データベースシステム 94−29
（1993. 7. 23）

コネクションマシンCM-2による大規模関係データベース処理とその評価

岡田 英明* 松本 和彦 喜連川 優
東京大学 生産技術研究所
{okada,matsu,kitsure}@tkl.iis.u-tokyo.ac.jp

概要

無共有型並列計算機による関係データベース処理性能向上への試みがこれまでに数多くなされている。一方、コネクションマシンをはじめとするデータパラレルマシンと呼ばれるSIMD型の並列計算機を関係データベース処理に利用するという研究は、今のところほとんど報告されていない。これまでに、我々は主記憶上での関係データベース処理アルゴリズムを提案、実装してきた。本論文では外部記憶装置を含めた大規模関係データベース処理をデータパラレル記法により記述可能であることを示すとともにCM-2に実装した性能評価に関し報告する。

Large relational database processing and its evaluation on the Connection Machine CM-2

Hideaki OKADA Kazuhiko MATSUMOTO Masaru KITSUREGAWA
Institute of Industrial Science, University of Tokyo

Abstract

Many attempts have been made to accelerate the performance of the relational database processing by utilizing shared nothing parallel machines. On the other hand, few researches that utilize SIMD type machines called data parallel machines such as Connection Machine for the relational database processing. So far we proposed and implemented relational database processing algorithms on the CM-2. In this paper we show data parallel model can describe relational database processing. Then we implement large relational database processing on the Connection Machine CM-2 with secondary storage and evaluate its performance.

*現在、三菱電機情報システム研究所

1 はじめに

これまでに、関係データベースシステムの性能向上のために数多くの研究がなされてきた。中でもGAMMA[1, 2],SDC[4]などのようなshared nothingの並列データベースシステムは最も有効なアプローチの一つと考えられており開発、評価等が進められている。また、NCUBE, Sequentなどの汎用並列計算機上に関係データベースシステムを構築して性能向上を達成しているものもある。

一方でコネクションマシンをはじめとするデータパラレルマシンとよばれる、制御レベルではなくデータレベルの並列性を利用する並列計算機が注目されている。ここでいうデータパラレルとは並列計算機アーキテクチャについて述べているのではなく、計算モデルの意味で述べている。データパラレルモデルはプロセッサ数の増加などのシステムの拡張に容易に対応できるという大きな利点を持っている。

現在のところ、このようなデータパラレルマシンの応用分野はデータ並列性を有する大規模数値計算や画像処理が主であるが、データ並列性を有し、大容量データを扱う関係データベース処理も有効な応用分野となりうるのではないかと考えられる。

しかし、我々の知る限りではデータパラレルマシンによる関係データベース処理の研究はほとんど報告されていない。そこで我々は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するため、まず主記憶上でのジョイン処理アルゴリズムを提案、実装した[5]。本論文では、外部記憶装置を含めたシステムの性能を概観するために、そのアルゴリズムを拡張し、主記憶上では処理しきれない大規模なジョイン処理について述べる。

第2節においてコネクションマシンシステムについて、第3節では主記憶内のジョイン処理アルゴリズムについて概観する。第4節において大規模データベース処理アルゴリズムについて説明し、その性能測定結果を第5節で述べる。最後に第6節で結論を述べる。

2 コネクションマシンシステム

コネクションマシンCM-2 コネクションマシンCM-2[6]はSIMD型の超並列計算機である。CM-2は1bitのPEを16PE単位にハイパーキューブ網で結合しており、ローカル、グローバル通信ともにこれを用いている。各PEはそれぞれローカルメモリを持ち、最大構成時では65,536PEで構成される。各PEの命令はフロントエンドからシーケンサと呼ばれるハードウェアを通してブロードキャストされる。各PEはその命令を内部状態より選択して実行し、SIMD動作を行う。

仮想マシンアーキテクチャ CM-2のプログラミングではPARISとよばれる並列命令セットが使用可能である。PARISは仮想プロセッサ(VP)の概念を持つ。仮想プロセッサ数と物理プロセッサ数の比を仮想プロセッサ比(VP比)と呼ぶ。各物理プロセッサはVP比数分の仮想プロセッサの命令を実行する。したがって、プログラム実行時にはシステムの物理プロセッサ数とプログラムの仮想プロセッサ数に応じて動的に仮想プロセッサ比が決定される。PARISの仮想プロセッサの機能は、1つのデータ要素を1つの仮想プロセッサに割り当てることを可能にする。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。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。複数のVPセットを使用することもできるが、制御対象は常に1つのVPセットであり、VPの数は物理プロセッサ数の2の冪乗倍の数でなければならない。メモリ割り当てなどもこのVPセットに対する命令によって行われる。

データボルト データボルトはコネクションマシンの大容量二次記憶システムである、RAID-2レベルのディスクアレイである。その容量は単体で10G bytesであり、20Gbytesまで拡張可能である。転送速度は25MB/secを維持でき、複数のデータボルトを並列に使うと合計100MB/secを越える転送速度を実現できる。データボルトのファイルにも仮想プロセッサの概念があり、コネクションマシン上の主記憶とのデータのやり取りは同じVP数のVPセットに対してのみ可能である。

3 主記憶内ジョイン処理アルゴリズム

これまでにデータパラレルマシンの関係データベース処理応用の第歩として主記憶内にリレーションが収まる時のジョイン処理アルゴリズムについて、ソートマージ法を用いたアルゴリズムとハッシュによる分割を用いたアルゴリズムを提案し、コネクションマシンCM-2上に実装、評価している。本論文ではソートマージ法を用いたアルゴリズムDPSMJ(Data Parallel Sort Merge Join)を使用した。

リレーションの仮想プロセッサへのマッピングは、1タプルが1仮想プロセッサに対応するものとする。

DPSMJアルゴリズムは大きく三つのステップに分けられ、それぞれソートフェイズ、マージジョインフェイズ、置換フェイズと呼ぶ。それぞれのフェイズにおける動作を図1,2,3に示す。

ソートフェイズ

最初に、リレーションRとSが併合されて1つのリレーションMになる。リレーションMは最終的なリレーションJを得るための準備として一時的に作られるだけのものであり、それ自体では関係データベースのリレーションとしての意味は持たない。したがって、Mはキー属性とアドレス、リレーションのID(Rが0、Sが1とする)のみからなるタブルとして作成する。次にリレーションMの中でタブルがキー属性に従って昇順にソートされる。同じキー属性を持つタブルについては、リレーションIDを利用してRからのタブルの後にSからのタブルが並ぶという順序にする。

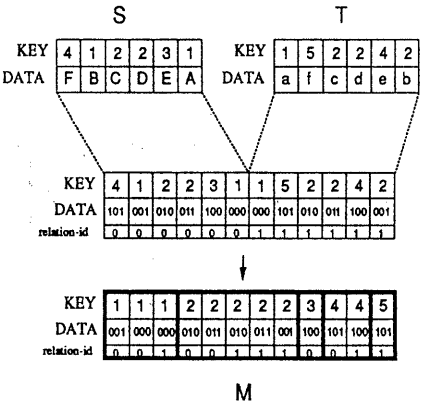

図 1: ソートフェイズ

マージジョインフェイズ

リレーションMの上で必要な処理を行い、ジョイン後のリレーションJを生成する。最初に、リレーションJのための空間をアロケートして必要なデータをそこに転送する。このフェイズはデータ送信とデータ複製からなる。

Mに属する全てのタブルは、自分と同じキー属性を持つタブルの数をかぞえる。Mのそれぞれのタブルは、自分と同じキー属性を持つタブルが、Jの中にいくつ存在することになるかを計算し、スキャン加算してJのタブル数を得る。

同じキー属性をもつタブルの中で最初のタブルが、生成されるJの数に応じたアドレスにデータを送信し、Jの中で必要な数だけそのデータを複製する。

置換フェイズ

マージジョインフェイズが終了した段階では、まだデータの組合せが適切なものではない。求めるリレーションを得るために、Sのタブルから送られてきたデータを再配置する。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。

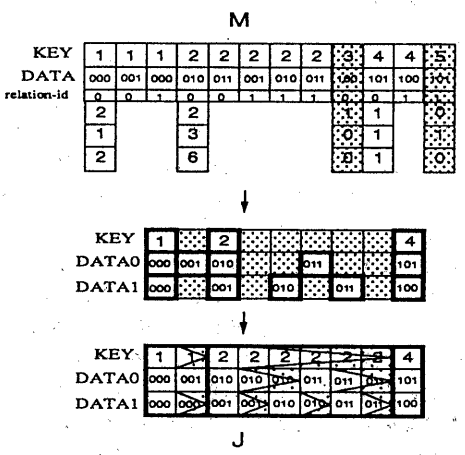

図 2: マージジョインフェイズ

4 データボルトを用いた大規模ジョイン処理アルゴリズム

4.1 アルゴリズムの概要

各リレーションを主記憶内で処理できるほどの大きさに分割する。これをスプリットフェイズと呼び、分割されたリレーションのそれぞれをバケットと呼ぶこととする。処理対象であるリレーションをバケットに分割する方法としては、ハッシュによる分割法を採用する。キー属性値を入力としてハッシュ関数を適用し、同じハッシュ関数値をもつタブルの集合が1つのバケットを構成する。したがって、同じハッシュ関数値を

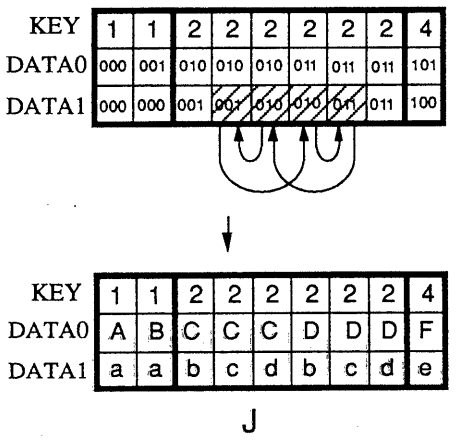

図 3: 置換フェイズ

もつバケット同士の主記憶内でのジョイン処理を全バケットについて行うことにより、全体のジョイン処理を完了する。これをジョインフェイズと呼ぶこととする。

ここでは、ジョイン処理されるリレーションをR、Sとする。スプリットフェイズにおいてN個のバケットに分割されるとき、リレーションR、Sから生成されたi番目のバケットをR_i、S_iとする。リレーションR、Sはそれぞれ、長さl_R、l_Sのタプルからなり、C_R、C_S個のタプルを持つものとする。また、キー属性の長さをl_{key}と表す。

はじめに、リレーションR、Sはデータボルトにファイルとして書き込まれている。このデータボルト上のファイルにもコネクションマシンの主記憶と同じくVP数という概念があり、あるVP数であるVPセットにより書き込まれたファイルは同じVP数のVPセットによってしか読み込むことができない。したがって、もしデータボルト上のファイルと異なるVPセットを読み書きする場合には、データボルト上のVPセットに等しいVPセットを確保しておき、それをI/Oバッファとして用いる。例えば、読み込み時にはI/Oバッファに読み込んだリレーションをsend命令を用いて、そのリレーションを必要としているVPセットに送る必要がある。このデータボルト上のVPセットのVP比をリレーションR、SについてそれぞれV_R、V_Sとする。また、物理プロセッサ数をPとする。

以下に、ジョインアルゴリズムの詳細について述べる。

スプリットフェイズ

スプリットフェイズでは、リレーションはキー属性値のハッシュ関数値に応じたバケットに分割され、データボルトに書き込まれる。リレーションR、Sについてそれぞれ独立に同じ処理を行なうので、以下の説明ではリレーションRのみについて説明する。

スプリットフェイズでは、スプリットバッファ、バケットバッファという2種類のバッファが少なくとも必要となる。スプリットバッファは読み込んだリレーションのキー属性に対してハッシュ関数値を計算して、各タプルがどのバケットに属するかを決定する。バケットバッファには、同じハッシュ関数値を持つタプルがスプリットバッファより送られる。それらをデータボルトに書き込む。

このスプリットバッファとバケットバッファの割り当て方法には次の2つ方法が考えられる。1つは図4に示すように、巨大なスプリットバッファとバケットバッファを1つだけ用意して共用する。もう一方は、図5に示すようにファイルのVPセットと同じVPセットであるスプリットバッファを用意して、バケットバッファは分割数と同じだけ用意する。

ここでは、次のような理由から前者の方法を採用した。

ハッシュ関数の値に応じてタプルをバケットバッファに送る時に、送信先になれるバッファは1つしか存在しない。すなわち、後者の方法を採って複数のスプリットバッファを小さくとった時には、分割数に応じたバケットバッファが存在していることになるが、スプリットバッファからすべてのバケットバッファへ同時にタプルを送ることは、コネクションマシンのVPセットの制約により許されていない。したがって、バケットバッファを多く確保することによる大きな利点を活かせないからである。

前者の方法を選択すると、スプリットフェイズではリレーションRを読み込むためのI/OバッファをVP比V_Rで確保し、V_Rの倍数でありかつ、できるだけ大きいVP比V^R_{split}でスプリットバッファを確保する。バケットバッファに送られるタプル数は、平均的にはスプリットバッファに入るタプル数の分割数N分の1となるので$\frac{V^R_{split}}{N}$、もしくはキー属性値の分布を考慮して$\frac{V_{split}}{N}$より大きいVP比V_{bucket}のバッファを確保する。VP比と主記憶割り当ての関係や分割数の決定方法については後に触れる。

以上をまとめるとスプリットフェイズでは、以下の処理を$\frac{C_R}{PV^R_{split}}$回行なうことになる。

1. 次の処理を最大で$\frac{V^R_{split}}{V_R}$回行なう

   (a) データボルトから入力バッファにVP比V_R

でタプルを読み込む

(b) 入力バッファからスプリットバッファへsend命令によりタプルを送る

2. スプリットバッファにおいてハッシュ関数値を計算する
3. 次の処理をiが0から$N-1$のときについてN回行なう
   (a) ハッシュ関数値がiであるバケットを数え、バケットバッファに送る
   (b) バケットバッファに送られてきたタプルをデータボルトにR_iバケットのファイルとして書き込む

データ分布の不均一性などにより3(a)段階において、バケットバッファの溢れが生じる場合がある。そこで、溢れが生じる場合にはバケットバッファに入る切るタプルは出力し、入り切らないのもはスプリットバッファに保存しておく。リレーション読み込み時にはI/Oバッファからスプリットバッファの空いている領域へタプルを転送する。

ジョインフェイズ

ジョインフェイズでは、図6に示すようにスプリットフェイズで生成されたバケットR_i,S_iをジョインバッファと呼ばれる2つのバッファにそれぞれ読み込んで主記憶内でのジョイン処理を行ない、結果をデータボルトに出力することを分割数の数だけ繰り返す。主記憶内でのジョイン処理アルゴリズムには、DPSMJアルゴリズムを使用する。

スプリットフェイズにおいてV^R_{bucket}のVP比でデータボルト上に書き込まれているバケットを読み込むには、V^R_{bucket}のVP比を持つバッファを用意しなければならない。そのバケットに対する書き込みが1度であれば、そのバッファがジョインバッファとなることができるが、複数回の書き込みを行なう必要があれば、そのバッファは単なる入力バッファとして使用され、バケットのサイズに応じたVP比V^R_{join}でジョインバッファを新たに確保する必要がある。

スプリットバッファよりもリレーションのサイズが大きい場合には、バケットバッファの書き込みが複数回行なわれる。したがって多くの場合、このような入力バッファが必要となる。

ジョインフェイズでは以下のような処理をN回行なう。

1. バケットR_iをすべて読み込むまで、次の処理を繰り返す
   (a) バケットを入力バッファに読み込む

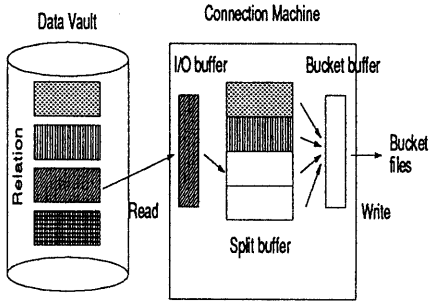

図 4: スプリットフェイズバッファ割り当て法1

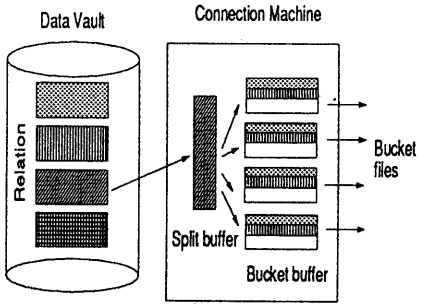

図 5: スプリットフェイズバッファ割り当て法2

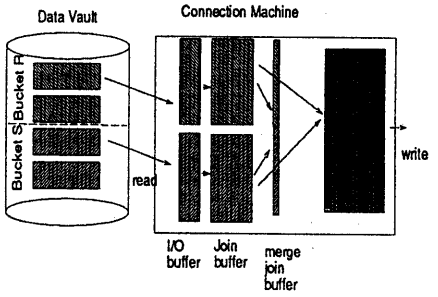

図 6: ジョインフェイズバッファ割り当て法

(b) 入力バッファからジョインバッファへ転送する

2. バケット S_i についても同様の処理を行なう
3. リレーション R,S から生成されたバケットファイルの VP 比を V^R_{join}, V^S_{join} とすると、キー属性とアドレス、リレーション ID からなる、

$$V^R_{join} + V^S_{join} \leq V_{merge}$$

をみたす VP 比 V_{merge} のリレーション M を用意し、R_i、S_i のキー属性、アドレス、リレーション ID を M に送る。
4. リレーション M において DPSMJ を部分的に利用してソートを行なう
5. DPSMJ により生成される結果リレーションのタプル数を求め、その数に応じた VP 比 V_{result} で長さが $l_S + l_R - l_{key}$ のバッファ J を確保する。確保できない時は、利用可能な主記憶量だけ確保して、以下の処理を分割して行なう。
6. リレーション M の内容に応じて、R_i、S_i からデータを J に送り、J 内で完全な結果リレーションの生成を行なう
7. 結果リレーションをデータポルトに出力する

4.2 主記憶の割り当て

このデータポルトを用いたジョイン処理アルゴリズムにおける各バッファの主記憶上の割り当てについて考察する。主記憶量を M、リレーション R、S のサイズを M_R, M_S と表す。

4.2.1 スプリットフェイズ

スプリットフェイズでは、入力バッファを VP 比 V_R で、スプリットバッファを VP 比 V^R_{split} で割り当てている。バケットバッファは、キー属性値の分布によるバケットの溢れを考慮した変数 α を用いて表すと、VP 比

$$V^R_{bucket} = \alpha \frac{V^R_{split}}{N} \tag{1}$$

で割り当てられる。α はバケットの溢れを大きく考えるほど 1 より大きくなっていく。

したがって、入力バッファ、スプリットバッファ、バケットバッファを確保するのに必要な主記憶量 $M^R_{split-inbuf}$, M^R_{split},M^R_{bucket} はそれぞれ、

$$M^R_{split-inbuf} = l_R P V_R \tag{2}$$

$$M^R_{split} = l_R P V^R_{split} \tag{3}$$

$$M^R_{bucket} = l_R P V^R_{bucket} = \alpha \frac{V^R_{split}}{N} l_R P \tag{4}$$

となる。これらが、主記憶内に収まるには

$$M^R_{split-inbuf} + M^R_{split} + M^R_{bucket} < M \tag{5}$$

を満たせばよい。

$M^R_{split-inbuf}$ は処理するリレーションによって決まっており、V^R_{bucket} と V^R_{split} には、式 (1) の関係が成立するため、

$$M^R_{split}(1 + \frac{\alpha}{N}) < M - M_{split-inbuf} \tag{6}$$

を満たす M^R_{split} が割り当て可能である。分割数 N はジョインフェイズに対する考察により決定されるが、$\alpha \leq N$ であるから、

$$M^R_{split} < \frac{M - M_{split-inbuf}}{2} \tag{7}$$

である M^R_{split} は割り当て可能である。したがって、M^R_{split} は、$\frac{M}{2}$ 程度には割り当て可能であるといえる。

4.2.2 ジョインフェイズ

ジョインフェイズでは、R_i, S_i 用の入力バッファ、ジョインバッファ、マージジョインバッファ、結果出力バッファが必要となる。入力バッファはそれぞれ VP 比 $V^R_{bucket}, V^S_{bucket}$ で割り当てられる。

ジョインバッファは、VP 比 $Vjoin^R$ で割り当てられる。

マージジョインバッファは、$V^R_{join} + V^S_{join} \leq V_{merge}$ を満たす VP 比 V_{merge} で割り当てる必要があるが、VP 比は 2 の冪乗であるので

$$V_{merge} = 2max(V^R_{join}, V^S_{join}) \tag{8}$$

で割り当てればよい。

結果出力バッファの VP 比 V_{result} については、選択率を考慮する必要がある。しかし、ジョイン処理時の選択率は動的に決定され、結果リレーションを生成するバッファも動的に確保するが、このバッファをどの程度の大きさにするかを決める必要がある。そこで、仮の選択率とも言える β を

$$\beta = \frac{V_{result}}{V_{merge}} \tag{9}$$

と定義する。100% ジョインは $beta = 1$ のときに対応する。実際に生成されるタプル数が PV_{result} を越えた場合には、結果リレーションの生成、出力過程を分割して行なう。

したがって、入力バッファ、ジョインバッファ、マージジョインバッファ、結果出力バッファに必要となる主記憶量 $M^R_{join-inbuf}, M^S_{join-inbuf}, M^R_{join}, M^S_{join}$,

M_{merge}, M_{result} は、アドレスを表すのに必要な領域の長さを l_{adrs} と表すと、

$$M^R_{join-inbuf} = l_R P V^R_{bucket} \quad (10)$$

$$M^R_{join} = l_R P V^R_{join} \quad (11)$$

$$M_{merge} = 2(l_{key} + l_{adrs}) P max(V^R_{join}, V^S_{join}) \quad (12)$$

$$M_{result} = 2(l_R + l_S - l_{key}) P max(V^R_{join}, V^S_{join}) \beta \quad (13)$$

となり、

$$M^R_{join-inbuf} + M^S_{join-inbuf} + M^R_{join} + M^S_{join} + M_{merge} + M_{result} < M \quad (14)$$

を満たすようにすればよい。

$l_{key}, l_{adrs} \ll l_R, l_S$ と仮定すると、

$$M^R_{join-inbuf} + M^S_{join-inbuf} + M^R_{join} + M^S_{join} + M_{result} < M \quad (15)$$

とできる。

ここで、

$$N^R_{split} = \frac{V^R}{V^R_{split}} \geq 1 \quad (16)$$

である β を定義する。β は、キー属性値の分布が等しい場合にはスプリットフェイズにおいてバケットに対して書き込みがあった回数、すなわち V_{join} が V^R_{bucket} の何倍になるかを示している。このとき、

$$M^R_{join} = N^R_{split} M_{join-inbuf} \quad (17)$$

となる。

したがって、式(15)は

$$M^R_{join}(1 + \beta + \frac{1}{N^R_{split}}) + M^S_{join}(1 + \beta + \frac{1}{N^S_{split}}) < M \quad (18)$$

とできる。

M^R_{join}, M^S_{join} より一度に主記憶内でジョインできるタプル数が確定する。そして、分割数が、

$$N = max(\frac{M_R}{M^R_{join}}, \frac{M_R}{M^S_{join}}) \quad (19)$$

ときまる。

5 性能評価

以上のようなデータボルトを用いたジョイン処理アルゴリズムをコネクションマシン CM-2 に実装した。ここでは、その性能評価、解析を行なう。

使用したのは 8K プロセッサ、ローカルメモリ 8KB、全体では 64MB の主記憶を持つ CM-2 である。この測定では、タプルのキーは、1 からタプル数までの値を重複なく設定している。すなわち、100% ジョインになる。測定時間は、コネクションマシンの処理時間のみであり、フロントエンドの処理時間は含まない。測定に当たり、キー長は 4 バイト固定とし、タプル長は 64,128,256, 512 バイトの 4 種類の値をとるものとする。両方のリレーションは同じタプル長、同じタプル数であるとし、リレーションのサイズを 8,16,32,64MB と変化させた。タプル数は、タプル長とリレーションのサイズに応じて決定され、16K から 1024K の値をとる。

主記憶の割り当てについては、$\alpha = 1, \beta = 1, N_{split} = 1 or 2$ として行なった。スプリットフェイズでは 32MB をスプリットバッファに、バケットバッファ、入力用バッファは 4MB とした。ジョインフェイズにおいては、入力用バッファ、ジョインバッファはリレーション毎に 4MB とし、出力バッファは 8MB とする。これらは第 4 節において導いた式から得られる値と比べると小さくなっているが、これはユーザには見えてこないメモリ領域、例えばソートルーチンでの使用、などにより M が主記憶量より小さくなっているためである。

図 7,8,9 にそれぞれ、全体の処理時間、スプリットフェイズの処理時間、ジョインフェイズの処理時間を示す。また、リレーションサイズが 32MB のときのスプリットフェイズ、ジョインフェイズの処理時間の内分けを図 10,11 に示す。

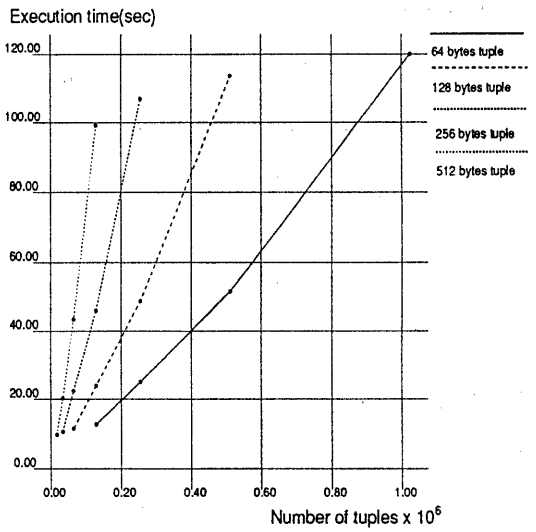

図 7: データボルトを用いたジョイン処理プログラム全体の処理時間

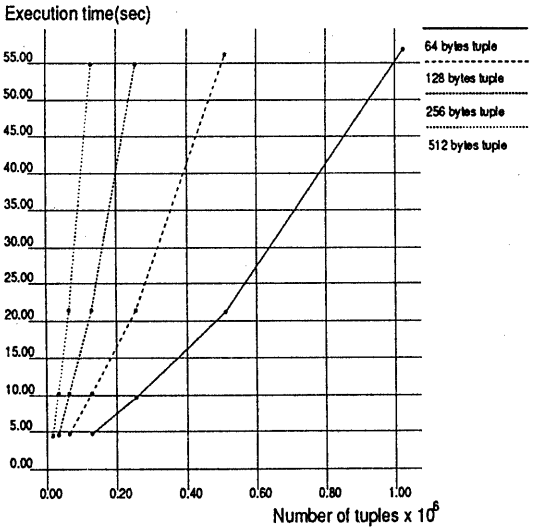

図 8: スプリットフェイズの処理時間

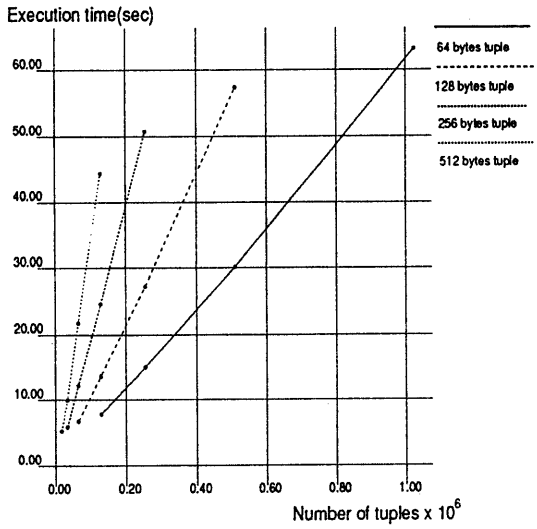

図 9: ジョインフェイズの処理時間

"DISK I/O" はデータボルトとの入出力時間を表す。スプリットフェイズにおける、"send to bucket" はスプリットバッファからバケットバッファへのデータ転送時間、"send to split buffer" は入力バッファからスプリットバッファへの転送時間、"hash" はハッシュ関数値を求める時間を表す。また、ジョインフェイズにおける"sort" はキー属性のソート時間、"make result" はキー属性のソート結果に基づいて結果のリレーションを生成する時間、"send to join buffer" は入力バッファからジョインバッファへのデータ転送時間、"merge join" はソート処理と結果タプルの生成以外の DPSMJ の時間をそれぞれ表している。

スプリットフェイズでは、処理時間の 7 割以上がデータボルトとの入出力によって占められていることがわかる。ジョインフェイズではタプル数に依存してソートの処理時間によって割合いは変わるものの処理時間の約半分がデータボルト入出力に費やされている。データボルトとの入出力性能はタプル数ではなく容量に依存している。

このジョイン処理でのデータボルトとの I/O 性能を転送速度で表すとスプリットフェイズで 6MB/s 程度、ジョインフェイズで 8-10MB/s と、最大転送速度の 1/2 から 1/4 しか引き出せていないことが分かる。これは、入出力のアクセス単位が 4-8MB とあまり大きくないことが原因として考えられる。ジョインフェイズの方が転送速度が大きいのはそのためであり、結果リレーションの出力時のアクセス単位が最も大きいからである。

また、リレーションのサイズが 32MB を越えると性能が落ちていくことがわかる。これらは、次のような理由による。

リレーションのサイズがスプリットバッファのサイズより大きい場合に、我々はバケットバッファをディスクに書き込む際に VP 比を小さくして書き込んでいる。これは第 4.1節でも説明したように、データボルトを用いたコネクションマシンのファイルシステムは VP セットそのものを書き込むために、バケットバッファに空きが多いと入出力性能に影響を及ぼすからである。また、中身のつまったバケットを読み込んでジョインバッファに送る時には、send 命令ではなく、次に述べるような分散した、負荷の小さい処理で置き換えられ、データ移動の時間が短縮されるからである。したがって、64MB のリレーションを処理する時にはスプリットバッファが 32MB で分割数が 16 であるから、平均的なバケットのサイズは 2MB となるため 4MB ではなく、半分の 2MB で処理している。図 8において、32MB リレーションに対する処理のデータを境にして直線の傾きが変化しているのは、2MB というデータボルトにとっては小さな書き込み単位で出力している

ため、転送速度が小さくなっているからである。

スプリットフェイズのバッファの割り当て方法について、我々は図4に示した方法を選択をした。その結果、小さなVP比のI/Oバッファから大きなVP比のスプリットバッファへのデータの移動が必要になる。このようなデータの移動はスプリットフェーズのバッファの割り当て方法の選択理由で述べたように、データの移動先の物理プロセッサの偏りが発生することになる。このデータの移動は本来は、物理プロセッサのメモリ上でのデータの移動にすぎないことから、我々は、移動先の物理プロセッサが分散するようにsend命令の移動先のアドレスを変更している。

次に、処理時間のディスクI/O以外の時間に注目してみると、VPセット間でのデータ転送の時間が大きいことがわかる。スプリットフェイズでは24%が、ジョインフェイズでは15%がそれらの時間に割かれている。スプリットフェイズでのデータ転送とは、スプリットバッファからバケットバッファへの転送が大部分を占めている。残りは、入力用バッファからスプリットバッファへの転送時間である。ここにも、ファイルにおけるVPセットの弊害が現れている。計算が主であるハッシングの部分は非常に小さい。ジョインでフェイズも、結果のリレーションを得るためのsend処理と、入出力用バッファのデータ転送時間が大きな時間を占めている。

全体の処理の内で、ディスクやプロセッサ間でのデータのやりとりを除いた処理は、スプリットフェイズのハッシングとジョインフェイズのソートである。ハッシングの占める割合は小さい。これはその処理の内容から当然といえる。ここで、ソートの時間が大きいといえる。しかも、このソート処理では、ソートキーとアドレスのみを扱っており、タプルのデータ部分の転送時間は含まれていない。このことからも、ソート処理の負荷の大きさが良くわかる。これに関しては、我々はソート自身に関する研究を進めている[10]。

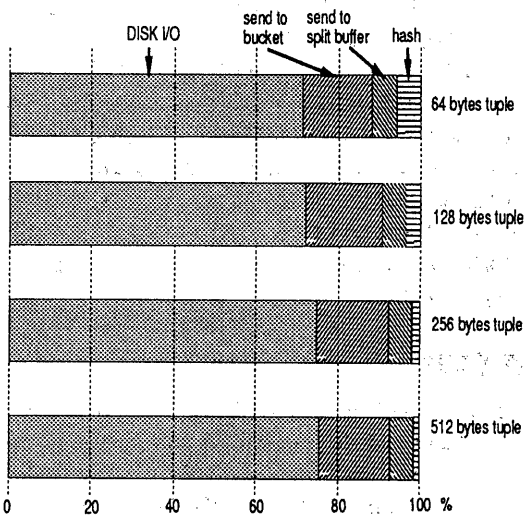

図 10: スプリットフェイズの処理時間の内分け

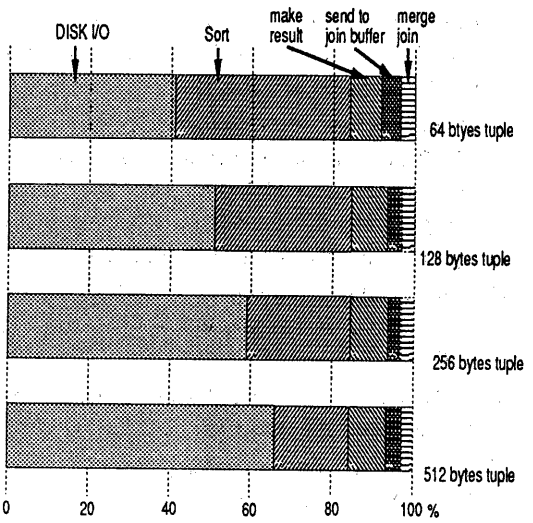

図 11: ジョイン各フェイズの処理時間の内分け

6 おわりに

これまでに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に関する研究はほとんどなされていなかった。そこで我々は外部記憶装置とのI/Oをも含めたデータパラレルジョインアルゴリズムをデータボルトと呼ばれるディスクアレイを備えたコネクションマシンCM-2に実装し、その性能について評価した。

まず、データパラレル記法により外部記憶装置を含めた関係データベース処理が可能であることを明らかにした。

現在のシステムでは充分な性能向上は必ずしも得られず、仮想プロセッサ機能によるメモリ、通信、ファ

イルに対する制限が大きいことが分かった。

本論文ではコネクションマシンCM-2への実装を扱ったが、アルゴリズム自身はデータパラレルマシンに汎用的に利用でき、仮想プロセッサに対する主記憶割り当てに関する形式化も有効である。

本論文のプログラムはまだ第一段階にすぎない。データボルトにおいては同期I/Oの他に、非同期I/Oの機能を備えている。この機能を用いた性能向上方法やジョイン処理における主記憶割り当て方法についてさらに詳細に検討し、可変長レコードなどの複雑な条件への対応などについて検討したい。また、VPの制限が緩和されていると言われているCM-5への実装なども行っていきたい。

参考文献

[1] 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.

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

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

[4] M. Kitsuregawa, *et al*: The Super Database Computer (SDC) System Architecture, Algorithm, and Preliminary Evaluation Proc. of the 25th Hawaii International Conference on System Sciences, pp.308-319, 1992.

[5] M. Kitsuregawa, K. Matsumoto: Massively Parallel Relational Database Processing on the Connection Machine CM-2, The 2nd International Symposium on Database Systems for Advanced Applications (DASFAA'91), pp.226-235, 1991.

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

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

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

[9] 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.

[10] 岡田, 喜連川, 高木: 超並列計算機における各種ソートアルゴリズムの実装と評価, 情報処理学会研究報告 92-ARC-97, 情報処理学会, 1992.