データベースシステム 100−13
（1994. 10. 14）

分散メモリマシン上の並列ハッシュ結合による ディスクとメモリ利用の影響

古野 真太郎　　横田 治夫
北陸先端科学技術大学院大学
{furuno,yokota}@jaist.ac.jp

概要

関係データベースにおける結合演算は処理コストの高い演算であり、その並列化も含め、これまでに様々の方式が提案されている。我々は、その中でもハッシュに基づく方式をメッセージパッシング型の超並列計算機 nCUBE2 上に実装し、限られた資源の中で、ディスク、メモリの割り当てがその性能にどのように影響するか測定を試みた。ディスクの利用形態としてディスク・ストライピングを使用した方式、ディスクアクセスを独立させた方式、バケット (中間リレーション) をメモリ上に保持する方式を実験した。nCUBE2 では、バッファを介した非同期メッセージ通信を基本としているが、有限のバッファサイズの下でオーバーフローを発生させずに効率の良いプロセッサ間通信を実現することが重要となる。

The Influence of Disk/Memory Utilization for Parallel Hash-Based Join Algorithms on a Distributed-Memory Machine

Shintaro FURUNO　　Haruo YOKOTA
School of Information Science
Japan Advanced Institute of Science and Technology,Hokuriku

Abstruct

The relational database join is one of the most time consuming operation, and various methods, including parallel approaches, are have been investigated. We have implemented parallel hash-based join algorithms on nCUBE2, a message-passing massive-parallel computer, to measure the performance by varying the assignment of disks and memories in limited resources. We have implemented three methods: disk-striping, independent accesses to devices, and storing intermediate relations into memories. Since communication in nCUBE2 is based on asynchronous mechanism using buffers, it is important to implement effective communication between processors by inhibiting buffer overflow under limited memories.

1 はじめに

関係データベースにおける結合演算は処理コストの高い演算であり、その並列化も含め、これまでに様々の方式が提案されている。それらは大きく分けて、ネステッドループ方式、ソートマージ方式、ハッシュに基づいた方式の三つに分類できる。その中でもハッシュに基づく方式は、並列化を図る際に各オペランドタプルを複数のプロセッサに割り付けることが可能であり、他の方式に比べ高い性能を得ることが知られている [Dew85]。

ここでは、東京大学の喜連川らが提案する GRACE ハッシュ方式 [Kit83] およびウィスコンシン大学の DeWitt らが提案するハイブリッドハッシュ方式 [Dew84] を、メッセージパッシング型並列計算機 nCUBE2 上に実装し、ディスクおよびメモリの効率的な利用方法について検討する。この並列化されたハッシュジョインアルゴリズムの性能評価については既に共有マルチプロセッサ環境 [Kit92a]、専用マシン [Dew89][Kit92b] において行われている。

マシンのスケーラビリティ、汎用性を考えた時、近頃主流となりつつあるメッセージパッシングを通信スタイルとする汎用分散メモリ並列マシン上で、効率の良い結合演算を実現することも重要となってくる。分散メモリ並列マシン上にハッシュジョインアルゴリズムを実装する場合、メッセージ通信をいかに効率良く行うかが一つの鍵となる。特に nCUBE2 の場合には、バッファを介した非同期通信により通信の効率化を図っているが、大量のデータ転送を前提とするデータベース処理では、有限のバッファサイズの下でいかにオーバーフローを発生させずに効率良くプロセッサ間通信を実現するかが重要となる。本報告では、非同期通信とハンドシェイクを行った場合の処理時間の比較を行うと同時に、与えられた条件の下での非同期通信の限界について考察する。

また、nCUBE2 では、各プロセッサは I/O プロセッサを介してほぼ独立にディレクトリにアクセスすることができる。さらに、その長所を生かしてソフトウェア的にディスク・ストライピングを実現している。本報告では、ソフトウェアによるディスク・ストライピングを並列ハッシュジョインに利用した場合の性能についても測定を行った。

超並列マシンは、単体ノードのメモリ容量が小さくても全体として非常に大きなメモリ空間を持つことができる。このため、ハッシュジョインの中間状態で作成されるバケットをメモリ上に置くことも、ある程度のサイズまでは可能となってきている。そこで、バケット (中間リレーション) をメモリ上に保持する方式も試みた。

以下第 2 節では前提条件として実現に用いた nCUBE2 の構成について、第 3 節では nCUBE2 上での並列ハッシュジョインのモデルについて述べる。第 4 節では実験結果、そして第 5 節でリレーションサイズとシステム構成に関する考察について言及する。

2 nCUBE2 の構成

nCUBE2 はハイパーキューブ結合による超並列計算機で、各ノードプロセッサは独自の 64bitCPU を用いている。本学のシステムは 256 のプロセッサからなる。各ノードが保有するメモリ容量は、物理ノード 0 番から 15 番までの 16 ノードが 16MB、それ以外は 4MB の設定である。各ノードは 14 本の双方向DMAチャンネルを持ち、その中の 13 本がハイパーキューブ結合に用いられ、残りの 1 本が I/O サブシステムとダイレクトに接続されている。

2.1 ディスクの結合形態

I/O サブシステムは 8 個の I/O プロセッサに 1GB のディスク 16 台を SCSI 接続した構成であり、各 I/O プロセッサに 2 台のディスクがディジーチェーンで結ばれている。概略を図 1に示す。

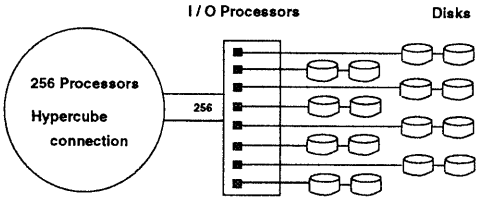

図 1: nCUBE2 のディスク結合形態

ディスク・ストライピングは、各ディスクを複数のパーティションに分けそれぞれ一つのパーティションを 8 台つなぎ合わせたものであり (I/O プロセッサもそれぞれ異なる)、これがソフトウェア的に実現されている。これを nsdisk というドライブ名でアクセスすることができる。

現在の nCUBE2 の I/O サブシステムでは、構造の単純さの理由により、ディスクのキャッシングテーブルを I/O プロセッサ上に構成している。従って、同一プロセッサでの同一ページのアクセスでも I/O プロセッサへの通信が発生する。

2.2 プロセス間通信の方式

プロセッサ同士の通信形態はメッセージパッシングであり、送信側のプロセッサは受信側のプロセッサの返答なしにメッセージを送り続けることが可能な非同期の通信となっている。しかし、受信側のプロセッサのバッファは有限であるので、受信側のプロセッサのローカルな処理によりメッセージの取り出しが遅れるような場合は、バッファの溢れが発生しその時点で対象のプロセッサはハングアップの状態となってしまう。

このオーバーフローを抑えるため、プロセッサの割り当て等を変更して、データ通信量と各プロセッサでのローカルな処理に費やされる時間の調整を行うことも可能である。しかし、通信量が増加するとこのような調整では追い付かなくなる。そのような場合は同期型の通信に変更する必要があるため、ソフトウェアでハンドシェイク方式を実現することになる。その場合、データを送信する側のプロセッサが受信対象のプロセッサに対してデータの受け取りが可能かどうかの確認を毎回行い、受信側のプロセッサはメッセージ受信用バッファが空くまで応答のメッセージを返さない。従って、送信側のプロセッサはその応答のメッセージが戻ってくるまでデータ転送を待たされることになり、受信側のバッファの溢れという現象は発生しない。

3 nCUBE2 上での並列ハッシュジョインのモデル

以下では、リレーション R と S に対し等結合を行い、結果をリレーション T として格納することを前提とする。またハッシュバケット B を $BR_0,...,BR_{m-1}$、$BS_0,...,BS_{m-1}$ と呼ぶ。

3.1 nsdisk を使用した方式

ここでは、nCUBE2 で用意されている、ソフトウェア・ディスクストライビングのデバイスコントローラ nsdisk を利用した、GRACE ハッシュ方式を先ず考える。

3.1.1 バケット分割フェーズ

予備実験として、リレーション R と S の読み出しにおいて、

- 先ず R を読み、次に S を読み出す。
- R と S を同時に読み出す。

というパターンと、

- R および S をある一つのプロセッサで読み出しハッシュして分配する。
- 各プロセッサが R と S を読み出してハッシュする。
- R 用のプロセッサと S 用のプロセッサを決め、それぞれが読み出しハッシュして分配する。

というパターンを組み合わせてバケット分割フェーズに要する時間を計測してみたところ、

「R 用のプロセッサと S 用のプロセッサを決め、それぞれが同時に読み出しハッシュして分配する」

というパターンが最も高速であった。このため、以下のような方針で実験を行った。

1. リレーションの読み出しにおいて担当のプロセッサを決め、そのプロセッサは可能な限りバケットへの書き込み処理を行わない。具体的にはプロセッサ数が三以上の場合には、リレーションの読み出しは物理ノード 0 番と 1 番のプロセッサが担当し、それ以外のプロセッサがバケットへの書き込み処理を行なうことにする。
2. 書き込み担当のプロセッサにおいて R と S のタプルが並行に到着することにもなるので、受け取ったタプルデータがどちらのタイプのものであるかを判別し、各バケットファイルへの書き込みを行う。

R、S、$BR_0,...,BR_{m-1}$、$BS_0,...,BS_{m-1}$ は全て nsdisk に格納されている／される。この状態を図 2に示す。

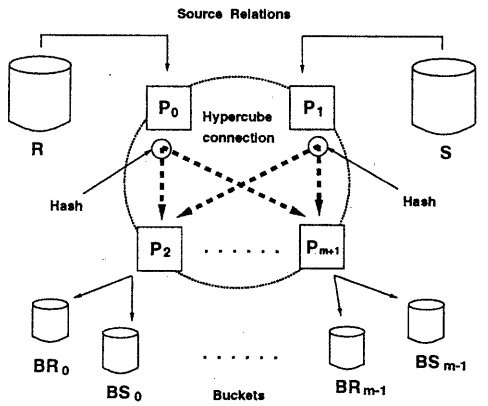

図 2: バケット分割フェーズ

なお、読み出されたタプルの結合属性に対して適用されたハッシュ関数の結果値に応じて、その内容が該当のプロセッサに送られるが、データの偏りが発生した場合には、プロセッサの負荷が偏ることになる。ここでは、単純化のため、先ずはデータの偏りのない乱数を用いる。

3.1.2 結合処理フェーズ

バケット分割フェーズと同様に、バケットの読み出しと結果のリレーション T への書き込みは異なったプロセッサで行う。ここで、非同期通信の場合には、結果用のリレーション T はプログラム開始時に指定されるプロセッサのキューブ次元数[1]によりその数を変化させる必要がある。これは、タプルデータの受け取りの際に書き込み担当のプロセッサのメッセージ受信用のバッファが溢れないよう、結合処理と書き込みを担当するプロセッサ数を、バケット数の増加に併せて調整するためである。この状態を図 3に示す。書き込み担当プロセッサへのタプルの振り分けもハッシュを用いて偏りのないようにしている。また、T も nsdisk に格納される。

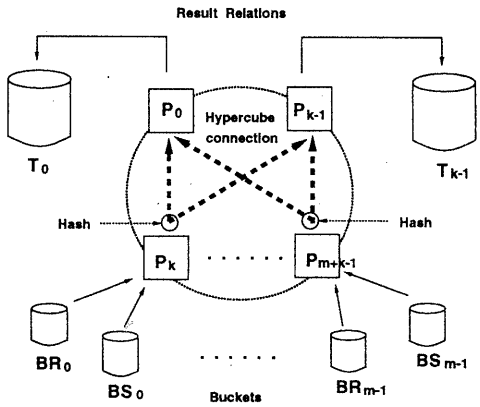

図 3: 結合処理フェーズ

3.2 独立にディスクアクセスを行う方式

今回の実験環境の場合、nsdisk は 1 つしか用意できない。nsdisk は、1 本の入出力ストリームに対してバンド幅を広げる効果は期待できるが、ハッシュ結合のように、同時に複数のストリームが必要な場合は、同期操作のオーバーヘッドやシークの競合により、必ずしも良い性能を示すとは思われない。nCUBE2 では、I/O サブシステムに結合されたディスクを、nsdisk としてではなく独立したディスクとしてアクセスすることも可能である。そこで、図 2および図 3において R、S、$BR_0,...,BR_{m-1}$、$BS_0,...,BS_{m-1}$、$T_0,...,T_{k-1}$ を 8 台もしくは 16 台のディスクに別々にアクセスするよう設定する。

3.3 バケットを分散しメモリ上に保持する方式

2 節で述べたように nCUBE2 の各ノードにおいては物理ノード 0 番から 15 番までが 16MB、それ以外の 240 個の

[1] キューブ次元数:n が与えられた時のプロセッサ数:N は $N = 2^n$ となる。

ノードでは4MBのメモリを保有している。これは、計算上16MB × 16 + 4MB × 240 = 1.216GBのメモリの利用が可能である。今後、半導体メモリは益々大容量で安価になることは明白であるから、バケットをメモリ上に置いて処理することも十分可能となる。仮にリレーションRおよびSのタプルサイズを208B、タプル数を各300,000と設定した場合でもバケットの合計サイズは約125MBである。従って、1ノードにおいてバケットに使用できるメモリが2MBとしても、64ノード用いれば全てメモリ上で処理できる。このようにハッシュ先のバケット数を増やし、配置するノード数を増加することにより、バケットをディスクに格納せずメモリ上に置いても処理できることになる。その場合、ハイブリッド・ハッシュジョインを適用することによりリレーションS用のバケット$BS_0,...,BS_{m-1}$を格納する必要がなくなり、さらに大きなリレーションまで扱うことができるようになる。このモデルを図4に示す。

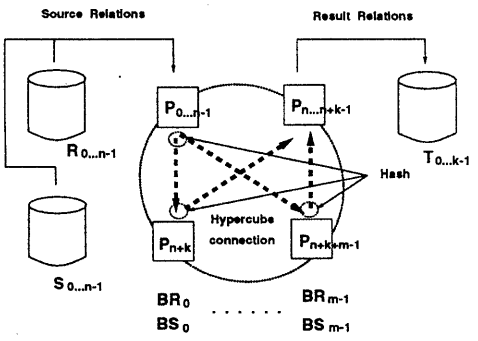

図4: バケットを分散しメモリ上に保持する方式

4 実験結果

実験に用いたリレーションは、ウィスコンシン・ベンチマーク[Gray91]に基づいている。そのリレーションの各タプルは13個の4Bの数値と、3個の52Bの文字列属性から構成される。また、ベンチマークにおける結合の問合せ式は*joinABprime*[2]である。

ここでは1,000個のタプルのリレーションR(約200KB)と10,000個のタプルのリレーションS(約2MB)を結合し、結果として1,000個のタプルのリレーションT(約400KB)を作成する場合と、その10倍のサイズのリレーションを用いた場合の、2種類のパターンを中心に測定を行った。なお、測定時のデータの配置は[Dew89]において試みられているケースと同様に2つのリレーションとも均一(Uniform)な状態としている。

それぞれの方式について、ハンドシェイクを使用する場合と使用しない場合での比較も行った。

4.1 nsdiskを使用した方式

nCUBE2におけるソフトウェア・ディスクストライビングであるnsdiskでは、8台のディスクパーティションに対して並行なファイル・オペレーションが可能である。その場合、nCUBE2が用意している専用テーブルの中のエントリを指定することにより、任意のブロックサイズを選択することができる。ここでは、ディスクのブロックサイズの違いによる性能を確認するために、複数のブロックサイズにおいてベンチマークの測定を行った。バケット数は22、従って使用対象のプロセッサ数はバケット分割フェーズでは24、結合処理フェーズでは26となる。また、各リレーションのタプル数をR=1,000、S=10,000とし、1タプル毎(208B)と100タプル毎(20.8KB)でファイルアクセスを行った。この結果を図5に示す。

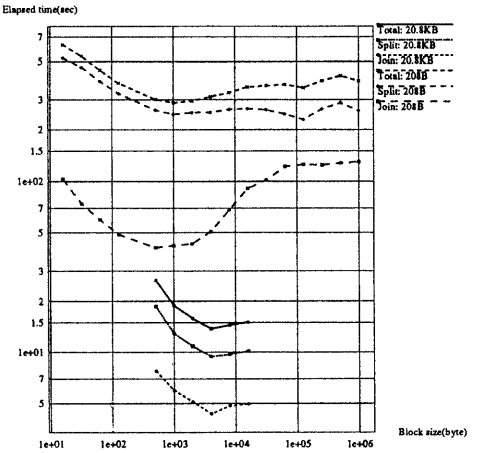

図5: nsdisk利用時のブロックサイズの影響(1,000 × 10,000)

ブロックサイズの違いによりその性能にばらつきがあり、ファイルアクセスの単位が208Bの場合では、1KBのブロックサイズにおいて全体の経過時間が288.0secと最良値を示した。一方、20.8KBの場合は、4KBのブロックサイズにおいて全体の経過時間が13.8secと最良値を示した。従って、以下20.8KBをファイルアクセス単位として実験を行った。

4KBのブロックサイズにおいてタプル数を10倍(R=10,000、S=100,000)にして測定したところ、バケット分割フェーズで74.0sec、結合処理フェーズで19.7sec、全体で93.7secであった。

次に、この4KBのブロックサイズを対象としてバケット数(m)を変化させた場合の性能の比較を試みた。これは、ハンドシェイクを使用する場合と使用しない場合を併せて行なった。この結果を図6に示す。

ハンドシェイクの有無にかかわらず相対的にバケット数が多い場合ほど、かえって処理時間が長くなることが判る。これは、分散メモリの利用という観点では、バケット数が多いほど並列化の効果が大きいと考えられるが、ディスク・ストライビングの使用によりディスク資源の競合が発生していることが予想される。つまり、R、Sと同一のnsdiskにバケットを配置させることにしたため、バケット数が増えるとシーク数も増加することになる。従って、必要以上にバケット数を増加させることは性能のを低下を引き起こす。

[2]INSERT INTO TMP
SELECT * FROM TENKTUP1, BPRIME
WHERE (TENKTUP1.unique1 = BPRIME.unique1)

ただし、バケット数を減少させていくと、バケットのディスクへの書き込みが間に合わなくなる。バケット数が2および4の場合では、ハンドシェイクを使用しないと、バケット分割フェーズでのバケット作成担当のプロセッサにおいて、メッセージ用バッファの溢れの減少が発生する。

また、ハンドシェイクを使用する場合の結合処理においては、結果リレーションの書き込みを1プロセッサで行っている。従って、その時点でのディスク資源の競合はハンドシェイクを使用しない場合よりも少いことが予想され、これが処理時間に影響しているものと考えられる。バケット数が32の時にタプル数を10倍にして測定したところ、結合処理の経過時間は、ハンドシェイクを使用しない場合で22.0sec、ハンドシェイクを使用する場合で25.1secであった。

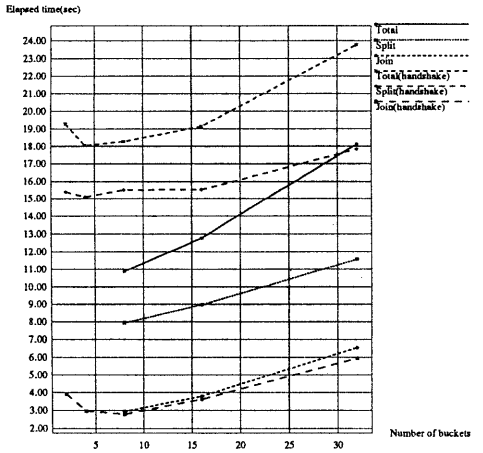

図 6: nsdisk 利用時のバケット数の影響 (1,000 × 10,000)

4.2 独立にディスクアクセスを行う方式

ディスクの利用台数を変化させた場合の結果(全体の経過時間)を図7に、ディスクの使用割り当てを表1に示す。バケット数=22、R=1,000、S=10,000である。

表 1: 独立アクセスのディスクの割り当て

ソース・結果 $[R/S/T_k]$	中間 $[BR_m/BS_m]$	利用ディスク数 (合計)
1	7	8
1	14	15
2	6	8
2	12	14
2	14	16

この結果では、バケット用のディスクを2倍にした場合、約1割ほど性能の改善が見られる。一般的に考えると、ディスクシークの時間が半減されているので、もっと多くの性能向上が期待できるのだが、そうはなっていない。これは、同一のI/Oプロセッサにディジーチェーンで接続されているディスク構成が原因だと考えられる。

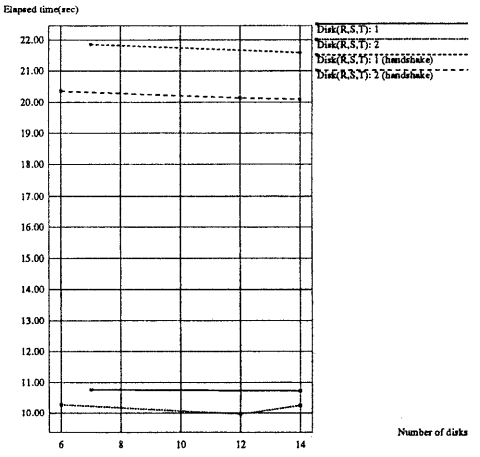

図 7: 独立アクセス時の R,S,T用ディスク数の影響 (1,000 × 10,000)

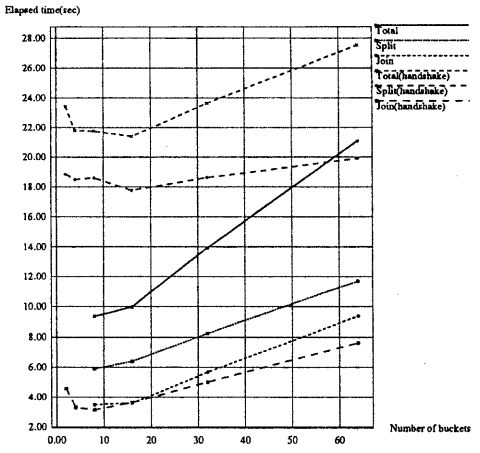

図 8: 独立アクセス時のバケット数の影響 (1,000 × 10,000)

つまり、単一のディスクにおいてシーク時間が減少したとしても、I/Oプロセッサまたは I/O バスの競合が頻繁に発生すると、ディスク並列化の効果が期待できず、資源を有

効に利用できない。これは、ハンドシェイクを使用しない場合に 2-12 台から 2-14 台にバケット用のディスクを 2 台増やした時、結合処理フェーズにおいて上記の競合が発生して性能が低下しているという結果からもうかがうことができる。

また、nsdisk を使用した場合の結果 (図 5) と比較して、ハンドシェイクを使用しない場合の性能が良好であることが判る。これによって nsdisk の使用に伴うコストがかなり大きいこと、現在の nCUBE2 のシステムではその最適化が十分行われていないことが推察できる。従って、ディスクを並列に使用して性能の向上を図る場合には、媒体へのアクセスに対してどのような手段を用いるかが大きな問題となる。

なお、ハンドシェイクを使用しない場合にタプル数を 10 倍 (R=10,000、S=100,000) にして測定を試みたが、結果の書き込み担当のプロセッサを 10 個に増加してもメッセージ受信用のバッファが溢れるという現象が発生した。

次に、バケット数を変化させた場合の結果を図 8 に示す。ディスク利用台数は R,S,T=1、B=7 である。nsdisk を使用した方式の場合 (図 6) と同様に、ハンドシェイクの有無にかかわらず相対的にバケット数が増えると性能が悪化する。

4.3 バケットを分散しメモリ上に保持する方式

ここでは 3.3 節の実装方式に基づき、バケットのファイルをディスクに作成せずに、各ノードのメモリ上に保持する場合の性能についてベンチマークの測定を行った。

R=1,000、S=10,000 およびディスクの利用台数としてリレーションの読み出しに 1 台、結果の書き込みに 7 台の状態で測定したところ、ハンドシェイクを使用しない場合で 5.2sec、使用する場合でも 11.8sec となり、nsdisk を使用した方式 (図 5)、ディスクアクセスを独立させた方式 (図 7) と比較した場合、いずれも大幅な性能の向上が見られた。

表 2: 分散メモリ保持方式におけるディスク数の影響 (10,000 × 100,000)

(R/S)	(T)	Buckets	Elapsed (sec)	Read (sec)	Write (sec)
1	7	21	45.4	30.2	2.1
1	14	28	46.7	30.5	1.4
2	6	24	22.9	14.9	2.5
2	12	24	24.0	15.0	1.6
2	14	28	24.7	15.6	1.4
4	4	24	11.8	7.4	3.7
4	8	24	12.7	7.2	2.3
(A)8	8	24	9.8	5.6	2.4
(B)8	8	24	8.9	5.1	2.2
(C)8	8	24	10.0	6.0	2.6
8	16	32	10.0	5.8	1.3

次に、R=10,000、S=100,000 の結果[3]を表 2 に示す。この中でリレーションの読み出しに 8 台、結果の書き込みに 8 台のディスクを使用している形態では、その利用方法を 3 パターン設定した。

[3]この場合の Read および Write の時間は、処理を担当する各プロセッサにおいてそれぞれのシステムコールに費やされた時間を合計したものであり、ここでは 1 プロセッサ当たりの平均値について示してある。

パターン A 合計 8 台でリレーションの読み出しと結果の書き込み対象のディスクが同一。

パターン B 合計 16 台でリレーションの読み出しと結果の書き込み時に I/O プロセッサの競合はない。

パターン C 合計 16 台でリレーションの読み出しと結果の書き込み時に I/O プロセッサの競合がある。

この 3 パターンの状態を図 9 に示す。

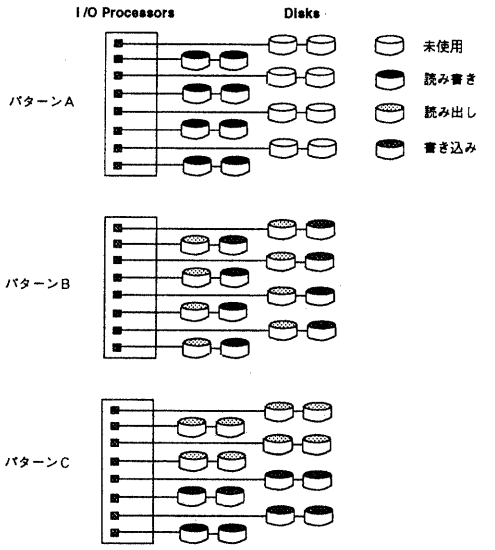

図 9: ディスクの利用形態

この結果では、ディスクの使用台数を単純に増加しただけでは性能が向上しないことが判る。特に、結果の書き込み用のディスク (T) を 2 倍にした場合を見ると、1 台当たりの書き込みに費やされる時間 (Write) は短縮されているのだが、全体の経過時間はほとんど変化していない。一方、読み出し用のディスク (R,S) を倍にした場合では、読み出しに費やされる時間 (Read) の短縮に伴い全体の経過時間も短くなっている。つまり、リレーションの読み出しにおけるファイルアクセスの速度が、全体の処理時間を抑制するという処理の I/O バウンドの現象が顕著に示されている。

また、読み出しと書き込みが 8 台の構成における 3 パターンの結果を比較した場合、I/O プロセッサの競合が発生しない、パターン B における性能が最も優れていることが判る。特に読み出しにおける I/O プロセッサ競合が、全体の経過時間にかなり影響を及ぼしていることがうかがえる。

以上のように、メッセージパッシングマシン上で結合演算を行う場合には、ディスクを単純に並列化しただけでは性能の向上が達成できるとは限らず、逆に性能低下を引き起こすこともあり得るので、ディスクの利用形態と台数には十分考慮しなければならない。

最後に、バケット数を変化させた場合の結果を図 10 に示す。R,S=300,000 およびディスク利用台数は R,S=8、T=8(パターンは A) である。なお、ハンドシェイクを使用しない時

の結果リレーションの書き込みについては、結合処理と同時に行う方法を用いると、書き込み担当のプロセッサにおいてメッセージバッファの溢れの現象が発生した。従って、その回避策として、書き込み担当のプロセッサにタプルデータが全て送られた後(結合処理が完全に終了した後)に実行するよう変更した。この場合、リレーションの読み出しと書き込みの処理においてディスク資源の競合は発生しないので、パターンAでも問題はないと言える。

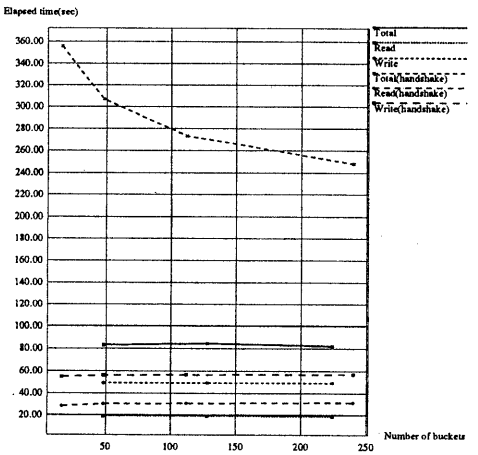

図 10: バケットを分散しメモリ上に保持する場合のバケット数の影響 (300,000 × 300,000)

ハンドシェイクを使用する場合では、バケット数の増加に伴い経過時間が短くなることが示されている。この場合、Read および Write に費やされる時間にはほとんど変化が見られない。これは、バケット数の増加に伴い結合処理担当のプロセッサも増え、読み出し担当のプロセッサの(結合処理担当のプロセッサに対する)データ送信の要求に対するACKの到着時間の間隔が短くなり、結果的にハンドシェイクに伴う待ち時間が短縮されたのではないかと推察できる。

また、ハンドシェイクを使用しない場合では、バケット数が増加しても経過時間に余り変化が見られない。これは、I/Oバウンドのため、バケット数の増加つまり結合処理担当のプロセッサ数の増加による処理コストの分散化が、それだけでは実行時間の短縮にはつながらないことを示唆している。

5 リレーションサイズとシステム構成に関する考察

バケットを分散しメモリ上に保持する場合の模式図を図11に示す。ここで、N_Rはソースリレーションの読み出し担当のプロセッサ数、N_Tは結果リレーションの書き込み担当のプロセッサ数、N_Iは結合処理担当のプロセッサ数である。そして、結合処理担当のプロセッサにおける処理速度をP、受信用バッファのサイズをm_i、書き込み担当のプロセッサにおける受信用バッファのサイズをm_tとする。また、プロセッサ間のバンド幅T_rは2.75MB/sec、予備実験として計測したディスクアクセスのバンド幅はRead(T_{DR})が765KB/sec、Write(T_{DW})が289KB/secであった。

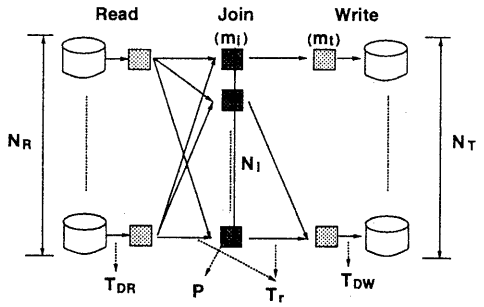

図 11: バケットをメモリ上に保持する場合の状態

ここで、リレーションRおよびSのサイズ(タプル数×208B)をそれぞれX_R、X_S、結合処理担当のプロセッサのメモリサイズをM_I、ハッシュテーブルのサイズをm_h、書き込み担当のプロセッサのメモリサイズをM_T、各プロセッサのメモリにおけるプログラム使用領域のサイズをm_pとすると、$M_I = m_i + m_h + m_p$、$M_T = m_t + m_p$および$m_h = \frac{X_R}{N_I}$となる。以上より、ハンドシェイクの必要性に関して次の関係を導くことができる。

$$\frac{(N_R \times T_{DR} - N_I \times P) \times (X_R + X_S)}{(N_R \times T_{DR})} \leq m_i \times N_I \qquad \text{(a)}$$

$$\frac{(N_I \times P - N_T \times T_{DW}) \times (X_R + X_S)}{(N_I \times P)} \leq m_i \times N_T \qquad \text{(b)}$$

この条件を満たしている間はメッセージ受信用バッファの容量が不足することはないので、ハンドシェイクの必要はない。特に$N_R \times T_{DR} < N_I \times P$且つ$N_I \times P < N_T \times T_{DW}$の場合には、リレーションサイズにかかわらずハンドシェイクを行う必要はない。

次に、各処理担当のプロセッサ数を調整することにより、ハンドシェイクを必要としないリレーションサイズの最大値を予測することが可能である。従って、この近似値を求めてみる。この時、m_p=2MBとし、結合処理担当プロセッサおよび書き込み担当プロセッサへのタプルデータの配分は、均等に行われることを前提とする。

1. 4.3節において、R,S=300,000の場合にメッセージ受信用バッファのオーバーフローが発生した同時書き込みのケースより、Pを算出する。この時、$N_I = 7$、$N_T = 7$、$M_T = 16$MB、$X_R + X_S = 124.8$MBであった。結果リレーションの書き込みにおいて発生したのでこれを(b)に代入し、この時点で臨界点に達したものと仮定する。計算の結果、$P \cong 1.38$MB/secとなる(対応する処理能力として)。
2. ハンドシェイクを行う可能性がある場合には、$N_R \times T_{DR} \geq N_I \times P$且つ$N_I \times P \geq N_T \times T_{DW}$であり、この時、$N_R + N_I + N_T \leq 256$である。従って、これらの条件を満たし、$N_T$を最大にするようなプロセッサの組合せを求めると、$N_R = 62$、$N_I = 34$、$N_T = 160$となる。
3. $M_I = m_i + \frac{X_R}{N_I} + m_p$に$N_I = 34$を代入する(この時$M_I = 4$MB)。計算の結果、$X_R = 68 - 34m_i$となるの

で、リレーション Rのサイズは $0 < X_R <$ 68MB ということになる。

4. 結合処理担当のプロセッサにおける受信用バッファm_iは、$0 < m_i <$ 2MB であることが導かれた。ここで m_i= 0.5MB の場合、X_R= 51MB となり、これを (a) に代入すると $X_S \leq$ 1.53GB、(b) に代入すると $X_S \leq$ 22.0GB となり、リレーション Sの最大サイズは 1.53GB となる。同様に、m_i= 1MB の場合には、X_R= 34MB、X_S= 3.13GB、m_i= 1.5MB の場合には、X_R= 17MB、X_S= 4.73GB となる。

従って、リレーションサイズは、Rが最大値の 68MB 未満の範囲において減少するのに伴い、Sは増加するということになる。例えば、Rが 3.4MB の時に Sは 6GB となるのでこの付近が Sの (計算上の) 最大値と考えられる。これらの最大値を超え、(a) の条件を満たせなくなった場合には、そのメモリをオーバーした分のバケットをディスクに書き込む必要が生じる。一方、(b) の条件を満たせなくなった場合には、無条件にハンドシェイクを行なわねばならない。

以上の様に、nCUBE2 での非同期通信に伴うメッセージのオーバーフローという問題を回避するために、ハンドシェイクを使用した同期通信での実装を試み、この同期通信が必要としている境界を求めるためのモデル化を行った。しかし、本来はシステム側でのフローコントロールによりオーバーフローを発生しないようなメカニズムが必要とされる。

6 おわりに

本報告では、GRACE ハッシュ方式およびハイブリッドハッシュ方式を、メッセージパッシング型並列計算機 nCUBE2 上に実装し、限られた資源の中で、ディスク、メモリの割り当てがその性能にどのように影響するか、実験を行った。そこでは、nsdisk を使用した方式、ディスクアクセスを独立させた方式、バケットを分散しメモリ上に保持する方式を使用した。また、非同期通信とハンドシェイクを使用した同期通信の処理時間を比較し、与えられた条件の下での非同期通信の限界について考察を行った。

この結果、バケットを分散しメモリ上に保持する方式で大幅な性能の向上が見られ、ディスク利用台数が最良の場合において R=10,000、S=100,000 では 8.9sec を、R,S=300,000 では 82sec(図 10) という結果を得た。しかし、この場合、ディスクを単純に並列化しただけでは期待通りに性能の向上を達成できないことも示された。そして、各処理担当のプロセッサ数を調整することにより、ハンドシェイクを必要としないリレーションサイズの最大値の予測が可能であることを示した。

次に、CPU およびディスク性能が異なっているので単純な比較はできないが、他のマシン上での性能についても参考に挙げておく。[Dew89] では、Gamma 上のハイブリッド・ハッシュジョインにおいて、利用メモリ容量が最も良い状態での R=10,000、S=100,000 のレスポンスが約 30sec となっている。また、[Kit92a] では、Symmetry S81 上の GRACE ハッシュジョインにおいて、最もスケーラブルなシステム構成での R,S=300,000 のリレーションサイズの処理時間が約 50sec となっている。

最後に、今後の課題について述べる。ここで測定対象となったタプルデータは、二つのソースリレーションとも均一な状態で配置されている。従って、不均一な状態で配置された場合の性能、特にバケットを分散しメモリ上に保持する方式の性能について測定を行い、その有効性を調べる必要がある。その上で、メッセージパッシングを通信スタイルとする分散メモリ並列マシン上での、効率の良い結合演算を実現するための方式を考えていかねばならない。

参考文献

[Dew84] D.J.Dewitt,*et al.*, "Implementation Techniques for Main Memory Database Systems," Proceedings of the 1984 SIGMOD Conference,Boston MA,June 1984.

[Dew85] D.J.Dewitt,R.Gerber, "Multiprocessor Hash-Based Join Algorithms," Proceedings of the 1985 VLDB Conference,Stockholm,Sweden,August,1985.

[Dew89] D.J.Dewitt, "A Performance Evaluation of Four Parallel Join Algorithms in a Shared-Nothing Multiprocessor Environment," Proceedings of the ACM SIGMOD,Portland,Oregon,June 1989.

[Gray91] J.Grey, "The Benchmark Handbook for Databese and Transaction Processing Systems," Morgan Kaufmann,1991.

[Kit83] M.Kitsuregawa,H.Tanaka,T.Motooka, "Application of Hash to Data Base Machine and Its Architecture," New Generation Computing,Vol.1,No.1,1983.

[Kit92a] M.Kitsuregawa,S.Tsudaka,M.Nakano, "Parallel GRACE Hash Join on Shared-Everything Multiprocessor: Implementation and Performance Evaluation on Symmetry S81," Proceedngs of the 8th International Conference on Data Engineering,pp.256-264,1992.

[Kit92b] M.Kitsuregawa,S.Hirano,M.Harada, *et al.*, "The Super Database Computer(SDC)System Architecture,Algorithm and Preliminary Evaluation," Proceedngs of the 25th Hawaii International Conference on System Sciences,pp.308-319,1992.

[LTS90] H.Lu,K.Tan and M.Shan, "Hash-Based Join Algorithms for Multiprocessor Computers with Shared Memory," Proceedngs of the 16th VLDB Conference Brisbane, Australia,1990.

[Omi91] E.R.Omiecinski, "Performance Analysis od a Load Balancing Hash-Join Algorithm for a Shared Memory Multiprocessor," Proceedngs of the 17th International Conference on Very Large Data Bases,Barcelona,September,1991.

[Shap86] L.D.Shapiro, "Join Processing in Databese Systems with Large Main Memories," ACM Transactions on Databese Systems,vol.11 No.3,pp.239-264, September,1986.

[Zha94] X.Zhang, "Parallelism of GRACE Hash-Join in the COMA Model:Implementation and Evaluation on the KSR-1 System," Thesis,University of Florida,July 1994.

[古野 94] 古野 真太郎, 横田 治夫, "nCUBE2 上への並列ハッシュジョインアルゴリズムの実装," JAIST,Reserch Reports IS-RR-94-0010A,1994.