計算機アーキテクチャ 85－7
（1990．11．21）

データパラレルハッシュジョインアルゴリズムとコネクションマシンによるその評価

松本 和彦　喜連川 優　高木 幹男
東京大学 生産技術研究所

データベース処理に並列コンピュータを利用する試みは、今までに数多くなされており、ハッシュに基づくアルゴリズムなどは特に並列処理に向いていることがわかっている。一方、並列コンピュータアーキテクチャの中でも、データパラレルマシンと呼ばれるＳＩＭＤ型のものは、画像処理や流体解析、熱伝導解析などに適することが知られており、既にこのような目的のためにコネクションマシンやＭＰＰ、ＭａｓＰａｒなどのマシンが使われてきた。しかし、このようなデータパラレルマシンをデータベース処理に利用しようとする研究は今のところほとんど報告されていない。本論文ではデータベース処理をコネクションマシンにインプリメントし、その性能評価を行なった結果を報告する。

Data Parallel Hash Join Algorithm and Its Evaluation on the Connection Machine

Kazuhiko Matsumoto　Masaru Kitsuregawa　Mikio Takagi
Univ. of Tokyo Institute of Industrial Science

Many attempts have been made to utilize parallel machines for the database processing. Especially it's well known that hash-based algorithms are appropriate for the parallel processing. On the other hand, the SIMD type machines called Data Parallel Machines are often used for the such applications as image-processing and flow-analisys. But no or little reserches which utilize these machins for the relational database processing. In this paper, we implement the database algorithm called Data Parallel Hash Join Algorithm and evaluate its performance.

1 はじめに

これまで、関係データベースシステムの性能を改善するための研究が、数多くなされてきた[2, 3]。その一つのアプローチは、高速なソートエンジンをハードウェアでインプリメントする方法である。なぜなら、ソーティングは最も基本的な処理の一つであり、キー属性に基づくソート時間を除けば、ほとんどの関係データベース演算は線形時間で処理することが可能だからである。

その他のアプローチとしては、ハッシュアルゴリズムを利用するものがある。1983年以降、GRACEハッシュ[4]、ハイブリッドハッシュ[5]のようなハッシュに基づいたいくつかのアルゴリズムが提案されてきた。ハッシュ関数によって分割されたバケットは互いに依存関係がなく複数のプロセッサによって効率的に処理できるため、これらのアルゴリズムは、並列処理に適している。

コンピュータアーキテクチャから見た場合、並列マシンは大きく二種類に分類することができる。NCUBE、BNN butterfly、SequentなどのMIMDマシンと、コネクションマシン、MPP、MasParなどのSIMDマシンである。MIMDマシンについては、これまでにもGAMMA[6, 7]、FDS[8, 9]、Teradata[10]などの多くの研究がなされている。しかし一方で、コネクションマシンなどのSIMDマシンによる研究は、今のところほとんど報告されていない。

現在、CM-2と呼ばれるコネクションマシンの最新モデルはWeiteckの浮動小数点アクセラレータを備え、主に膨大な数値計算を行なうアプリケーションのために利用されている。本論文では、コネクションマシンCM-2上で関係データベース演算の性能評価を行ない、SIMDマシンのデータベース処理に対する有効性について考察する。

2 データパラレルハッシュジョインアルゴリズム

2.1 関係データベースにおけるジョイン処理

関係データベースでは、データがタプルと呼ばれる単位で管理される。1つのタプルはいくつかのデータフィールドからなるが、ある処理に着目した場合には、1つのデータフィールドのみが検索や比較の対象となることが多い。このような処理の主役となるようなフィールドをキー属性、その他の全てのデータフィールドをまとめてタプルの持つデータと呼ぶことにする。

ある情報を表現するためのタプルの集合体を、リレーションと呼ぶ。関係データベース演算の一つであるジョイン処理とは、2つのリレーションを元にして新しい1つのリレーションを生成するものである。

処理の内容は、2つのリレーション間でキー属性の同じタプルどうしが直積を作るというもので、具体的には図1のように表現される。

2.2 データパラレルハッシュジョインアルゴリズム

2.2.1 概要

データパラレルマシンの上でジョイン処理を行なうには、従来の逐次型マシンの場合とは異なったアルゴリズムが要求される。なぜなら、データパラレルマシンでは数万個にも及ぶデータアイテムを並列に処理することが可能なので、逐次型マシンのための最適化はそのままでは全く有効でないからである。

本論文で採用したアルゴリズムは、ハッシュに基づいたものであり、データパラレルハッシュジョインアルゴリズム（DPHJアルゴリズム）と呼ぶ。まず、二つのリレーションのタプルをそのキー属性に従ってクラスタに分割し、それぞれのクラスタでローカルにジョイン処理を行なう。そして最後に生成したタプルをプロセッサに分配して求めるリレーションを形成する。アルゴリズムは大きく三つのステップに分けられ、それぞれクラスタリングフェイズ、ネストループフェイズ、リレーション生成フェイズと呼ぶ。詳細については、以下に述べる。

2.2.2 クラスタリングフェイズ

最初に、タプルは複数のクラスタに分割される。クラスタリングは各々のタプルのキー属性をハッシュにかけた値に従ってなされ、同一のハッシュ値を持つタプルは、同一のクラスタに含まれることになる。

アルゴリズムの正確な記述は次のようになる。なお、本文中で*parallel-variable*のようにイタリックで表記される英単語は、各々のプロセッサに1つずつ、全体としてはプロセッサの数だけの値が保持される「並列型変数」を意味している。

i) クラスタリングのための空間Hの中に、各々のハッシュ値に対して二つずつのキューが用意される。一方はリレーションRに対応し、*R-queue*と呼ばれ、もう一方はリレーションSに対応し*S-queue*と呼ばれる。キューは受けとったメッセージを到着した順に積み上げていくので、複数のメッセージが同じキューに届けられたとしても、メッセージが失われることはない。また、キューは届いたメッセージの数を記憶するためのカウンタも備えている。これらは、それぞれ*R-counter*、*S-counter*と呼ばれる。

ii) Rの全てのタプルは自分のキー属性を入力としてハッシュ値を計算する。そしてハッシュ値をもとに、Hの中の対応するキューに自分のキー属性と自分のアドレ

スからなるメッセージを送信する。同様の処理をSに対しても行なう。

こうして、図1に示すように、2つのキューと2つのカウンタがセットアップされ、RとSのタプルのクラスタ分割が終了する。

2.2.3 ネストループフェイズ

二番目のフェイズは、それぞれのクラスタ、即ちそれぞれのPE内でのローカルなジョイン処理である。ここでは、クラスタリングによって、1つのPEの担当するタプルの数は十分に小さいことが期待できるので、最も簡単なジョインアルゴリズムであるネストループアルゴリズムを採用する。

i) 通常の変数（つまりスカラ変数）であるIRとISを用意して、IRを0で初期化する。また、配列 *msg-array* とそのカウンタ *joined-msg-counter* を用意し、*joined-msg-counter* を0で初期化する。

ii) IRが *R-counter* と比較され、*R-counter* の値がIRより大きなクラスタだけがアクティブにされる。以降の処理はアクティブなクラスタのみが実行する。この時点でもしHにアクティブなクラスタが無くなったら、ネストループフェイズは終了する。

iii) ISを0で初期化する。

iv) *S-counter* の値がISより小さなクラスタは、インアクティブにされる。この時点でアクティブなクラスタがHに無くなったら、IRをインクリメントしてからii)からの処理を繰り返す。

v) *R-queue* のIR番目のメッセージと、*S-queue* のIS番目のメッセージが比較される。2つのメッセージのキー属性が一致したクラスタでは、次の処理を行なう。

2つのメッセージから新しいメッセージが作って *joined-msg-array* に積み、*joined-msg-counter* をインクリメントする（図2参照）。新しいメッセージはキー属性と2つのアドレスからなる。

vi) ISがインクリメントされ、iv)から処理が繰り返される。

2.2.4 リレーション生成フェイズ

この時点では、それぞれのクラスタの持つ（つまりそれぞれのPEの持つ）メッセージの数は異なっているので、1つのPEが1つのメッセージを持っているような状態にしなければならない。また、メッセージはデータのある場所のポインタを持っているだけなので、データの本体を持って来なければならない。このような処理によって、リレーション生成フェイズでは、最終的なリレーションJを生成する。

i) *joined-msg-counter* の値の総和が計算される。この数字に基づいて、生成されるリレーションのための空間Jがアロケートされる。

ii) *joined-msg-counter* の累積和が計算され、結果が並列型変数 *evenup-adrs* にストアされる（図3参照）。

iii) スカラ変数Iを用意し、0で初期化する。

iv) *joined-msg-counter* がIより大きなクラスタだけがアクティブにされる。この時点でアクティブなクラスタが無くなったら、vii)からの処理を行なう。

v) アクティブなクラスタは *joined-msg-array* のI番目のメッセージを *evenup-adrs* に従ってJに送信する（図4参照）。

vi) アクティブなクラスタは *evenup-adrs* をインクリメントする。Iをインクリメントしてiv)からの処理を繰り返す。

vii) Jのそれぞれの PE は送信されたメッセージの対応するアドレスに従い、タプルのデータをSから受信する（図5参照）。

viii) Jのそれぞれの PE は送信されたメッセージの対応するアドレスに従い、タプルのデータをRから受信する（図6参照）。

このようにして、最終的に望むリレーションを、Jの中に作ることができる。

3 コネクションマシンによるデータパラレルハッシュジョインアルゴリズムの実装

3.1 Paris ライブラリ

PARIS とは、コネクションマシンシステムをプログラミングするための、PARallel Instruction Set のことである。それは、フロントエンドからコネクションマシンプロセッサを指令するための低レベルインターフェースである。PARIS の実体は、関数、マクロ、変数の集合

である。関数やマクロは、呼び出されることにより、必要に応じてコネクションマシンのシーケンサにマクロ命令を送出する。各種の変数は、ユーザにコネクションマシンシステムに関する情報を与える。

PARISは、抽象化されたコネクションマシンハードウェアをユーザに見せることによって、コネクションマシンのスケーラビリティを拡張している。その核心となるのは、各々の物理プロセッサが複数の仮想的なプロセッサをシミュレートするという、仮想プロセッサの機能である。プログラマは好きなだけの仮想プロセッサ数を仮定してプログラムすれば良く、プログラムのできた後で、仮想プロセッサは物理プロセッサに割り当てられる。こうすることによって、ソフトウェアが物理的なハードウェアによって変更を強いられることは無くなる。そこに残るのは、単に、物理プロセッサ数と実行時間やメモリ容量との間のトレードオフである。

PARISは基本的に、より高レベルの言語を開発するための基礎となるように意図されている。それは、従来の計算機のマシンコードに相当するレベルの多くのオペレーションを用意している。具体的には、符号付き整数、符号無し整数、浮動小数点数、複素数の各々に対する基本操作と、プロセッサ間通信、コネクションマシンプロセッサとフロントエンド間のデータ転送をサポートしている。

以下に、PARISにおける基本概念の代表的なものについて説明をする。

3.1.1 仮想プロセッサ

PARISは、抽象化されたコネクションマシンハードウェアをユーザに見せることによって、スケーラビリティを拡張している。その核心となるのは、各々の物理プロセッサが複数の仮想的なプロセッサをシミュレートするという、仮想プロセッサの機能である。

プログラマは好きなだけの仮想プロセッサ数を仮定してプログラムすれば良く、プログラムのできた後で、仮想プロセッサは物理プロセッサに割り当てられる。こうすることによって、ソフトウェアが物理的なハードウェアによって変更を強いられることは無くなる。そこに残るのは、単に、物理プロセッサ数と実行時間やメモリ容量との間のトレードオフである。また、1つの物理プロセッサがいくつの仮想プロセッサをシミュレートするかという数のことをVP比と呼ぶ。

3.1.2 VPセット

あるデータ集合が割り当てられている仮想プロセッサの集合のことを、仮想プロセッサセット（VPセット）と呼ぶ。仕事によっては、複数のデータ集合を扱うことが必要となるので、PARISは複数のVPセットが同時に存在することを許している。例えば、ドキュメント検索のプログラムでは、ある時点では記事（数キロバイト）をデータ要素として扱い、またある時点では記事中の単語（数十バイト）をデータ要素として扱う。このプログラムでは、それぞれに対応した2つのVPセットを使うのが最も適切であろう。そして、単語をデータ要素とするVPセットは、記事をデータ要素とするVPセットよりはるかに大きくなるだろう。VPセットはPARISへの関数呼び出しによって生成される。その大きさ（VPの数）は生成された時点で決まり、以後変化しない。VPの数は物理プロセッサの倍数でなければならない。

3.1.3 フィールド

メモリは、フィールドという単位で扱われる。フィールドとは、メモリ中に割り当てられた連続するビット群のことである。フィールドの長さは任意で、割り当てる時にユーザが指定する。フィールドは、必ずどれか一つのVPセットに所属する。

3.1.4 フラグ

PARISの仮想プロセッサは、いくつかの1ビットフラグを持っている。多くのPARISオペレーションは、実行結果をメモリ以外にフラグにも書き込む。

コンテクストフラグは特に重要である。ほとんどのPARISインストラクションは条件付きであり、コンテクストフラグが立っている仮想プロセッサでのみ実行される。コンテクストフラグ自身を変更する命令を含む幾つかのインストラクションについては、無条件に実行がなされる。

3.2 DPHJアルゴリズムのParisライブラリによる展開

基本的に、リレーションR、S、M、Jはそれぞれ VPセットとして、並列型変数はフィールドとして表現される。VPセットやフィールドは実行時に動的に生成、解放することができる。

並列型変数に対する加算や乗算といったオペレーションは、コネクションマシンの全てのPEによってローカルに実行される。一方、全てのPEの持つ値の総和を求める等のオペレーションは全てのPEの協調の下にマシン全体で行なわれる。

4 DPHJアルゴリズムの性能評価

4.1 性能測定環境

- 使用したハードウェアは、クロック 6.7MHz プロセッサ数 8192 個のコネクションマシン CM-2。
- タプルのサイズは、キー属性 2 バイト、データ 13 バイトの計 15 バイト長。
- ジョイン前のリレーションのプロセッサへのロード時間は測定時間に含まれない。
- 選択率は 100% と 200% について、VP 比は、1、2、4、8 と変化させて測定を行なった。
- 選択率 100% のジョインでは、元となるリレーションでのキー属性値はユニークであり、その配列はランダムである。また、選択率 200% のジョインでは、元となるリレーション内で、あるキー属性について 2 つのタプルが存在するようになっており、その配列はランダムである。
- 測定結果に示される「タプル数」とは、各々のリレーションの持つタプルの数である。つまり、タプル数が 1024 ということは、1024 タプル × 1024 タプルのジョインで 100% なら 1024 タプル、200% なら 2048 タプルが生成されることを意味する。

4.2 性能測定結果

測定結果を図 7、8 に示す。これらのグラフから、逐次マシン上の最も簡単なジョインアルゴリズムがタプル数の二乗に比例する処理時間を必要とするのに対し、コネクションマシン上における DPHJ ジョインアルゴリズムは、タプル数に対して定数に近い時間しか必要としないことがわかる。これは、一般的にデータパラレルマシンにおいては全てのデータ要素に対して同時にオペレーションを行なうことができるため、処理に必要な時間はデータ数に依存しないからである。

また、図 7 と図 8 の比較から、選択率が 100% から 200% になると、処理時間が二倍程度に増大することがわかる。

5 おわりに

データパラレルソートマージジョインのインプリメントをコネクションマシン上にすることによって、データパラレルマシンの関係データベース処理への有効性を確認した。データベース処理は数多くのデータ要素を扱うアプリケーションであるため、画像処理や流体解析などと同様に、このようなマシンによる大幅な高速化が可能であると思われる。

謝辞

我々の研究に多くの協力をしていただいた Thinking Machines Corporation、TEPCO に感謝の意を表する。

参考文献

[1] D. ヒリス著 喜連川訳: コネクションマシン、パーソナルメディア、 1990.

[2] E. Ozkarahan: Database Machines and Database Management, Prentice-Hall, N.J., U.S.A., 1986.

[3] S. Y. W. Su: Database Computers, McGrow-Hill, N.Y., U.S.A., 1988.

[4] M. Kitsuregawa, H. Tanaka, and T. Moto-oka: Application of Hash to Data Base Machine and Its Architecture, New Generation Computing, Vol.1, No.1, pp.66-74, 1983.

[5] D. J. DeWitt, R. H. Katz, F. Olken, L. D. Shapiro, M. R. Stonebraker, and D. Wood: implementation Techniques for Main Memory Database Systems, in ACM SIGMOD '84, pp.1-8, 1984.

[6] D. J. DeWitt, *et al*: GAMMA - A High Performance Dataflow Database Machine, Proc. of Int. Conf. on Very Large Data Bases, pp.228-237, 1986.

[7] D. J. DeWitt, *et al*: A Performance Evaluation of Four Parallel join Algorithms in a Shared-Nothing Multiprocessor Environment, Proc. of SIGMOD, 1989.

[8] M. Kitsuregawa, *et al*: Functional Disk System for Relational Database, Proc. of the 3rd Int. Conf. on Data Engineering, pp.88-95, 1987.

[9] M. Kitsuregawa, *et al*: Query Execution for Large Relations on Functional Disk System, Proc. of Int. Conf. on Data Engineering, pp.159-167, 1989.

[10] Teradata Corp.: DBC/1012 Data Base Computer Concepts and Facilities, CO2-0001-05.

[11] Thinking Machines Corp.: Connection Machine Model CM-2 Technical Summary, Version 5.1, 1989.

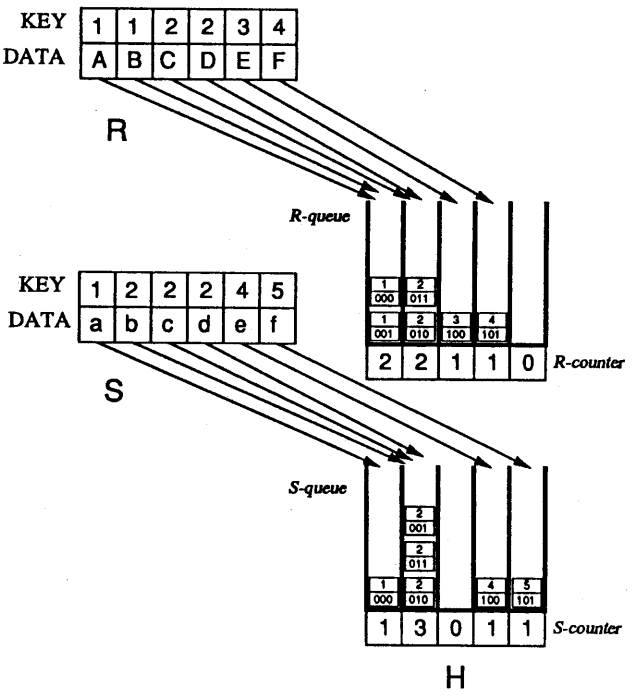
KEY
DATA
R
S
R-queue
S-queue
R-counter
S-counter
H

図1

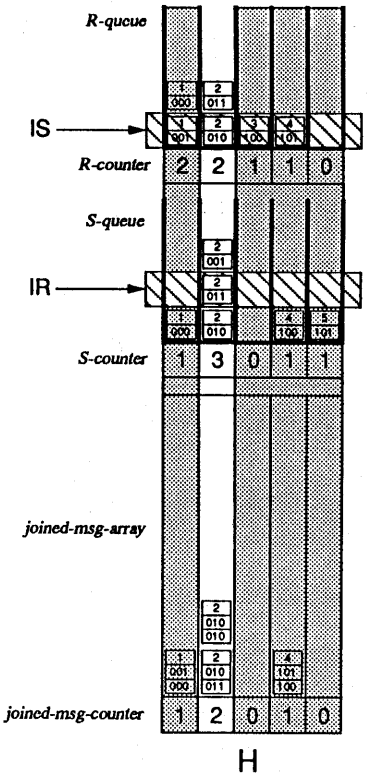
R-queue
IS
R-counter
S-queue
IR
S-counter
joined-msg-array
joined-msg-counter
H

図2

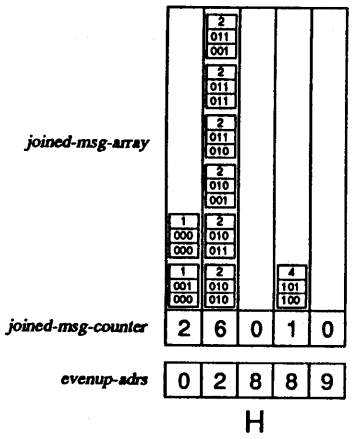
joined-msg-array
joined-msg-counter
evenup-adrs
H

図3

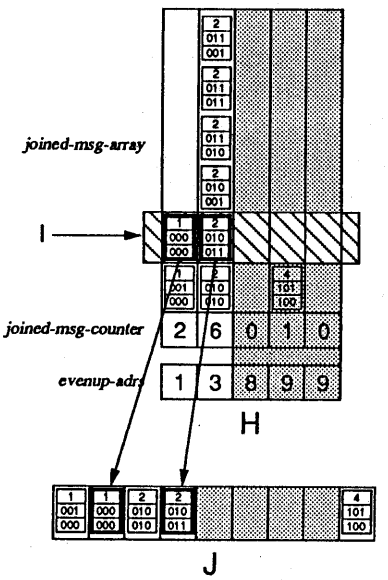
joined-msg-array
I
joined-msg-counter
evenup-adrs
H
J

図4

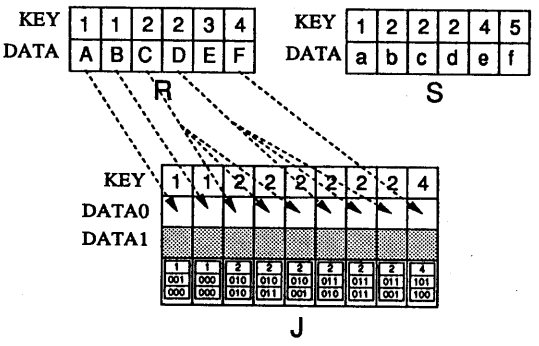
KEY
DATA
R
S
J
DATA0
DATA1

図5

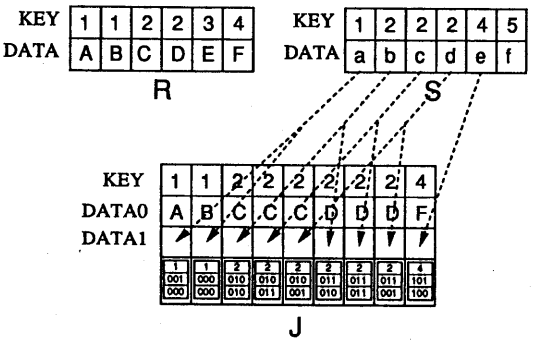
KEY
DATA
R
S
J
DATA0
DATA1

図6

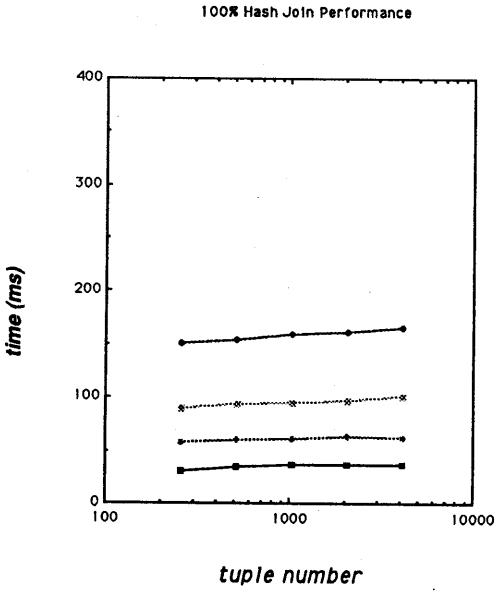
100% Hash Join Performance
time (ms)
tuple number

図7

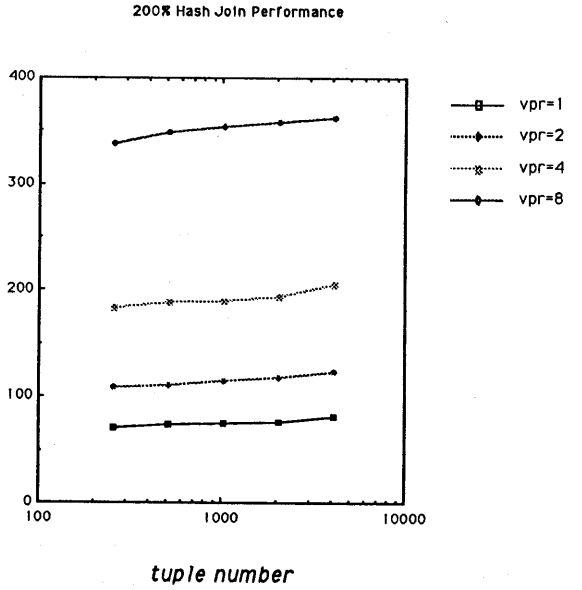
200% Hash Join Performance
vpr=1
vpr=2
vpr=4
vpr=8
tuple number

図8