1R−1

分散共有メモリ計算機における並列ハッシュ結合演算方式の実装

今井 洋臣, 中野 美由紀, 喜連川 優
東京大学生産技術研究所

1 はじめに

分散共有メモリ型並列計算機において，アプリケーションの実装を共有メモリ型並列計算機と同様に行うと，[1] にあるようにノード間をまたがるメモリアクセスによりノード間の通信を増大させ性能が低下する可能性がある．従って，分散共有メモリ型並列計算機上でのアプリケーションの実装では，各ノードに分散している物理的なメモリの配置とメモリアクセスの局所性を意識することで性能が改善されることが期待できる．本稿では分散共有メモリマシン上での関係データベース処理実装方式を検討するために，並列グレースハッシュ結合演算を商用マシン (HP Exemplar SPP-1000) 上に実装し，分散共有メモリマシン上のメモリアクセスの局所性を考慮したバッファ管理実装方式の検討を行う．

2 Exemplar SPP1000

Exemplar SPP1000 は分散共有メモリ型計算機である．8CPU を密結合してノードを構成し，ノード間をネットワークで疎結合している．ノード内は共有メモリ，ノード間は分散共有メモリである．

ノード上のメモリは大きく次の 3 つの領域に分類される．

1. ノード間にまたがるグローバルメモリ
2. 他のノードからも参照可能なローカルメモリ
3. 他ノードのページをキャッシュするのに使われる CTI キャッシュ (ネットワークキャッシュ)

本研究室のマシンは 4 ノード構成で，32CPU と 2GB のメモリを持っている (8CPU+512MB/node)．

3 並列ハッシュ結合演算実装方式

関係データベース処理の結合演算処理は非常に負荷の高い処理であり，並列処理による高速化の研究が多く行われている．今回の実装では並列 Grace ハッシュ結合演算を対象として単純な共有メモリ型実装方式 (以下 SE 方式) とメモリの物理的な分散を意識しアクセスの局所性を考慮した実装方式 (以下 SN 方式) の 2 つの方式を実装した．並列 Grace ハッシュ結合演算処理はリレーション R からハッシュテーブルを生成するビルドフェーズ及びそのハッシュテーブルを走査しリレーション S との結合処理を行うプローブフェーズからなる．本実装ではリレーションを読み出すリードプロセス，データバッファ内のタプルをハッシュテーブルエントリに登録するビルドプロセス，ハッシュテーブルを走査するプローブプロセスの三つのプロセスから構成される．

ビルドフェーズ　各ノードの異なるプロセッサ上でリードプロセスとビルドプロセスが並列に動作する．

SE 方式ではメモリのローカリティは意識しない．従ってハッシュテーブルエントリ領域，データバッファ領域共にグローバルメモリ上に獲得し，データバッファ領域はページ単位にリスト形式で管理する．各ノード上のリードプロセスはリレーションをデータバッファに書き込む．各ノード上のビルドプロセスは書き込まれたデータバッファを一ページずつ取り込み，各タプルごとにハッシュテーブルエントリに登録する．

SN 方式ではハッシュテーブルエントリ領域データバッファ領域，共にノード毎に他ノードから参照可能なローカルメモリ上に獲得し，ページ単位にリスト形式で管理する．SN 方式ではノード毎にハッシュテーブルが分割されるため，リードプロセスではリレーションを自ノード内のデータバッファに書き込む時点で該当するハッシュテーブルを持つノード毎にバッファを振り分ける．バッファは一杯になると各ノードのバッファリストにつながれる．ビルドプロセスは各ノード内のバッファリストにあるデータを取り込み，ローカルメモリ上のハッシュテーブルに格納する．

プローブフェーズ　各ノード上の異なるプロセッサ上でリードプロセスとプローブプロセスが並列に動作する。ビルドフェーズと同様にプローブリレーションを読み込むバッファを SE 方式ではグローバルメモリ上に獲得し、SN 方式では各ノード上のメモリ上に獲得する．しかし，ビルドフェーズと異なり，ハッシュテーブルを走査し結合処理を行った後はプローブリレーションを保持する必要はないため，数ページ分のバッファしか用意せず，そのバッファを使い回ししてプロセス間のデータ通信を行う．

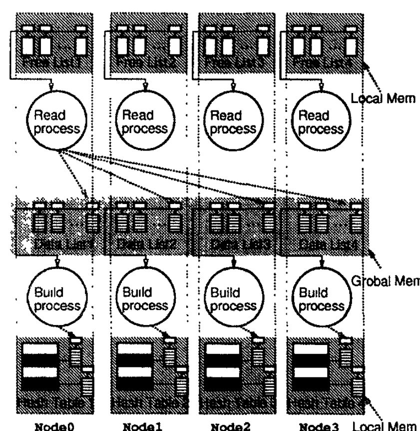

図 1: SN 方式におけるビルドフェーズ

4 実装結果と考察

4.1 測定条件と全体の処理時間

SE, SN 方式におけるデータサイズを変化させた場合のビルドフェーズ，プローブフェーズの合計処理時間の結果を図 2,3 に示す．本測定では 4 ノードを用い，データサイズを 8MB ～ 640MB まで変化させた．ここではメモリアクセスストラテジーの差を確認することが目的であるため，ディスクアクセス時間は除いている．またノード間のメモリア

Implementation of Parallel Hashjoin in Distributed Shared Memory Archtecture
Hiroomi IMAI, Miyuki NAKANAO and Masaru KITSUREGAWA
Institute of Industrial Science, University of Tokyo
roppongi 7-22-1, Minato-ku, Tokyo, 106 JAPAN

クセスにはメモリコヒーレンスを保つために CTI ディレクトリ情報を作成しなければならないが，その影響を軽減するため warm_start で測定を行う．全体の処理時間から SN 方式は SE 方式と比較して 78% 以上の性能向上がみられ，メモリの物理的な分散を意識してプログラミングすることは効果があるといえる．以下ではビルド / プローブの各フェーズについて述べる．

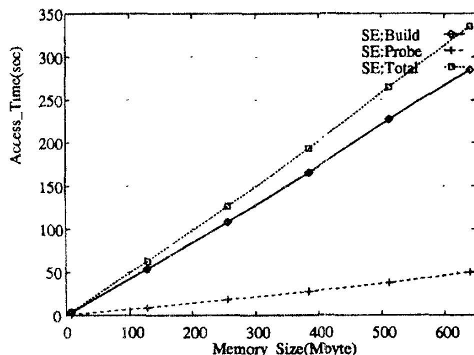

図 2: SE 方式　処理時間 (4 ノード)

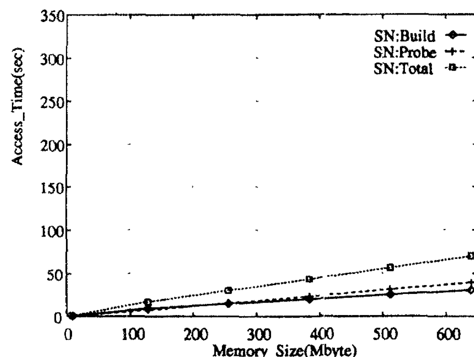

図 3: SN 方式　処理時間 (4 ノード)

4.2　ビルドフェーズ処理時間

ビルドフェーズの実行時間に関しては SN 方式が SE 方式より約 88% 程度の性能向上がみられる．

SE 方式のリードプロセスはどのノード上のデータ領域かを意識せずにタプルを書き込むため，負荷の高い他ノード上のデータ領域への書き込みが生じる．そのため SN 方式のそれに比較して性能が悪くなる．ビルドプロセスはリードプロセスが処理したデータを待って処理するので，この性能差はビルドプロセスに反映される．

SN 方式のビルドプロセスはテーブルを作るため他ノード上のデータ領域を参照しなければならないがリードプロセスとオーバーラップするため影響は小さい．SE 方式のビルドプロセスは他ノードのデータ領域の参照に加え，グローバルなハッシュテーブルを作成するために他ノードのデータ領域へロックを取りながら書き込みを行うため性能が著しく劣化する．一方の SN 方式ではハッシュテーブルはローカルに確保されている．ハッシュテーブルに関してはローカルに書き込まれるため，ロックは必要ない．

4.3　プローブフェーズ処理時間

プローブフェーズの実行時間に関しては SN，SE の差はそれほど無い．

プローブフェーズではリスト形式で管理された数枚のページでプロセス間のデータ通信を行っているが，それらのページは使い回しされるため常に他ノード上の CTI キャッシュに取り込まれている．リードプロセスは読みだしたリレーションを自ノード上のデータ領域へ書き込むが，その領域が他ノード上のの CTI キャッシュに取り込まれているため invalidatoin を引き起こし処理時間のネックになる．このためプローブフェーズの結果はプローブプロセスによるハッシュテーブル及びボディに対するメモリアクセスコストを反映しているわけではない．

今回はページの使い回しによる invalidation のネックを避ける 2 つの方式を実装てみた．両方式ともノード間でのページの使い回しを行うコストを他ノード上のデータ領域を参照するコストに置き換えている．一つの方式はリードプロセス - プローブプロセス間のデータ通信をノード内に限定し，プローブプロセスが該当する他ノード上のハッシュテーブル及びボディを参照する方式である (図 4 SN_b)．ハッシュテーブルはビルドフェーズでローカルに作成されるのだが，他ノードからも参照可能なためこの方式でノード間のページの使い回しを回避できる．もう一つの方式はメモリのコヒーレンスを保たないような読み捨てのデータ領域を用意し，そのデータ領域を用いてノード間の通信を行い invalidation のコストを削減する方式である (図 4 SN_a)．同じページに書き込みを行うものの invalidation のコストはかからない．(図 4 に示したものはメモリの空きを利用して仮想的に読み捨てのデータ領域があった場合のものである) 前者はアプリケーションに特化した方式だがこのようなプロセス間のまとまった大きさのデータ通信には後者の方が汎用性がある．

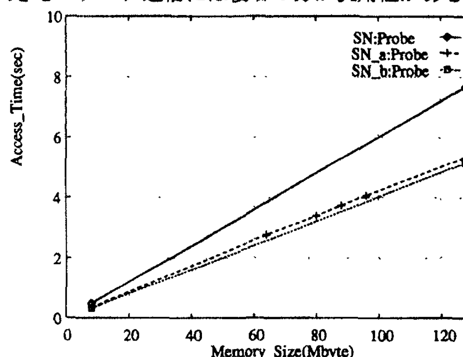

図 4: SN・SN_a・SN_b プローブフェーズ処理時間

5　終わりに

本報告では分散共有メモリ計算機上での並列データベース処理を単純に共有メモリ型方式とメモリの分散を意識しアクセスの局所性を高めた方式とで実装し，メモリの分散を意識したプログラミングの有効性を示した．分散共有メモリ並列計算機はデータベース処理において有用である．しかしながら，単純にすべてのメモリをキャッシュにマッピングするのでは invalidation を多発させる可能性がある．アプリケーションの用途に応じてメモリの参照を工夫することで，性能向上が得られる．

参考文献

[1] A.Shardal and J.F.Naughton: *Using Shared Virtual Memory for Parallel Join Processing*, Proc. of SIGMOD '93, pp.119-128, 1993