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．またキャッシュミスで発生するメモリアクセスに伴う SDT 転送の中断から再開までの時間は，全実行時間の約 30%を占めていた．

表 3 選択質問の実行サイクル数 -1-

タプル数	方式	属性 two	属性 four	属性 ten
100	normal	14101	13644	13346
	sdt	3400	2642	2274
1K	normal	137103	132292	129553
	sdt	27198	19890	15323
10K	normal	1367720	1321782	1293886
	sdt	265948	191554	146742
100K	normal	13669824	13209135	12929342
	sdt	2663753	1899912	1461912

表 4 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 1.3 倍以上の性能向上が得られた．

表 5 は（b）結合質問の実行結果である．それぞれの探索条件の属性 unique1 はユニークであるため結果としてタプル数が 100 のリレーションを得る．表中のリードミスのサイクル数は，データキャッシュアクセ

表 4　選択質問の実行サイクル数 -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	1367720	1468762	1567542	1669179
	sdt	265948	1054132	1153984	1255263

表 5　結合質問の実行サイクル数

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
R=1K S=100	normal	1298040(62%)	2102074
	sdt(S に適用)	120000 (8%)	1496103
	sdt(R に適用)	947160(53%)	1779760

スのミスであり，括弧内は総サイクル数に対する比率を示す．

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．内側ループで検索されるリレーションは外側ループのリレーションより参照回数が多くキャッシュの効果を得やすいため，従来方式ではタプル数の少ないリレーション S を内側ループに適用した．SDT 方式では同様にリレーション S を内側ループとして，リレーション R，S のどちらか一方に対して SDT 転送を適用した．総サイクル数を比較するとリレーション S に対して SDT 転送を実行した結果，従来方式に対して約 1.4 倍，リレーション R に SDT 転送を適用した場合に対して約 1.2 倍の性能向上となった．データキャッシュに対するリードミスのサイクル数の割合は小さいが，SDT 転送の中断によって発生する，再開までのストールサイクルによって性能向上が抑えられている．

表 6，**表 7** はそれぞれ（c）射影質問,（d）集約質問の実行結果である．

射影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る．従来方式では結果リレーションへのアクセスを含めたメモリアクセス時間は総サイクル数の約 76〜78%を占めていた．同様に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り，メモリアクセス時間の割合は総サイクル数の約 70〜74%であった．二つの質問で SDT 転送を適用した場合，約 2.1〜3.0 倍の性能向上が得られた．

以上の結果から，これらの単純な質問処理に SDT 転送方式を適用した場合，プロセッサの命令実行と並行して，参照する属性をメモリから連続して転送することによりメモリアクセス時間を削減し，質問処理時間が高速化されることを確認した．

表 6　射影質問の実行サイクル数

タプル数	アクセス方式	総サイクル数
100	normal	17583
	sdt	6245
1K	normal	170455
	sdt	52741
10K	normal	1720355
	sdt	547934

表 7　集約質問の実行サイクル数

タプル数	アクセス方式	総サイクル数
100	normal	20636
	sdt	9059
1K	normal	197320
	sdt	79358
10K	normal	1954698
	sdt	782039

6. 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．

提案した SDT 方式と従来方式をシミュレーションにより比較し，質問処理を実行した場合の総サイクル数に関して最大 8.1 倍の性能向上が得られた．これは DRAM 特性を利用した制御信号の再入力の除去およびプロセッサと MC 間のメモリリクエストの一括化の効果である．

今後は SDT 方式の複雑な質問文に対する評価および，TPDT 方式の評価を行う．これらの機構を持つ MC に，更にデータベース演算機構を追加することでプロセッサと MC で演算の負荷を分散し，質問処理の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.” Design Gaia 2002, ARC, 2002.