共有メモリ型マルチプロセッサによる並列ハッシュ結合演算処理とその評価

喜連川 優† 津高 新一郎† 中野 美由紀†

関係データベースの各種の処理の中で特に2つのリレーションを動的に結合する結合演算は処理負荷が重く，数々のアルゴリズムが開発されてきたが，中でもハッシュ法に基づくアルゴリズムが現時点で最も有望と考えられている．また近年，並列計算機システムが一般化しつつあり，並列アーキテクチャの採用による並列関係データベースシステムの性能向上が注目されている．このような背景から，我々はハッシュに基づく結合演算方式として GRACE ハッシュ結合演算技法を取り上げ，シンメトリS81共有メモリ型マルチプロセッサ上に実装を行い，結合演算の性能評価を詳細に行い，理想的な並列台数効果を確認するとともに，その有効性を明らかにした．本論文では GRACE ハッシュ結合演算技法の並列化手法について検討するとともに，その実装化について考察し，更に実装システムの性能評価により，関係データベースの共有メモリ型並列プロセッサによる性能向上の有効性について明確化する．

Parallel Hash-Based Join Processing on Shared-Memory Multiprocessor and Its Performance Evaluation

Masaru Kitsuregawa,† Shinichiro Tsudaka† and Miyuki Nakano†

Join is one of the most time-consuming operation in relational database systems. In order to accelerate its performance, several algorithms have been proposed. At present, hash-based join algorithm is regarded as the best approach in comparison with nested loop join and sort merge join. Recently, most of the computer systems such as mainframes and workstations tend to adopt the shared-memory multiprocessor architecture. This initiates the research on parallel execution of relational operation on shared-memory parallel machines. We implemented the GRACE hash join on Symmetry S81 multiprocessor system consisting 18 processors and 8 disk drives. This paper discusses the implementation details on the parallelization of GRACE hash algorithm and shows the performance evaluation results. We could attain almost linear performance increase by scaling up the system. It was shown that shared-memory parallel processor and the proposed parallel GRACE hash join method are very effective to increase the performance of relational database.

1. はじめに

1970年代 E. F. Codd によって提案された関係データベースは，強固な論理基盤，非手続き的なユーザインタフェースなど多くの特長を有する反面，処理負荷は重く，その商用化には10年以上の歳月を要した．関係データベースシステム実用化の最も大きな課題の一つはその性能向上にあり，関係演算の高速化に関して多くの研究がなされてきた．とりわけ2つのリレーションの動的な結合を可能とする結合演算は，単純な手法では両リレーションのタプルの積に比例した処理時間が必要となるため最も処理負荷が重く，その性能向上を目的として種々のアルゴリズムが開発されてきた．

ネストループ技法はアルゴリズムが簡単であり，小規模なリレーションに対しては中間リレーションを生成しないため最も高速な手法であるが，大規模なリレーションに対しては負荷が過大となり適用できない．これに対しソートマージ技法が用いられたが，ソート自体負荷が重いためより高速化が望まれてきた．80年代に入りハッシュによる結合演算技法が開発され，ほぼタプル数に比例した処理時間で結合演算が可能となった[2),8),9)]．このハッシュによる結合演算技法は現在最も高速な結合アルゴリズムであると考えられている[2),5)]．

一方，近年，単一プロセッサの性能限界から，複数個のプロセッサを用いた並列プロセッサの実用化が進み，メインフレームやワークステーションのサーバ機

† 東京大学生産技術研究所
Institute of Industrial Science, University of Tokyo

では，マルチプロセッサアーキテクチャを採用する事例が増えつつある．上述のごとく，関係データベース処理は負荷が重く，並列マシンによる性能向上の可能性が模索されている．本論文では現行のメインフレームやワークステーションに見られる共有メモリマルチプロセッサ上での関係データベース，とくに結合演算の並列処理技法について検討するとともに商用マルチプロセッサであるシンメトリ S81 上に実装し，評価を試みる．

従来の多くの研究は，シミュレーションに基づくものであり，実際に実装された例は少ない．ウィスコンシン大学ではトークンリングで結合した疎結合マシン上での並列処理に関し種々の実験を行っている[3),4)]が，共有メモリマルチプロセッサに関しては研究がなされていない．日本では，清木らがワークステーションの複合体や，共有メモリマルチプロセッサなど一般化された並列システム上での並列データベース処理に関し実験を行っている[16),17)]が，結合演算はネストループ技法を用いておりハッシュ結合演算技法の評価は行われていない．また，NTT のグループでは，データベースマシン RINDA の開発と共に筆者らと同様に共有メモリマルチプロセッサ上で関係データベースの並列処理に関する研究を行っており[14),15)]，適応的なページサイズの変更による負荷分散の効率化を試みているが，測定はディスクの入出力は考慮せずすべてのリレーションが主記憶に常駐されると仮定しており，想定している環境が異なる．なお，主記憶上の結合演算の並列処理に関しては著者らも既に性能評価を行っている[19)]．

Lu らは，共有メモリマルチプロセッサの上でのハッシュ結合演算技法の性能をシミュレーションによって評価することを試みている[6)]．従来の単一プロセッサ環境での評価では，排他制御が必要となるハッシュテーブルや出力バッファへの書き込みに関し競合を考慮しておらず不十分とし，競合確率を算出し，より正確な評価を行うとともにハイブリッドハッシュの改良版を提案している．彼らの主張する競合が最も顕著に表れるのはスプリットフェイズにおけるバケットバッファであり，例えばリレーションが小規模で分割数を2とするような時に，ディスクを6台駆動しプロセッサを10台以上駆動させると問題が出るとしている．我々の実装では出力バッファをディスクごとに設け分散化しているため競合が減っており，またタプルの移動全体をクリティカルセクションとせずアドレスポインタの変更のみを排他制御の対象とすることからも競合は減少しており，Lu らの論文が取り上げる問題は実装上の工夫で十分回避できると考えられる．この他 Lu らはディスクの台数を一定とし，プロセッサの台数を変えるシミュレーションを行っているが，測定範囲はほとんど CPU バウンドな環境下であるため，その結果は当然のものであり興味深いとは言えない．データベース処理では主記憶に展開されたデータに対する複数プロセッサによる並列処理と同時に入出力の並列化による高速化が重要と考えられる．我々は与えられた入出力性能の下でどの程度の並列性が内在するか明らかにした後に入出力を含めたスケーラビリティ，並列台数効果を明らかにしており，視点が大きく異なっている[13),18)]．すなわち，入出力デバイスと CPU の双方とも拡大可能な計算機環境において両者の負荷を均衡させた上でそれぞれの台数効果の確認を試みている．

以上のごとく，近年ワークステーション等で広く採用されつつある共有メモリマルチプロセッサ上での並列ハッシュ結合演算技法の実装ならびにその評価に関する研究は未だ十分なされていない．本論文では共有メモリマルチプロセッサに適したハッシュ結合演算技法の並列化を行うとともに商用マルチプロセッサに実装し，詳細な評価を行うことによりその有効性を明らかにする．

以下，第2章で従来提案されてきた種々のハッシュ結合演算技法を紹介した後，第3章において本稿で使用する商用共有メモリマルチプロセッサ，シンメトリ S81 の構成を紹介するとともに第2章で述べた GRACE ハッシュ結合演算技法の並列化ならびにその実装方式について考察する．第4章では，シンメトリ S81 上に実装した実験システムを種々の側面から評価する．すなわち，結合演算の並列処理効果について詳細な測定を行うとともに，ほぼ理想的な台数効果が得られることを示す．第5章は結論であり，今後の課題について触れる．

2. ハッシュに基づく結合演算技法

2.1 ハッシュに基づく種々の結合演算技法

ハッシュによる結合演算手法は，前節に述べたように，多くの場合最も高速な手法と考えられており，その最も基本的な手法が GRACE ハッシュ結合演算技法[9)]である．これは対象とする2つのリレーションの結合属性に同一のハッシュ関数を施していくつかのバ

ケットに分割し，その後に同じハッシュ値を有するバケット同士でつき合わせ処理を行うことにより結合演算結果を得る手法である．異なるバケット間では結合される可能性がないことに着目し，ハッシュを施しリレーションを分割するという前処理を導入することにより，処理負荷を大幅に低減することが可能となっている．

ハイブリッドハッシュ結合演算技法[2]は単純なシンプルハッシュ法と GRAEC ハッシュ法を融合させた手法であり，その名称もこの事実に依っている．GRACE ハッシュ結合演算技法では前処理としてのハッシュ分割時には主記憶を入出力バッファとしてしか使用していないため，その利用効率が低い．ハイブリッドハッシュ法ではこの点に着目し，1つ目のバケットをハッシュ分割時の余った主記憶空間を利用して処理することにより，主記憶に比べてそれほど大きくない比較的小規模なリレーションに対する処理性能の向上を図っている．

このような改良を行ったとしても，ハッシュ関数の分割のゆらぎを常に避けることは不可能であり，ハイブリッドハッシュ結合演算技法において必ずしも1つ目のバケットが残った主記憶空間を十分に利用するという保証はない．動的 GRACE ハッシュ結合演算技法[10]は主記憶に残すバケットを動的に選択する適応的な手法であり，不均一なデータ分布に対しても安定な性能を得ることを目的として改良されている．更に最近ではハッシュ技法の並列化に伴うスキューの取り扱いを含めたアルゴリズムの改良が種々試みられている[7],[11]．このようにハッシュ結合法には種々の変形が存在するが，ここでは簡単のため，基本型である GRACE ハッシュ結合演算技法に関し共有メモリマルチプロセッサ上での並列化を試みる．

2.2 GRACE ハッシュ結合演算技法

本節では，基本となる GRACE ハッシュ結合演算技法の処理の流れに関して簡単に述べる．本ハッシュ法では2種類のハッシュ関数（スプリット関数，ハッシュ関数）が用いられる．前者はリレーションを主記憶より小さな複数のバケットに分けるために，後者は主記憶上でタプルのつき合わせ処理を行うために用いられる．結合演算の対象となるリレーションをR，S，結果リレーションをTとし，Rの大きさはS以下とする．また，スプリット関数は1からNまでの整数，ハッシュ関数は1からMまでの整数を関数値として返すとする．このとき，GRACE ハッシュ結合演算技法は次のような2つのフェイズからなる．

スプリットフェイズ

本フェイズでは1つのリードバッファとN個のライトバッファを主記憶上に割り当てる．まずRを1ページずつリードバッファに読み込み，タプルの結合属性に対しスプリット関数を適用する．この時の関数値をiとすると，i番目のライトバッファにそのタプルを転送する．ライトバッファが一杯になると内容をバケット Ri $(1 \leq i \leq N)$ としてディスクに書き出す．こうしてRをN個のバケットに分割し，Sについても同様にN個のバケット Si に分割する．

ジョインフェイズ

本フェイズではリレーションRおよびS共用のリードバッファ1つと結果リレーションT用のライトバッファ1つを主記憶上に割り当てる．残りの領域はM個のエントリを持つリレーションRのバケット用のハッシュテーブルとして利用する．ジョインフェイズではリレーションRのバケット Ri に対し以下のビルドサブフェイズを実行した後リレーションSのバケット Si に対しプローブ

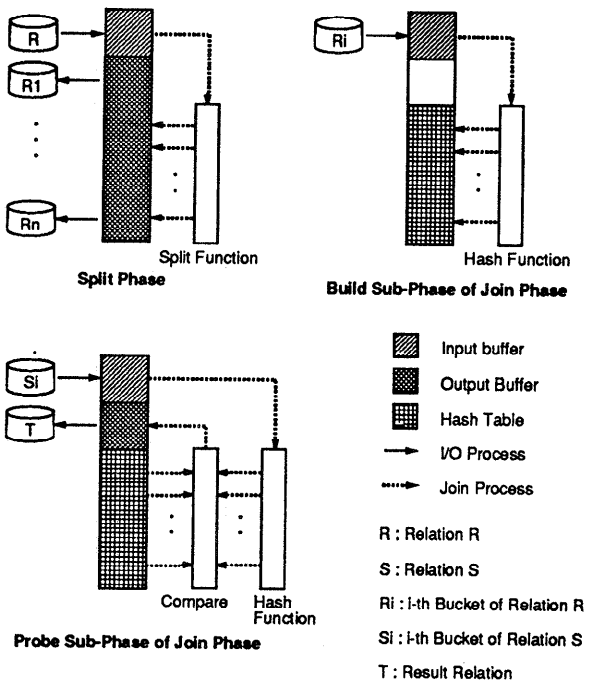


図 1 GRACE ハッシュアルゴリズム
Fig. 1 GRACE hash algorithm.

サブフェイズを実行する．これをすべてのバケットに適用する．

ビルドサブフェイズ

まず，Ri バケットを1ページずつリードバッファに読み込み，各タプルの結合属性に対しハッシュ関数を適用する．この時の関数値をmとすると，ハッシュテーブルのm番目のエントリにそのタプルを追加する．バケット Ri のすべてのタプルに対して上記の処理を繰り返す．

プローブサブフェイズ

Si バケットを1ページずつリードバッファに読み込み，各タプルの結合属性に対しハッシュ関数を適用する．この時の関数値をmとすると，ハッシュテーブルのm番目のエントリを検索し，同じ結果属性を持つタプルがあれば結果タプルを生成してライトバッファに転送する．ライトバッファが一杯になると内容を結果リレーションTとしてディスクに書き出す．バケット Si のすべてのタプルに対して上記の処理を繰り返す．

GRACE ハッシュ結合演算技法の各フェイズにおけるデータの流れを**図1**に示す．

3. 共有メモリマルチプロセッサへの実装

3.1 実験環境

ワークステーションからメインフレームに至るまで近年マルチプロセッサ化が進んでおり，このような環境でデータベース処理がどの程度高速化可能であるかを明確にすることが本論文の目的であるが，現行の商用ワークステーションでは未だ並列度は低く設定されており，接続可能なプロセッサの数も10台程度と限定されている．一方，近年のマルチプロセッサにおけるキャッシュコヒーレントプロトコルに関する研究の進展により，より多数のプロセッサを共有バスに接続することが可能になりつつある．このような背景から，シーケント社シンメトリ S81 を用いて，並列処理による性能向上に関する評価を試みることにした．シンメトリは i80386 (16 MHz) を用いたプロセッサボードを高速共有バス (80 MB/秒) を介して結合したアーキテクチャをとっており，**図2**に示されるように本評価においては18台のプロセッサ，4台のディスクコントローラ，8台のディスクからなるシステムを用いた．ここで DCC なるディスクコントローラは1つあたり2つのチャネルを有し2台のディスクを同時に駆動することが可能となっている．共有メモリは40 MB，ディスクは1台あたり約 600 MB の容量を有する．シンメトリシステムでは UNIX に並列処理機構を強化した DYNIX と呼ばれる OS が採用されており，ユーザによって生成されたプロセスは空いているプロセッサに動的に割り振られ並列に実行される．

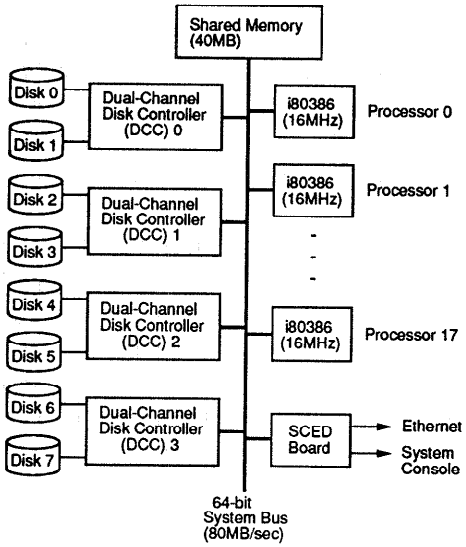


図2 本実験で使用したシンメトリ S81 のハードウェア構成
Fig. 2 Hardware configuration of symmetry S81 used in the experiment.

3.2 共有メモリマルチプロセッサに対する GRACE ハッシュ結合演算技法の並列化

上述の共有メモリマルチプロセッサ S81 に対する2章で示した GRACE ハッシュ結合演算技法の並列化について検討する．前章で述べたように GRACE ハッシュ結合演算技法はそのアルゴリズムの性質から容易に並列化できるが，単純な並列化を採用しても，Lu らの結果からもわかるように共有資源への不必要な排他制御による競合などがおきて，望ましい並列効果が得られない[6]．図1から分かるように，GRACE ハッシュ結合演算技法では，ディスクから読み出したデータを主記憶にロードし，主記憶上で必要な処理を行い，データをディスクに書き戻すという一連の流れの繰り返しであり，入出力動作と主記憶上の処理を分離して並列化を考慮することができる．従って，以下では GRACE ハッシュ結合演算技法の各フェーズおよび入出力動作について内在する並列性を十分に抽出し，理想的な並列効果が得られるよう並列処理方式を検討する．

(1) スプリットフェイズの並列化

スプリットフェイズでは，演算対象のリレーションをリードバッファに読み込み，タプルの結合属性に対しスプリット関数を適用し，関数値に応じたライトバッファにタプルを転送する．主記憶上のタプルに対するスプリット関数の適用処理は多数のプロセッサを用いて並列に実行することが可能である．このとき，タプルの転送は同一ライトバッファに対する同時書き込みを防ぐために排他制御が必要となるが，通常バケットの数はプロセッサの数に比べると多いため衝突は少なく，またタプルの転送時間を考慮するとタプル長が極端に短くない限り，ロックのオーバヘッドは問題にならない．

(2) ビルドサブフェイズの並列化

スプリットフェイズと同様に主記憶に読み込まれたタプルに対するハッシュ関数の適用処理は並列に実行することが可能である．このとき，タプルの転送は同一ハッシュエントリに対する同時書き込みを防ぐために排他制御が必要である．一般にハッシュエントリの数はプロセッサの数に比べ極めて多く，そのオーバヘッドは問題にならない．

(3) プローブサブフェイズの並列化

バケット S_i のタプルの結合属性に対するハッシュ関数の適用，ハッシュエントリの検索，結合演算等の処理は並列に実行することが可能である．このとき，結合演算結果のライトバッファへの転送は排他制御が必要であるが，実際には多数のハッシュエントリの検索に時間がかかり，また結合率は通常低いため衝突は少なく，並列に実行することが可能である．

(4) 入出力の並列化

前述のフェイズごとの考察で述べたように，複数プロセッサによる並列処理に加えて，複数ディスクによるデータの並列入出力により性能を大幅に向上させることが可能である．特にデータベース処理では実行時間の多くは2次記憶装置のアクセスで消費されており，複数ディスクの並列駆動は極めて有効と考えられる．ディスクの台数に比例して入出力の並列度を抽出するためには各ディスクのデータ量を均等化することが望ましいため，今回の実装では関係データベースシステムにおける基本要素であるタプルを単位としたソフトウェアによるストライピングの適用を試みることとする．これにより，データベースの各タプルはラウンドロビン手法でディスクごとに均等に配置され，各々のディスクは独立に入出力動作可能であるため，台数分の並列効果が期待できる．これは，デクラスタリングとも呼ばれ，GAMMA[4], XPRS等の実験システムでも採用されている．

3.3 実装方式

前節での検討から明らかなように，GRACE ハッシュ結合演算技法は多大な並列性を内在しており，適切な並列化を行うことにより大きな性能向上が期待できる．本節では GRACE ハッシュ結合演算の並列処理のための実装方式について検討する．

前節で述べたように，GRACE ハッシュ結合演算技法では入出力動作と主記憶上での処理を分離することができる．今回の実装ではそれぞれを2種類のプロセス，すなわち入出力プロセスならびにジョインプロセスで独立に行うこととした．入出力プロセスはディスクの読み書きを管理し，ジョインプロセスはタプルの比較や転送などその他の共有メモリ上での処理を行う．すなわち，入出力プロセスは図1において実線の矢印で示されているデータ転送を行い，ジョインプロセスは点線の矢印で示されている処理を行う．また，データベース処理においてその実行時間の大半は2次記憶装置へのアクセスに費やされており，ディスクをできる限り動作状態とするため，各ディスクにつき1つの入出力プロセスを生成し，各入出力プロセスは特定のディスクのアクセスのみを管理するものとする．

これら2つのプロセスは，主記憶上に設けられた共有ページキューを通してデータ転送，データ処理を行う．基本的には，ディスク上のすべてのデータを入出力プロセスがページ単位で共有ページキューを通して主記憶に転送し，ジョインプロセスは，データを読み込まれたページがある限り，所定の処理を行うこととなる．今回の実装では，主記憶上に構成される共有のフリーページ管理キューを用意し，入出力プロセスはディスクから1ページ分のデータをフリーページ上に読み込み，リードキューに追加する．ジョインプロセスはリードキューから1ページとり，各フェイズに応じた処理をタプルごとに行い，結果を同様にフリーページキューから獲得したライトバッファに書き込み，一杯になるとライトキューに追加する．各入出力プロセスは固有のライトキューを持っており，ジョインプロセスは各ライトキューへの出力データ量を管理する変数に従ってデータを分散し，ディスクごとの

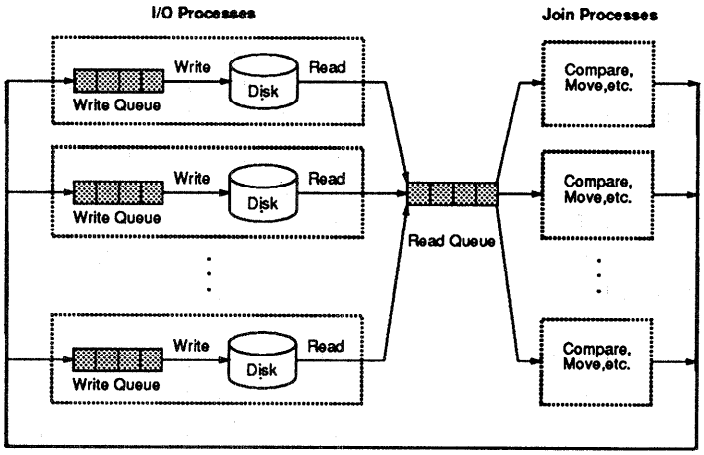


図 3 プロセス間のデータの流れ
Fig. 3 Data flow among processes via shared memory.

データ量が均等になるようにする．共有キューによるデータの通信の様子を図3に示す．

以上のように，本処理では，入出力プロセスにより生成されるページによって駆動される．従って，入出力処理の効率化が全体性能に与える影響が大きいことから，入出力動作をできる限り，中断しないようシステムを構築した．すなわち，共有メモリ上における共有キューを採用し，入出力時のバッファを各ディスクに固定せず，すべてをフリーページとしてキューからのポインタで管理している．これにより，各ディスクの読み込みはキューからのページ獲得の時のみ競合するだけで，入出力プロセス同士はほぼ独立に動作でき，十分並列効果が得られる．また，入力動作を優先的に行うために，出力バッファが一杯になった時，あるいは一連の動作の終了時のみ入力動作が中断され，出力動作に切り替えられる．一方，各ジョインプロセスはリードキュー上のページを処理後，優先的にフリーキューに登録する．

分散メモリマシンではメッセージパッシングにより実体を送受する必要があるのに対し共有メモリマルチプロセッサでは共有メモリ上のデータ移動を多くの場合ポインタの付け換えで実行することが可能であり，本論文での実装においてもデータの転送量を極力減らすよう工夫している．例えば，ビルドフェーズでは，リードバッファからハッシュエントリ領域へのデータ転送をせず，各タプルに対してポインタをはることで，リードバッファをそのままハッシュエントリ領域として用いている．これにより，ロック時間がタプル転送時間ではなくポインタの付け替え時間のみとなり，競合が避けられると同時に，タプルの転送がなくなることでジョインプロセスの負荷が減り，入出力プロセスが読み込むデータを遅滞なく処理することができる．

4. 並列 GRACE ハッシュ結合演算技法の性能評価

3.1節に示した構成のシンメトリS81マルチプロセッサ上に，3.3節で述べた方針に従い並列 GRACE ハッシュ結合演算技法を実装し，種々の側面から性能評価を行うことにより，共有メモリマルチプロセッサの並列関係データベース処理に対する有効性を明らかにする．

本章の性能評価に用いるリレーションのフォーマットはウィスコンシンベンチマーク[1]に基づいており，タプル長208バイト，結合演算の対象となるアトリビュートは4バイト整数であり，リレーションのカーディナリティを N とすると $0\sim(N-1)$ のユニークな値を有し，その出現順序はランダムに設定されている．対象リレーションの大きさは指定しない限り 30 万タプルとする．結合率は 100%，すなわち N タプルの2つのリレーションの結合結果は N タプルから構成され，各タプルは416バイトとする．演算対象となるリレーションは8台のディスクに対してタプル単位で均等に分配されている．また，ハッシュ結合演算途中で生成される中間リレーションも同様に生成される．また本評価では共有メモリとして 8MB を使用した．なお，以下の測定はすべて 10 回の実測値の平均値を示す．

以下，まずページサイズ，並列駆動ディスク数の変動に対する入出力性能を測定した後，スプリットフェイズ，ジョインフェイズの各フェイズの並列性を駆動プロセッサ数を変化させることにより測定する．最後に結合演算全体としての並列処理効果について評価する．

4.1 ページサイズと入出力性能

一般にページサイズはプログラムの参照の局所性を考慮し，適切な大きさが決定され，通常512バイト～4kバイトの数値が採用されることが多い．ここではプログラムではなくデータベースそのものを格納する際のページサイズについて検討することとする．最近

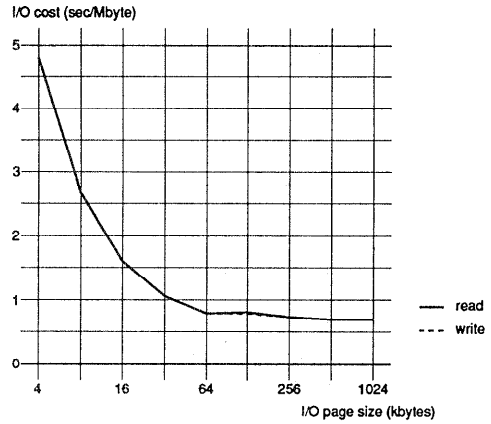


図 4 ページサイズと入出力性能
Fig. 4 Page size and I/O performance.

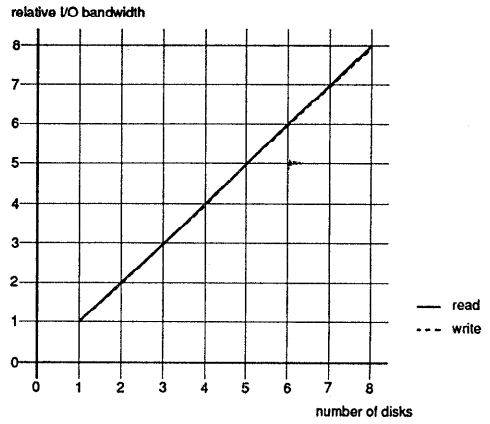


図 5 ディスクの並列駆動による入出力性能の向上
Fig. 5 Speed-Up of I/O by parallel disk access.

のメインフレームにおける拡張アーキテクチャでも同様にプログラムとデータのページサイズを別々に設定可能となっている.

関係データベースにおける問い合わせ処理では一般に連続したデータをアクセスすることが多い. インデックスの利用も考えられるが, 非クラスタリングインデックスの利用はその選択率がかなり小さい場合にしか効果を発揮せず, 連続アクセスの高速化は必須である. このような背景から, ここではページサイズを変えながら入出力時間を計測した. **図4** に1Mバイト当たりの読みだし時間を示す. 図から明らかなように, ページサイズの増大とともに入出力性能は向上するものの, 64kバイト程度で飽和し, それ以上の改善は少ない. この結果に基づき, 以下の計測においてはページサイズは 64k バイトと設定することとした.

4.2 並列入出力性能

前章において実験環境を示したが, シンメトリにおいて8台ものディスクを並列駆動するような事例は稀であることから, まず並列入出力に関する性能を確かめることとした. 同時に駆動するディスクの台数を変化させ, 入出力性能を測定した結果を**図5**に示す. 読み出し, 書き込みいずれの場合にもほぼ理想的な並列駆動効果が得られることがわかった. ただし, 実験の過程で入出力バッファの先頭アドレスによって入出力性能が変化する傾向が見受けられた. 入出力バッファバウンダリを変化させつつ8台の並列入出力性能を測定した所, バッファバウンダリが512バイト未満では, 書き込み性能が低下することがわかった. また読み出しの性能はバッファバウンダリにほとんど依存しない. この結果に基づき, 以下の実験において入出力バッファはすべてその先頭アドレスが512の倍数になるように設定することとした. なお, 図5の並列ディスクアクセスによる性能はこのようなバッファを用いて測定した.

4.3 スプリットフェイズの並列処理効果

3.2節では GRACE ハッシュ結合演算技法の並列処理について各フェイズに分けて検討した. ここではスプリットフェイズの並列処理効果について測定する. 動作させるディスクの数, すなわち入出力プロセスの数をパラメータとし, ジョインプロセスの数を変化させて, スプリットフェイズでの消費時間を測定した結果を**図6**に示す.

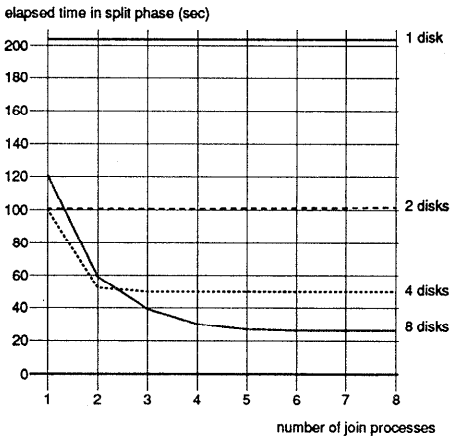


図 6 ジョインプロセス数とスプリットフェイズの実行時間
Fig. 6 Execution time of split phase vs. number of join processes.

ここでジョインプロセスの数とは正確にはジョインプロセスを実行するプロセッサの数を指している．またディスク駆動台数をパラメータとしているが，前節で述べたように各ディスクに対して1台のI/Oプロセス（プロセッサ）が割り当てられ，データのディスクからの読み出しおよび結果の書き込みを司っている．

1台または2台のディスク使用時はジョインプロセスの数は性能に影響を与えないことがわかる．しかし，4台または8台のディスクを使った時は，消費時間はジョインプロセスの数の増加に伴い減少し，次第に一定の値に漸近する．これはディスクからのデータの流れに十分追従できるだけのジョインプロセスが供給されている，すなわち処理が入出力バウンドになっていることを示している．最高性能を保つための最小のジョインプロセス数は，ディスク4台時で3，ディスク8台時で6である．このことから，スプリットフェイズにおいて，ディスク1台からのデータを遅れなく処理するには，入出力プロセス用に1台，ジョインプロセス用に 0.75 台で十分であることが分かる．

スプリットフェイズの並列台数効果については，紙面の都合上グラフをまとめて後節の図 12 に示すが，図から明らかなように理想的な台数効果が得られた．ここでデータベース処理は，主記憶上での処理を主体とする科学技術計算とは異なり，入出力負荷の占める割合が大きいことから，並列台数効果を示すグラフは横軸をプロセッサ数とするのではなく，ディスクとディスク1台当たりに必要なプロセッサを単位としている点に注意されたい．

実際の関係データベース処理では，結合演算だけが単独で行われることは少なく，以下の例のように選択演算を伴うことが多い．

```
select * from A, B
where A. key=B. key          (1)
and A. a<X                   (2)
and B. b>Y                   (2)
```

上の条件節（where 以下）において，(1)は結合演算を表す．その他の述語(2)の数は条件節の複雑さを表す．上記 SQL 文は選択演算と結合演算を分離して各々独立に処理することも可能であるが，通常の SQL コンパイラは結合演算のスプリットフェイズにおいて条件節を判定することで効率化を図っている．スプリットフェイズの消費時間はこの条件節の複雑さによって大きく変化する．

8台のディスク，8つの入出力プロセス，8つのジョインプロセスを駆動し，述語数を0から100まで変化させて，スプリットフェイズにおける消費時間を測定した．結果を**図7**に示す．条件節がごく簡単なときには消費時間は述語数によらずほとんど一定である．これは，ジョインプロセスの数が十分であるために，追加した条件節を含めた処理がディスクの速度に十分追従できるだけの速度で行われていることを示している．ジョインプロセスの数に余裕があることは図6から明らかである．一方，述語数が 20 を越えると，消費時間は条件節の複雑さにほぼ比例して増加していく．これはジョインプロセスの数に余裕がなくなり処理が CPU バウンドになったことを示している．

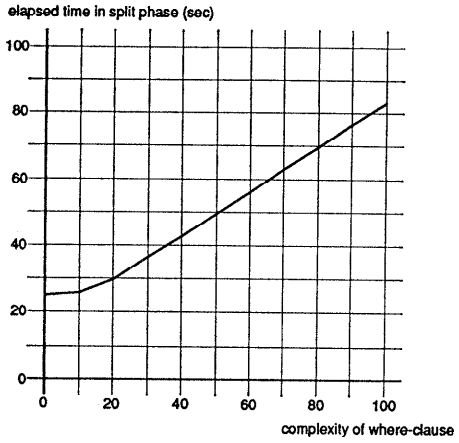


図 7 述語数とスプリットフェイズの実行時間
Fig. 7 Execution time of split phase vs. number of where-clause.

4.4 ジョインフェイズの並列処理効果

本節ではジョインフェイズの並列処理効果について述べる．スプリットフェイズの測定と同様に，動作させるディスクの数，すなわち入出力プロセスの数をパラメータとし，ジョインプロセスの数を変化させて，ジョインフェイズの消費時間を測定した（**図8**）．ディスク1台使用時はジョインプロセスの数は性能に影響を与えない．しかし，2台以上のディスクを駆動すると，消費時間はジョインプロセッサの数の増加に伴い減少し，次第に一定の値に漸近する．これは処理が入出力バウンドになったことを示している．最高性能を保つための最小のジョインプロセス数は，ディスク2台時で2，4台時で3，8台時で7である．このことから，ジョインフェイズではディスク1台からのデータを遅れなく処理するには入出力プロセス用に1台，ジョインプロセス用に1台，で十分であることが

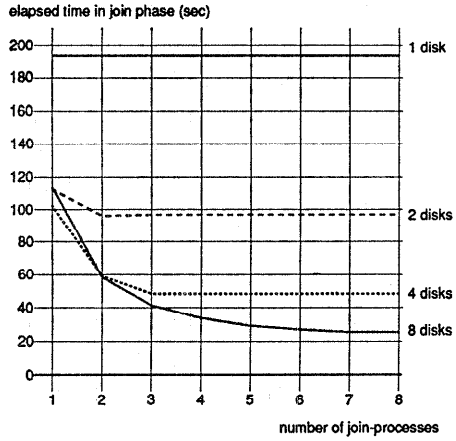


図 8 ジョインプロセス数とジョインフェイズの実行時間
Fig. 8 Execution time of join phase vs. number of join processes.

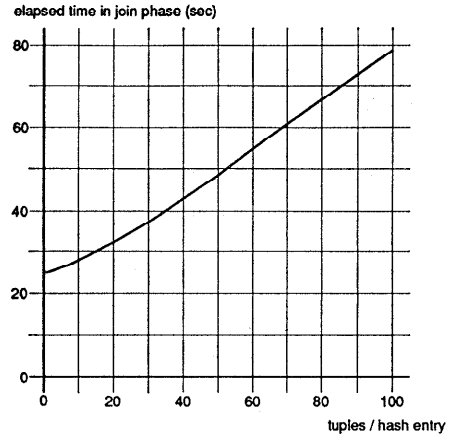


図 9 ハッシュテーブルのエントリ数とジョインフェイズの実行時間
Fig. 9 Execution time of join phase vs. number of tuples per hash table entry.

分かる.

ジョインフェイズにおいて性能に最も大きな影響を与える要因としてハッシュテーブルのエントリ構成が挙げられる. ハッシュエントリ数が多いほど, ビルドサブフェイズ時でのタプル挿入のロック競合が生ずる割合が減少し, またハッシュエントリが多いことは逆に1つのエントリ当たりのタプルリスト長が短くなり, プローブサブフェイズ時のサーチ時間が減少することになる. このようにハッシュテーブルのエントリ数を増加させることにより, メモリの消費は増大するが性能は向上すると考えられる.

8台のディスク, 8つの入出力プロセス, 8つのジョインプロセスを駆動し, 1ハッシュエントリ当たりのタプル数を変化させてジョインフェイズにおける消費時間を測定した (**図9**). ハッシュテーブルにおける1エントリ当たりの平均タプル数を横軸とする. 図7の結果と異なり, ハッシュエントリ当たりのタプル数にほぼ比例して消費時間が増加している. これは図8から分かるように, 8ディスク, 8ジョインプロセス時にはジョインプロセスの余裕がほとんどなく, そのためハッシュを粗くすることによる CPU 負荷の増加が直接性能に影響を与えるためと考えられる. このようにハッシュテーブルを細かくとることは性能向上に大きく寄与することが明らかであり, 他の測定ではハッシュエントリ当たりのタプル数は1と設定している.

ハッシュテーブルにおけるエントリの粗さと同様にスプリットフェイズで生成するバケットの数も性能を左右するパラメータとなる. ハッシュテーブルの場合はなるべく細かく分割することが望ましいという結果が導かれたが, バケットの分割に関しては, 極端にバケット数を多くしない限り, 分割数は性能にあまり影響を与えない. **図 10** に, 横軸をバケット数, 縦軸をジョインフェイズ実行時間とした評価結果を示す. なおこの測定に限り, 評価環境をタプル数 10 万件, 主記憶 16 M バイトと設定した. バケット数が5個から 50 個になっても, その性能はほとんど変わりがない. これは, 逆に見ると, 1個のバケットサイズが2万件でも2千件でも処理時間が変わらない, つまり主記憶量が小さくなっても処理時間が変わらないことを意味

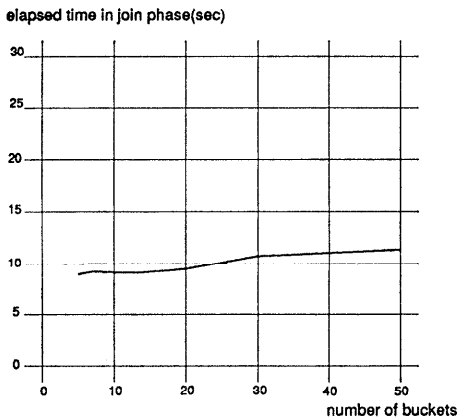


図 10 ジョインフェイズにおけるバケット数と実行時間
Fig. 10 Execution time of join phase vs. number of buckets.

している．つまり，GRACE ハッシュ結合演算技法の性能が主記憶容量にそれほど依存しない性質が並列化しても変らないことを示している．また，バケット数を大きくすることに生じるオーバヘッドは，バケット切替えによるものであるが，実験によりこれは十分に小さいことがわかる．この事実は，ハッシュ結合技法がマルチユーザ環境においても，良好な特性を示唆するものと言えよう．

4.5 GRACE ハッシュ結合演算技法の並列処理効果

ここでは前節までの入出力自体の性能と各フェイズにおける並列処理効果の評価に基づき結合演算全体の性能と並列処理効果について検討する．4.3, 4.4 と同様に動作させるディスクの数，すなわち入出力プロセスの数をパラメータとし，ジョインプロセスの数を変化させて，結合演算にかかる消費時間を測定した(**図 11**)．前節までの評価から明らかなように，スプリットフェイズに比べジョインフェイズの方が負荷が大きく，ハッシュジョイン全体の性能特性はほぼこれら2つを加えた形となっている．

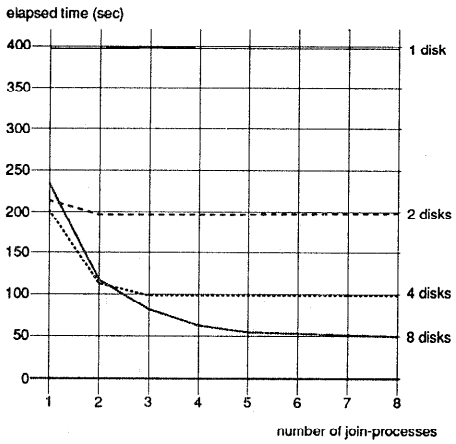


図 11 ジョインプロセス数と総実行時間
Fig. 11 Execution time of total join operation vs. number of join processes.

図 12 にシステムのスケーラビリティを示す．ここで横軸はディスク台数で正規化したシステム構成，すなわちディスク n 台のときには I/O プロセッサ n 台，ジョインプロセッサ n 台を駆動しており，この時の結合演算性能を縦軸としている．図から明らかなように，ディスク8台時の性能が若干低下しているものの（1台時の 7.7 倍の性能），本論文の実装手法によればほぼ理想的なスケーラビリティを達成できることが確認された．

従来のネストループ結合演算技法やシンプルハッシュ結合演算技法が両リレーションのカーディナリティの積に比例した処理時間を必要とするのに対し，GRACE ハッシュ結合演算技法は両リレーションのカーディナリティの和に比例した時間，すなわち線形時間で処理することが大きな特徴となっている．8ディスク，8入出力プロセス，8ジョインプロセス時に，リレーションの大きさを変化させて演算時間を測定した結果を**図 13** に示す．図から明らかなように，ほぼリレーションの大きさに比例した時間で演算が終了しており，本論文で示した並列化実装手法により我々の提案した並列 GRACE ハッシュ結合演算技法が理想的に実装されていることが分かる．

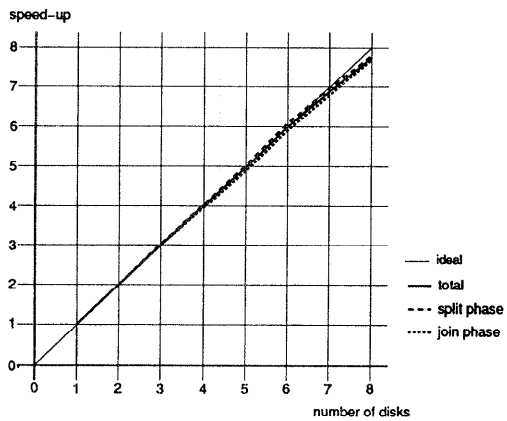


図 12 並列アクセスによる結合演算の性能向上
Fig. 12 Speed-Up ratio of join operation vs. number of activated disk drives.

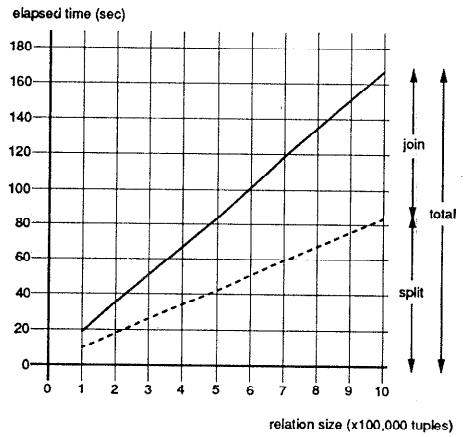


図 13 リレーションサイズと実行時間
Fig. 13 Normalized execution time of join operation vs. relation size.

5. おわりに

近年，ワークステーション等でバス結合型共有メモリマルチプロセッサアーキテクチャが広く採用されつつあることから，本論文では並列処理による関係データベース処理の高速化の可能性を明確化することを目的とし，最も負荷の重い結合演算を対象に，商用マルチプロセッサであるシンメトリ S81 に GRACE ハッシュ結合演算技法を実装すると共に評価を行い，ほぼ理想的な並列処理効果を確認することができた．すなわち，ハッシュ結合演算技法の並列化に関し考察し，共有メモリマルチプロセッサに適したプロセスモデルを明らかにするとともにシンメトリ S81 上に実装し，スプリットフェイズ，ジョインフェイズ各々に関し並列度を評価し，ディスクからの入出力に追従した処理が可能であることを示した．さらに，ディスク1台当たりに対しプロセッサを2台ずつ増加させることにより，システムを理想的にスケールアップ可能なことを明らかにし，共有メモリマルチプロセッサにより関係データベース処理を効率良く並列処理できることを示した．特にデータベース処理は入出力負荷が大きく，駆動ディスク台数を考慮に入れた並列処理効果の評価が不可欠である．しかしながら，筆者の知る限り，このような点に関する十分な性能評価は報告されていない．今回，ディスク8台，プロセッサ16台からなる比較的大規模な商用マシンの上で GRACE ハッシュ結合演算技法の実装とその性能評価を試みた．

本論文ではリレーションのデータ分布は一様であり，リレーションはほぼ等しい大きさのバケットに分割でき，バケットは一様なハッシュテーブルに展開可能であるとした．データの分布が不均一な場合の評価は今後の課題である．筆者らは不均一分布に対して有効な手法として GRACE ハッシュ結合演算技法の改良版である動的 GRACE ハッシュ結合演算技法を既に提案しており[10]，この技法の並列マシンへの実装ならびにその評価を今後進める予定である．

なお，共有マルチプロセッサでは単一バス上に結合可能なプロセッサ数に限界がある．これに対して我々は共有メモリパラダイムとメッセージパッシングパラダイムを融合したスーパーデータベースコンピュータ SDC を開発している[12]．

謝辞 本研究を遂行するに当たり，パナシーケント社中村氏ならびに平尾氏より多大な御協力ならびに技術的支援を賜り，感謝する次第である．

参考文献

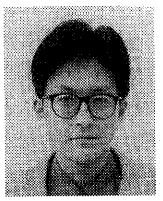
1) Bitton, D., DeWitt, D. J. and Tubyfill, C.: Benchmarking Database Systems—A Systematic Approach, *Proceedings of the International Conference on VLDB*, pp. 8-19 (1983).
2) DeWitt, D. J., Katz, R. H., Olken, F., Shapiro, L. D., Stonebraker, M. R. and Wood, D.: Implementation Techniques for Main Memory Database Systems, *Proceedings of the International Conference on SIGMOD*, pp. 1-8 (1984).
3) DeWitt, D. J. and Gerber, R.: Multiprocessor Hash-Based Join Algorithms, *Proceedings of the International Conference on VLDB*, pp. 151-164 (1985).
4) DeWitt, D. J., Gerber, R. H., Graefe, G., Heytens, M. L., Kumar, K. B. and Muralikrishma, M.: GAMMA—A High Performance Dataflow Database Machine, Computer Sciences Technical Report, 635 (1986).
5) DeWitt, D. J.: A Performance Evaluation of Four Parallel Join Algorithms in a Shared-Nothing Multiprocessor Environment, *Proceedings of the ACM SIGMOD*, Portland, Oregon (1989).
6) Lu Hongjun, Tan Kian-Lee, Shan Ming-Chien: Hash-Based Join Algorithms for Multiprocessor Computers with Shared Memory, *Proceedings of the International Conference on VLDB*, pp. 198-209 (1990).
7) Wolf, J., Dias, D. M., Yu, P. S. and Turek, J.: Comparative Performance of Parallel Join Algorithms, *Proceedings of International Conference of Parallel and Distributed Information Systems*, pp. 78-88 (1991).
8) Bratbergsengen, K.: Hashing Methods and Relational Algebra Operations, *Proceeding of the International Conference on VLDB*, pp. 323-333 (1984).
9) Kitsuregawa, M., Tanaka, H. and Moto-oka, T.: Application of Hash to Data Base Machine and Its Architecture, *New Generation Computing*, Vol. 1, No. 1, pp. 62-74 (1983).
10) Kitsuregawa, M., Nakayama, M. and Takagi, M.: The Effect of Bucket Size Tuning in the Dynamic Hybrid GRACE Hash Join Method, *Proceedings of the International Conference on VLDB*, pp. 257-266 (1989).
11) Kitsuregawa, M. and Ogawa, Y.: Bucket Spreading Parallel Hash: A New, Robust, Parallel Hash Join Method, *Proceedings of the International Conference on VLDB*, pp. 210-221 (1990).

12) Kitsuregawa, M., Hirano, S., Harada, M., Nakamura, M. and Takagi, M.: The Super Database Computer (SDC): System Architecture, Algorithm and Preliminary Evaluation, *Proceedings of the 25th Hawaii International Conference on System Sciences*, pp. 308-319 (1992).
13) Kitsuregawa, M., Tsudaka, S. and Nakano, M.: Parallel GRACE Hash Join on Shared-Everything Multiprocessor: Implementation and Performance Evaluation on Symmetry S 81, *Proceedings of the 8th International Conference on Data Engineering*, pp. 256-264 (1992).
14) Satoh, T., Hirano, Y., Honishi, T. and Inoue, U.: Design and Implementation of Parallel Database Processing on a Shared Memory Multiprocessor System, *Proceedings of International Workshop Future Database 92*, pp. 337-346 (1992).
15) Hirano, Y., Satoh, T., Inoue, U. and Teranaka, K.: Load Balancing Algorithm for Parallel Database Processing on Shared Memory Multiprocessors, *Proceedings of International Conference on Parallel and Distributed Information Systems*, pp. 210-217 (1991).
16) Kiyoki, Y., Hasegawa, R. and Amamiya, M.: A Stream-Oriented Parallel Processing Scheme for Relational Database Operations, *Proceedings of International Conference on Parallel Processing*, pp. 1013-1020 (1986).
17) Kiyoki, Y., Kurosawa, T., Kato, K. and Masuda, T.: The Software Architecture of a Parallel Processing System for Advanced Database Applications, *Proceedings of 7th IEEE International Conference on Data Engineering*, pp. 220-229 (1991).
18) 津高新一郎，中野美由紀，喜連川優，高木幹雄：共有メモリ型マルチプロセッサマシンにおける並列結合演算処理，電子情報通信学会技術研究報告（コンピュータシステム），Vol. 91, No. 130, pp. 159-166 (1991).
19) 中野美由紀，喜連川優，高木幹雄：密結合マルチプロセッサにおける関係代数演算の評価—結合演算—，第 35 回情報処理学会全国大会論文集，4 CC-1 (1987).

（平成 4 年 7 月 20 日受付）
（平成 5 年 3 月 11 日採録）

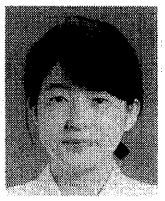
喜連川　優（正会員）

昭和 30 年生．昭和 53 年東京大学電子工学科卒業．昭和 58 年東京大学工学系研究科情報工学博士課程修了．工学博士．昭和 59 年東京大学生産技術研究所第 3 部講師．昭和 60 年同所機能エレクトロニクス研究センター助教授．現在に至る．並列コンピュータアーキテクチャ，データベース工学の研究に従事．DataBase Machines and KnowledgeBase Machines (Kluwer),「コネクションマシン」（著訳）．「データベースベンチマーキング」（訳）．電子情報通信学会，IEEE, ACM 各会員．



津高新一郎

1966 年生．1989 年東京大学工学部電子工学科卒業．1991 年同大学院工学系研究科情報工学専攻修士課程修了．1992 年三菱電機(株)入社．現在，同社中央研究所勤務．在学中，関係データベースシステムにおける並列演算アルゴリズムや並列計算機アーキテクチャに関する研究に従事．現在，自己組織型情報ベースシステムについての研究を進めており，超並列アーキテクチャやコネクショニストモデルに興味を持つ．



中野美由紀（正会員）

昭和 55 年東京大学理学部情報科学卒業．昭和 55〜60 年富士通(株)にてプログラム開発に従事．昭和 61 年東京大学生産技術研究所に入所．現在，同所第三部助手．データベースシステムの研究に従事．データベースにおける並列処理技法に興味がある．電子情報通信学会会員．