大規模関係データベースの並列処理技術に関する研究

平成6年

佐藤　哲司

内容梗概

本論文は，筆者が昭和57年から日本電信電話公社(現在，日本電信電話株式会社に社名変更)の研究所において行なった大規模関係データベースの並列処理方法に関する研究成果をまとめたものである．

関係データベースの適用領域の拡大に伴ってデータベースの大規模化と問合せの複雑化が進み，問合せ処理の高速化が関係データベースの持つ利便性と柔軟性を十分に発揮する本質的な課題となってきている．そこで，本研究では，関係データベース処理に内在する多様な並列性を，システム構成論の観点から種々のレベルで抽出し，専用ハードウェアおよびマルチプロセッサで実現するのに適した並列処理方法を提案する．データベース応用では，対象とするデータの個数や長さが広範に変化するため，これらの変化に対応できる柔軟なハードウェア構成の研究が重要である．提案した方法をデータベースプロセッサ，および市販マルチプロセッサシステム上に実装し，大規模な関係データベースの問合せ応答時間の短縮効果を評価することで，広範な問合せに対する有効性を確認する．関係データベースの問合せ応答時間を長大化させる基本演算は，データベースの表をディスク装置から読み出して全ての行に対して条件検索を行なうサーチ，問合せ結果の並べ換え等で多用されるソート，複数の表を関連づけるジョインである．高速化要求の高い問合せの多くはこれらの基本演算に分解できることから，上記の基本演算を並列処理によって高速化することで，広範な問合せの応答時間を大幅に短縮できることを示す．

本論文は，全7章から構成される．第1章に序論を述べ，第2章では比較器を1次元アレー配置したソートアレーの柔軟な構成法を，第3章では，ソートアレーを基本構成要素とするマルチウェイマージソータの構成法を，第4章では，ソートの前処理としてふるい落とし手法を適用した3フェーズジョイン法を提案し，第5章では，提案法を含めたソート，ジョイン，および，サーチの高速化手法を有機的に統合したデータベースプロセッサの構築事例と性能評価結果を述べる．第6章では，資源共有型マルチプロセッサにおける粗粒度並列処理技術として，負荷配分オーバヘッドを削減す

る手法，および，共有データであるインデックスのアクセス競合を低減する手法を提案し，第７章に以上の章の結論を述べる．以下に各章の概要を示す．

第１章の序論では，問合せ応答時間の長大化要因が，サーチ，ソート，ジョインの３種類の基本演算であることを示し，従来の研究例を紹介するとともに本論文の目的と位置付けを明確にする．

第２章では，１組のメモリと比較器からなる比較ユニットを１次元アレー配置したソートアレーを考案し，比較ユニット間の接続を可変としてレコード長とソート速度に拡張性を持たせた柔軟な構成法を提案する．階層化冗長構成を適用することで，基本ブロックとして40個の比較ユニットを一括集積した巨大チップ (約 $4cm \times 2cm$) が高い良品率で実現できることを示す．

第３章では，上記ソートアレーを基本構成要素とし，ソートできる最大レコード数とソート処理速度に柔軟に対応できるマルチウェイマージソータの構成法を提案し，レコード列を再帰的に格納するワークメモリの利用効率を向上するエリア格納法，および，ソートアレーの両端からレコードを入出力する往復ソート法を考案する．本ソータは，ソータ容量とソート速度の２つの利用者要求に応じた最適な構成を柔軟に実現できることを示す．

第４章では，ハッシュ化ビットアレイを用いたふるい落とし法をマージ結合法と組み合わせて専用ハードウェア化する３フェーズジョイン法を提案する．本方法は，フィルタ，ソート，マージ結合の３フェーズからなり，前２者を専用ハードウェア化して行単位のパイプライン処理を実現すると同時に，処理内容が問合せ条件に依存するマージ結合をホスト計算機上のソフトウェアで処理することとし，種々の問合せに柔軟に対応できる高速なジョインを実現できる．

第５章では，第２章から第４章に示したソートとジョイン，および，データ並列による並列処理を適用したサーチを実現したデータベースプロセッサ RINDA のアーキテクチャと性能評価結果を述べる．行を中心とする比較的粒度の小さい様々な並列処理 (細粒度並列) を専用ハードウェアで実現し，広範な問合せをソフトウェア処理と比

較して数倍から100倍以上高速化できることを示す.

第6章では，資源共有型マルチプロセッサを前提に，負荷配分量を処理の進行に伴って可変とする適応型動的負荷配分法と，インデックスの参照/更新を並列処理するのに適したハイブリッド・インデックス構成法を提案し，市販マルチプロセッサシステム上に問合せの処理系を実現し，提案法がプロセッサ数にほぼ比例した性能向上を達成できることを示す.

最後に，第7章では，本研究で得られた成果を要約し，今後に残された課題について述べる.

本研究に関する関連発表論文

I. 学会論文誌発表論文

(1) 佐藤哲司, 武田英昭, 津田伸生 : “大容量データベース処理に適したソータ構成法”, 情報処理学会 論文誌, Vol. 31, No. 11, pp. 1653–1660 (Nov. 1990).

(2) 佐藤哲司, 武田英昭, 井上 潮, 福岡秀樹 : “データベースプロセッサ RINDA の結合演算処理機構の構成と評価”, 情報処理学会 論文誌, Vol. 32, No. 8, pp. 1006–1013 (Aug. 1991).

(3) Satoh, T., Takeda, H., and Tsuda, N. : “A Multiway Merge Sorter for Sorting of Large Databases”, Journal of Information Processing, Vol. 15, No. 3, pp. 434–440 (Mar. 1993).

(4) 佐藤哲司, 井上 潮 : “関係データベースシステムの高度化技術”, コンピュートロール, Vol. 43, 特集 / 高度応用のためのデータベース, 上林弥彦 責任編集, pp. 54–61, コロナ社 (July 1993).

(5) Satoh, T. and Inoue, U. : “Rinda: A Relational Database Processor for Large Databases”, Emerging Trends in Database and Knowledge-Base Machines : The Application of Parallel Architectures to Smart Information Systems, Abdelguerfi, M. and Lavington, S. H. eds, IEEE-CS Press (掲載予定).

(6) 武田英昭, 佐藤哲司, 中村敏夫, 速水治夫 : “関係演算高速化プロセッサ”, 情報処理学会 論文誌, Vol. 31, No. 8, pp. 1230–1241 (Aug. 1990).

(7) Inoue, U., Satoh, T., Hayami, H., Takeda, H., Nakamura, T., and Fukuoka, H. : “Rinda: A Relational Database Processor with Hardware Specialized for Searching and Sorting”, IEEE Micro, Vol. 11, No. 6, pp. 61–70 (Dec. 1991).

(7) 平野泰宏, 佐藤哲司, 井上 潮, 寺中勝美 : “資源共有型マルチプロセッサにおけるデータベース処理の動的負荷配分法”, 電子情報通信学会 論文誌, Vol. J75–D–I, No. 3, pp. 152–159 (Mar. 1992).

(8) 速水治夫, 佐藤哲司, 中村敏夫, 黒岩淳一, 武田英昭 : “リレーショナルデータベースマシンにおけるサーチ処理方式”, 電子情報通信学会 論文誌, Vol. J75–D–I, No. 4, pp. 232–240 (Apr. 1992).

(10) 井上 潮, 佐藤哲司, 速水治夫 : “データベースプロセッサ RINDA”, 情報処理, Vol. 33, No. 12, pp. 1403–1408 (1992).

II. 研究会等発表論文 (査読有り)

(1) Satoh, T., Takeda, H., and Tsuda, N. : "A Compact Multiway Merge Sorter Using VLSI Linear-array Comparators", Lecture Notes in Computer Science Vol. 367, Proc. 8th Int. Conf. Foundation of Data Organization and Algorithms, Litwin, W. and Schek, H.-J. eds. pp. 223–227, Springer-Verlag (June 1989).

(2) Satoh, T., Takeda, H., Inoue, U., and Fukuoka, H. : "Acceleration of Join Operations by a Relational Database Processor, RINDA", Proc. 2nd Int. Symp. Database Systems for Advanced Applications, pp. 243–248 (Apr. 1991).

(3) Satoh, T., Hirano, Y., Honishi, T., and Inoue, U. : "Design and Implementation of Parallel Database Processing on a Shared Memory Multiprocessor System", Proc. 2nd Far-East Workshop on Future Database Systems, pp. 337–346 (Apr. 1992).

(4) Tsuda, N., Satoh, T., and Kawada, T. : "A Pipeline Sorting Chip", Proc. 1987 IEEE Int. Solid-State Circuits Conf., pp. 270 (Feb. 1987).

(5) Tsuda, N. and Satoh, T. : "Hierarchical Redundancy for A Linear-Array Sorting Chip", IFIP WG 10.5 2nd Workshop on Wafer Scale Integration, Wafer Scale Integration II, Lea, M. ed., pp. 63–74, North-Holland (1988).

(6) Takeda, H. and Satoh, T. : "An Accelerating Processor for Relational Operations", Proc. Int. Conf. Databases, Parallel Architectures, and Their Applications (Parbase-90), Rishe, N., Navathe, S., and Tal, D. eds., pp. 559 (Mar. 1990).

(7) Hirano, Y., Satoh, T., Inoue, U., and Teranaka, K. : "Load Balancing Algorithms for Parallel Database Processing on Shared Memory Multiprocessors", Proc. 1st Int. Conf. Parallel and Distributed Information Systems, pp. 210–217 (Dec. 1991).

(8) Honishi, T., Satoh, T., and Inoue, U. : "An Index Structure for Parallel Database Processing", Proc. Int. Workshop Research Issues on Data Engineering: Transaction and Query Processing, pp. 224–225 (Feb. 1992).

III. 研究会等発表論文 (査読無し)

(1) 佐藤哲司, 津田伸生 : "階層化冗長構成を用いたソート回路の欠陥検出切替法", 昭和 59 年度電子通信学会総合全国大会講演論文集, 398 (1984).

(2) 佐藤哲司, 津田伸生：“一次元アレイを用いたソート処理装置の構成法”, 情報処理学会第 29 回 (昭和 59 年後期) 全国大会講演論文集, 4F–1 (1984).

(3) 佐藤哲司, 津田伸生：“2 段階自己テストによるソート回路の冗長切替法”, 昭和 60 年度電子通信学会総合全国大会講演論文集, 408 (1985).

(4) 佐藤哲司, 津田伸生：“キー指定可能なマルチウェイマージソータの構成”, 情報処理学会第 31 回 (昭和 60 年後期) 全国大会講演論文集, 1B–3 (1985).

(5) 佐藤哲司, 津田伸生：“階層化冗長構成による 1 次元アレイ論理ＬＳＩの欠陥救済”, 電子情報通信学会 (フォールトトレラントシステム研究会), FTS87–4, pp. 27–33 (May 1987).

(6) 佐藤哲司, 津田伸生：“1 次元アレイ論理 LSI の階層化冗長構成法”, 電子情報通信学会創立 70 周年記念総合全国大会 (昭和 62 年) 講演論文集, S18–2 (1987).

(7) 佐藤哲司, 武田英昭：“大容量データベース処理に適したソート手法”, 電子情報通信学会 (データ工学研究会), DE88–1, pp. 1–7 (May 1988).

(8) 佐藤哲司, 武田英昭, 中村敏夫, 速水治夫：“データベースプロセッサ RINDA の大容量ソート処理方式”, 情報処理学会第 38 回 (昭和 64 年前期) 全国大会講演論文集, 3Q–1 (1989).

(9) 津田伸生, 佐藤哲司：“階層化冗長構成による LSI の欠陥救済”, 昭和 58 年度電子通信学会半導体・材料部門全国大会講演論文集, 161 (1983).

(10) 平野泰宏, 佐藤哲司, 井上 潮, 寺中勝美：“資源共有型マルチプロセッサにおけるデータベース処理の動的負荷配分法”, 情報処理学会 (データベースシステム研究会), 92–DBS–82–1 (1991).

(11) 芳西崇, 佐藤哲司, 井上 潮：“並列処理向きのインデックス構成法に関する一考察”, 電子情報通信学会春季全国大会講演論文集, D–104 (1991).

IV. 特許 (成立済)

(1) 佐藤哲司, 津田伸生：“半導体メモリ装置”, 第 1445340 号 (昭 63. 6. 30).

(2) 津田伸生, 佐藤哲司, 川田忠通：“集積回路の冗長構成方式”, 第 1575243 号 (平 2. 8. 24).

(3) 津田伸生, 佐藤哲司：“ソート処理装置”, 第 1581751 号 (平 2. 10. 11).

目次

第 1 章

序論

1.1 はじめに

関係データベースは，1970 年に理論的な基盤がコッドによって提案されて以来，汎用計算機の主要アプリケーションとして事務処理分野を中心に発展してきた．データベース言語 SQL が標準化された 1980 年代後半には，汎用計算機からパソコンまでの幅広い領域で利用されるようになった．利用者の拡大と並行して，数十ギガバイトを超える大規模データベースや複雑な問合せを必要とする応用情報システムが，関係データベースを用いて構築されてきている．製品の販売状況から経営方針を決定するこのようなシステムでは，関係データベースが提供するアドホックな問合せを随時実行できる機能が有効である．

問合せの応答時間は，データベースの大規模化や問合せの複雑化に伴って長大化する．SQL は，利用者が問合せを記述する場合に，問合せがデータベース管理システム DBMS 内で実際にどのように処理されるかを指示する必要がなく非手続的に記述できるので簡便である．しかし，問合せの応答時間は，問合せ内容に大きく依存し数ミリ秒から時には数時間以上に達することもある．アドホックに作成した問合せの条件を少し間違えたためにシステム全体の応答性能が極端に悪化する場合もある．このため，オンライン業務で定常的に利用している時間帯は，アドホックな問合せの実行を禁止する等の運用面での規制が必要になり，関係データベースの利便性が大幅に制限されている．関係データベースの持つ利便性を十分に発揮するには，これまで時間オーダの応答時間を必要としていた問合せを 2 桁から 3 桁程度向上させる高速化が

課題となる．複雑な問合せを十分に高速化することによって，これまでよりも利用者が気軽にアドホックな問合せを行ったり，従来は制限されていたオンライン業務で使用中のデータベースに対する統計処理が可能となり，例えば，精度の高い経営戦略情報などを得ることもできる．

本研究では，最初に，関係データベースの問合せ応答時間を長大化させる主要な基本演算が，データベースの表を条件検索するサーチ，問合せ結果の並べ換え等で多用されるソート，複数の表を関連づけるジョイン[1]であることを示し，次いで，これらの基本演算を**並列処理**によって高速化する手法を述べる．並列処理は，複数の演算回路を並列に動作させることで必要な処理ステップ数 (時間) をハードウェアの面積 (演算回路のゲート数 × 並列度) に置き換えることであり，データベース処理に内在する多様な並列性のどのレベルを利用するかによってシステムが特徴づけられる．一般に並列度を上げる，すなわち並列処理の粒度を小さくして処理の多重度を上げると，問題分割のため，あるいは同期実行のためのオーバヘッドが増大する傾向にある．このため，本研究では，データベースに格納された行あるいは行より細かい粒度での並列処理 (**細粒度並列**) は，専用ハードウェア化することで同期実行のオーバヘッドを削減することを目指す．問合せ条件やデータ分布の偏りによって均一な問題分割が困難な場合は，ソフトウェアで制御可能なマルチプロセッサで実現 (**粗粒度並列**) し，分割のためのオーバヘッドを増やすことなく高速化することを目指す．データの比較と交換を行う比較器を複数個並列動作させる並列ソートは細粒度並列の代表例であり，マルチプロセッサによる並列ジョインは粗粒度並列の代表例である．

本論文では，こうした背景を踏まえて，関係データベースの基本演算であるサーチ，ソート，ジョインの各処理を飛躍的に高速化することを目的に，システム構成論の観点から並列処理の粒度に着目して，専用ハードウェアによる実現を狙いとする細粒度並列処理技術，および，マルチプロセッサでの実現を狙いとする粗粒度並列処理技術を示す．

---

[1]本論文ではジョインを一般的な結合より狭い範囲を示す用語として使用している．詳細は **1.7**用語の定義で述べる．

1.2 本研究の背景

データベースの歴史は1960年代に始まるが，1970年にIBM社のコッドが提案した関係データモデル[11]は，モデルの分かり易さ，データ独立性，データ操作の非手続性に優れており，このモデルに基づいて多くのデータベース管理システムDBMSが開発されてきた．1987年にデータベース言語**SQL**[28, 55]がISOとJISで標準化されたのを契機に，関係データベースは益々広い分野で利用されるようになった．データベース言語SQLが標準化されたことで，DBMS利用者(エンドユーザ)はSQLを使ってプログラムすることでDBMSに依存しない移植性の高いアプリケーションを開発できるようになった．

データベース言語SQLは，何(what)を指定する**非手続型言語**であり，DBMSの内部構造やそこでの動作を意識することなく問合せを記述できる．データベースは，複数の利用者がデータ，さらにはアプリケーションプログラムまで共用する場合が多く，非手続的な記述ができることはデータベースの利便性を向上する上で極めて重要な特徴である．

しかし，非手続型言語SQLを用いて応用プログラムを作成する場合も問題はある．問合せの実行時間がDBMSの内部処理に依存するために，利用者は実際にDBMSの内部でどのような処理が行われているかを考えざるを得ない．関係データモデルでは表が行の集合として定義されるが，DBMS内部では集合演算を個々の行に対する繰り返し演算として実行する．一般に，行の格納位置や格納順序から行を同定できるが，行の持つ属性値から行の格納位置を高速に同定する方法にインデックスがある．与えられた問合せの条件がインデックスを使って評価できるかどうかによって応答時間に2桁から3桁程度の差が生じる．複数の問合せ条件や複数の表に跨がる条件を指定したり副問合せを用いた複雑な問合せでは，実行順序によって応答時間に大きな差が生じる場合もある．この結果，エンドユーザは非手続型の言語を使っているにもかかわらず，処理時間という制約からDBMS内部の処理を想定しながらアプリケーションを作成しなければならなかった．

また，ある種のアプリケーションでは最適な実行順序が明らかであるにも関わら

ず，利用者はDBMS内部の実行順序を制御することができない．DBMSの最適化機構は，問合せ内容，スキーマ，データのサンプリング結果等を解析して最適と思われる実行順序を決定するが，真に最適な実行順序はデータの値と分布に大きく依存し短時間で最適解を求めることは困難である．

上記の問題は，データベース処理の高速化，特に負荷の重い処理を含む問合せの高速化によって解決できる．データベース処理が高速になれば，大規模データベースへの適用や複雑な問合せが実行可能となり，関係データベースの適用領域を拡大できる．

現在，新聞記事データベースや電話帳データベースなどの商用データベースが簡単に利用できる状況になってきている．商用データベースに限らず企業内あるいは個人のデータベースでも，登録されている項目が長期間，広範囲であるほどデータベースを利用する効果が大きくなる．新聞記事などのテキストデータベースの利用では，あらかじめ問合せが決まらない場合が多く，あらかじめ付与した大量のキーワードあるいは記事そのものに対する全文検索が必要となり，ソフトウェア的な高速化手法であるインデックスを有効に利用できない．このような問合せは，関係する表を最低1回は読み込んで全ての行に対する条件判定が必要であり，このような処理をサーチと呼ぶ．

一方，銀行の窓口業務やスーパマーケットの販売在庫管理等では，日報・月報の作成や各種の統計処理を行って経営戦略を立てる業務が日常化している．このようなシステムで有益な情報を抽出しようとすると，複数の表のジョインやソートが必要となる．次の例は，表1.1と表1.2を使って1月期の電話機の販売額を販売員毎に集計する問合せの例である．

SELECT S. 販売員, **SUM**(S. 数量 × P. 単価)	: 出力項目名の指定
FROM 販売表 S, 価格表 P	: 検索対象となる表の並び
WHERE S. 販売日 LIKE '1/%' AND S. 品番 = P. 品番	: 検索条件 (述語の論理式)
GROUP BY S. 販売員	: グループ化

この例では，販売表と価格表の2つ (それぞれS, Rの別名が与えられている) に関連する検索条件 [S. 品番 = P. 品番] が指定されているから，品番によって2つの表を結

表 1.1: 販売表の例

品名	品番	数量	販売日	販売員	倉庫
ハウディホンA	10011	3	1/8	田中	青山
ビジネスホンC	20013	2	1/10	鈴木	港
ハウディホンB	10012	5	1/11	鈴木	渋谷
ハウディホンA	10011	8	1/12	村上	青山
ハウディホンA	10011	3	1/23	田中	渋谷
ビジネスホンA	20011	6	1/25	村上	港
ビジネスホンA	20011	32	2/3	田中	港
ハウディホンB	10012	3	2/8	鈴木	渋谷
ハウディホンB	10012	2	2/10	村上	渋谷
ハウディホンA	10011	1	2/15	村上	青山

表 1.2: 価格表の例

品名	品番	単価
ハウディホンA	10011	42,000
ハウディホンB	10012	58,000
ハウディホンC	10013	64,000
ビジネスホンA	20011	52,000
ビジネスホンB	20012	88,000
ビジネスホンC	20013	126,000

合するジョインが必要になる．更に，販売員毎に集計するグループ化のためにジョイン結果を販売員順に並び換えるソートを必要とする．このように関係データベースでは，単なる格納データの検索・更新以上に複雑な操作を行おうとするとジョイン操作が不可欠である．また，集合操作を前提としているため，条件に合致する行が複数存在する場合は，有効な情報を得るためにソートが必要になる場合が多い．

以上述べたように，販売在庫管理システムに代表される高度なデータベース応用では，前述のサーチに加えてソートやジョインを必要とする複雑な問合せの実行が不可欠である．また，単に個々の表の統計情報を取るだけでなく，新しい観点で属性間の関連を調べたり複数の表間の相関関係を調べたりするので，当然のこととしてジョインを多用することになる．

1.3 本研究の問題設定

関係データベースが本来提供している高度なデータ利用は，データベース言語 SQL に代表される非手続的な言語インタフェースと，問合せ内容に依存しない安定した応答性能によって実現可能となる．そのためには，大規模データベースに対する複雑な問合せの高速化が鍵である．研究の背景で述べたように，関係データベースの利用形態の拡大に伴って，大規模データベースに対するサーチ，検索結果を並べ換えるソート，複数の表を結合するジョインの利用要求が高まってきている．これらの演算は，集合演算を前提とする関係データベースの基本演算である．以下では，サーチ，ソート，ジョインの処理時間を長大化させる要因を分析する．

サーチ

一般に，大容量のデータベースは，ビット単価が安く信頼性の高い磁気ディスク装置に格納されている．近年，磁気ディスク装置はめざましい大容量化が進んでいるが，磁気ディスク装置の技術動向を見ると，ディスク容量の増加と比較してデータの読み出し / 書き込み速度の向上はわずかであり，相対的に読み出し時間が長大化する傾向にある．その結果，たとえ検索条件が 1 つでもインデックスが使用できなければ，アクセス速度の遅いディスク装置から大量のデータを読み出さなければならず応答時間が悪化する．

サーチの応答時間を決定する他の要因として，問合せ文で指定された検索条件を繰り返し判定する条件判定処理がある．この処理は，データベースのサイズ (行数) に比例して繰り返し回数が増大するため，行数に比例した演算量を必要とする．SQL では，検索条件として文字列の一致判定を数値の大小比較と同様に記述することができる．文字列の一致判定 (テキストサーチ) は演算量が大きく，大規模データベースでは膨大な処理時間を要する．

ソート

集合演算を前提とした関係データベースの基本演算 (関係代数演算) は，対象とするデータベースの表，すなわち行の集合が演算する属性に関してソートされていれば，行数に対して線形な時間で処理できる．このため，DBMS の内部では結合演算などでソートが多用されている．また，関係データベースの問合せ結果も集合であり，多くの行を含む場合に利用者はある属性に関してソートされていることを望んでいる．

このように，ソートは関係データベース処理およびその応用で多用される極めて基本的な演算で，レコード数を N とすると $O(N \times \log_2 N)$ の演算量が必要であり，レコード数が大きい場合に多大な演算量が必要とされる．また，行の属性値からソートするためのキー (レコード) を抽出するキー抽出処理も多くの演算量を必要とする操作である．

ジョイン

データベース中に項目が重複して格納されていると，格納媒体である磁気ディスク装置が有効利用できないばかりでなく，重複した項目に対する更新操作がデータベース中に伝搬し論理的な矛盾を生じる場合がある．この問題を避ける方法がデータベースの正規化である．一般に，正規化は表の分割によって実現されるため，正規化されたデータベースに対する問合せは，複数の表を関連づけるジョインが必要になる場合が多い．

ジョインは，2 個の表に共通する列のいくつかから結合キーを定義して，結合キーの値が等しい行を連結する操作である．単純なアルゴリズムでは，第 1 表の各行に対して第 2 表の全ての行を結合キーを用いて比較する操作を繰り返す．このため，結合する 2 つの表の行数の積に比例する演算量を必要とし，結合する表が大きくなると急激に演算量が大きくなる．

本研究は，このような大規模データベースに対する問合せ応答時間の短縮要求を背景として，負荷の重いサーチ，ソート，ジョインの 3 種類の基本演算を高速化する並列処理技術を，システム構成論の観点から整理し確立することを目的とする．

1.4 従来の研究

関係データベースの持つ利便性を発揮するにはサーチ，ソート，ジョインを含む問合せの高速化が重要な課題であり，高速化のための並列処理技術が盛んに研究されている．以下では，並列処理アルゴリズムの提案だけでなく，並列処理システムとして実装・評価が行なわれている研究事例を，サーチ，ソート，ジョインに分類して示す．

1.4.1 サーチ

サーチ時間を長大化させる最大の要因はデータの読み出し速度である．このため，データベースマシン研究の初期段階では，高速アクセスが可能な記憶装置である CCD (Charge Coupled Device) を用いた DIRECT[12] や，磁気バブルメモリを用いて実現した EDC[83] が提案された．しかし，その後の半導体メモリ (RAM) の大容量化と低価格化によって，CCD や磁気バブルメモリはほとんど使用されなくなり，小容量で高速アクセス可能な半導体メモリと，安価で大容量な磁気ディスクによるメモリ階層が確立されるに至っている．データベース応用においても，大容量のデータベースは磁気ディスク装置に格納するのが一般的となっている．

磁気ディスク装置の優位性が確立されるに伴って，ディスク装置をインテリジェント化して 記憶装置の近くでサーチを行う "Logic-per-Track"[73] の考え方に基づく RAP[57] や CASSM[74] が設計・試作された．これらは，ディスク装置のヘッドに対応させてサーチ用のプロセッサ (専用ハードウェア) を搭載し，ディスクの回転に合わせて検索処理を行う構成であり，一定容量毎にプロセッサを付加していくことで応答時間を維持したままデータベース容量を大きくすることができる．ヘッドからのデータ読み込みと同期して検索条件に指定された列の属性値を抽出し条件判定による選択演算を行ない，検索条件に合致した行だけをホスト計算機に返却する．このような処理方法をオンザフライ [58] と呼ぶ．

この方法では，データベースの格納位置に物理的な制約があるため，障害回復処理等のデータベース管理システムで必須な機能を実現するのが困難であった．また，

ディスク装置そのものを専用化するアプローチでは，汎用のディスク装置を使用する場合と比較して装置コストが1桁以上高くなり実用的ではなかった．

次に，磁気ディスク装置には手を加えずに，磁気ディスクの制御装置にインテリジェンスを持たせる方法が提案された．図1.1に示すCAFS[3]の例では，関係データベースにおける選択，準結合，準射影に相当する機能をディスク制御装置に組み込んであり，1個の制御装置で複数のディスク装置に格納されたデータを逐次的に処理できる．汎用のディスク装置では，固定サイズのページを単位として読み出し/書き込み操作および誤り訂正を行うため，ページ単位でサーチが行なわれる．ページ内に格納された行から検索条件に指定された列の属性値を抽出し，条件判定による選択演算を行ない，検索条件にヒットした行だけからなるページを新たに作成してホスト計算機に返却する．

この方法は，汎用の磁気ディスク装置を使用するために記憶コストが小さく，大容量データベースへの適用が容易である．検索条件に合致した行だけがホスト計算機に返却されるため，サーチに要するホスト計算機の負荷が大幅に軽減されている．しかし，ページ単位のサーチであるためディスク装置からのデータ転送が間欠的となり十分な高速化が達成されていなかった．また，制御装置に付加された機能が配下のディスク装置に格納されたデータに対してのみ有効であるため，複数のディスク制御装置間に跨がる結合演算のような複雑な機能を単独で実現するのが難しかった．

SQLではデータ項目の値が存在しない時は，ゼロやスペースでなくNULLと表現する．これに伴って，3値(真，偽，不定)の論理演算が必要になり，ハードウェアの複雑化と処理時間の長大化要因となっていた．ASLM[51]では問合せをNULLを含むサブ問合せと含まないサブ問合せに分解し，サブ問合せの繰り返し実行で3値論理を実現している．連想モジュールと呼ぶ専用ハードウェアでサブ問合せの実行を高速化しているが，ハードウェアの複雑化が避けられなかった．

テキストサーチすなわち文字列の一致判定処理に関しても種々のアルゴリズムが研究されている[75]．このうち，専用ハードウェア向きのアルゴリズムは，連想メモリ等を用いる並列比較法，1文字比較器をアレイ状に接続するセルラアレイ法，状態

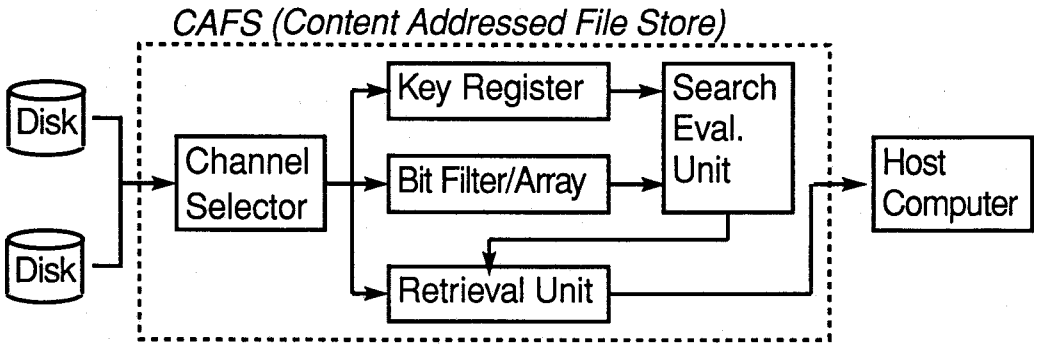


図 1.1: CAFS のアーキテクチャ

遷移表を専用メモリに持たせた有限オートマトン法である．しかし，複雑な検索条件を処理させようとした場合，前 2 者はハードウェア量が大きくなる，後者は処理速度が低下するという問題がある．

1.4.2 ソート

関係データベース処理におけるソートは，データ数に比例する処理時間 $O(N)$ を達成することを目標とするのが一般的である．これは，大規模データベースが通常ディスク装置に格納されており，データ量に比例する時間を必要とするディスクからの読み出し操作とソートをオーバラップさせることを狙いとしているからである．

ソートは，データベース処理以外にも広く利用される基本的な演算であり，専用ハードウェア化の研究が盛んに行われていた [1]．これまでの研究は高速化に主眼が置かれていたため，ソータのハードウェア量が，ソート対象とするレコード数 N に対して $O(N)$[10, 46, 44, 52, 84]，または，$O(\log_2 N)$[79, 77, 30, 35] となる構成，すなわち，ハード量がレコード数に依存して増加する構成が一般的であった．このうち，ハードウェア量が $O(N)$ のソータは，100 万個を超えるような大量レコードをソートするデータベース処理に適用するのは困難である．

現在，データベース処理への適用を狙いとして開発されているソータは，全てマージソート法を基本アルゴリズムとしている [38]．2-way マージ回路を多段接続したパイプラインマージソータの実現例として GREO[2, 35] や DBE[49]，ベクトル演算器

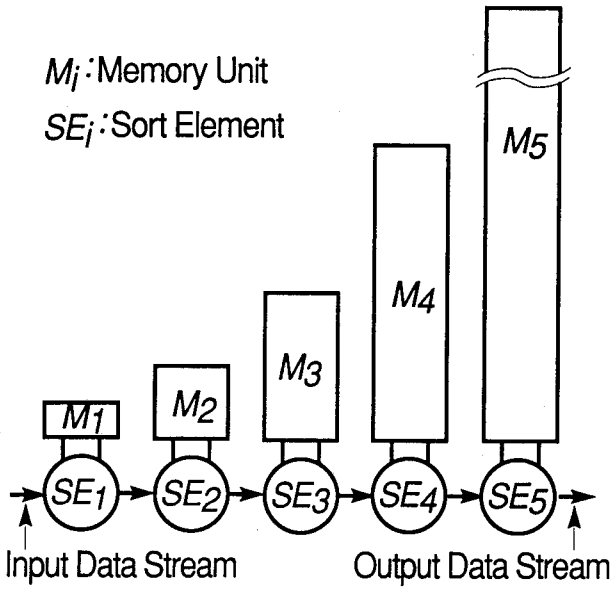


図 1.2: パイプラインマージソータのアーキテクチャ

で実現した 2-way マージャを繰り返し使用する例として IDP[43] がある．図 1.2はパイプラインマージソータの基本アーキテクチャであり，実現された GREO と DBE は，ともに 2-way マージ回路 (図中の SE_i) を 19 個カスケード接続して，2^{19}(約 50 万) 個のレコードをほぼレコード数に比例する時間でソートできる．この方式の問題は，処理対象とするレコード数に見合ったハードウェア量が必要なことである．データベース処理では，ソートする最大レコード数がアプリケーションやデータベースサイズに大きく依存し，事前にそのサイズを予測できないため，パイプラインマージソータを繰り返し使用する等の複雑な制御とそのための回路が必要になる [36].

他の実現例である IDP は，科学技術計算を高速化するためのベクトル演算器で 2-way マージャを実現し，デュアルベクトルと呼ぶキーと実体へのポインタ対をベクトル演算器に繰り返し入力することで高速なソートを実現している．しかし，IDP は大型メインフレームの内蔵プロセッサとして実現されており，演算器が高価で簡単には利用できないという問題がある．

1.4.3 ジョイン

ジョインは2つ以上の関連する表を結合して1つの表にする操作である．結合する2つの表の行数を M,N として従来のジョイン法の演算量を比較する．単純なアルゴリズムであるネストループ法 [5] は $O(M\times N)$ の演算量が必要で，大規模データベースには適用できない．ネストループ法の比較回数を削減する方法として，あらかじめ結合キーの値で2つの表をソートしておき，ソートされた2表をマージ操作によって結合するソートマージ結合法 [5] がある．この方法は，ソートを除く演算量が $O(M+N)$ であり，高速なソートが可能であれば大規模データベースに適用できる．結合操作の演算量を削減する別の方法としてハッシュジョイン法 [34] が提案されている．この方法は，2つの表を同一のハッシュ関数によりバケットに分割 (スプリット) した後に，対応するバケット間に限定して結合操作を行うので，演算量を $O(M+N)$ 近くまで削減できる．ただし，結合するバケットの一方を全て主記憶上に展開できない場合は，ネストループ法と類似な処理が必要になり演算量が増加する．

上記ジョイン法のいずれにも適用できる高速化方法に，結合の可能性のない行を線形時間で除去するふるい落とし法 [50] がある．この方法は，一方の表のジョイン属性に関するハッシュ値をビットアレイに登録した後に，他方の表に関して各行でジョイン属性のハッシュ値を求めてビットアレイを参照し，対応するビットがセットされていない場合は結合の可能性はないとしてその行を除去する手法である．

この方法の課題は，種々のジョイン属性 (キー) に対してハッシュ値が衝突しない優れたハッシュ関数を設計することである．文献 [48, 42] では，短い固定長キーを前提として，これまでに提案されているハッシュ関数の衝突の発生を比較している．キー値の集合が未知な場合は，除算法が比較的良い結果をもたらすことが報告されているが，ふるい落とし処理で扱うキーは，比較的長い可変長のキーで，種々の属性 (整数，文字列等) の組み合わせで構成されるため，単純に除算法を適用することができない．長いキーを扱う方法に，折り畳み法 [48] と排他的論理和法 [6] がある．また，除算法より少ない演算量で同等の効果が得られる乗算法 [42] も知られている．これらの手法を組み合わせて，ふるい落とし処理で要求される条件を満足するハッシュ関数を設計

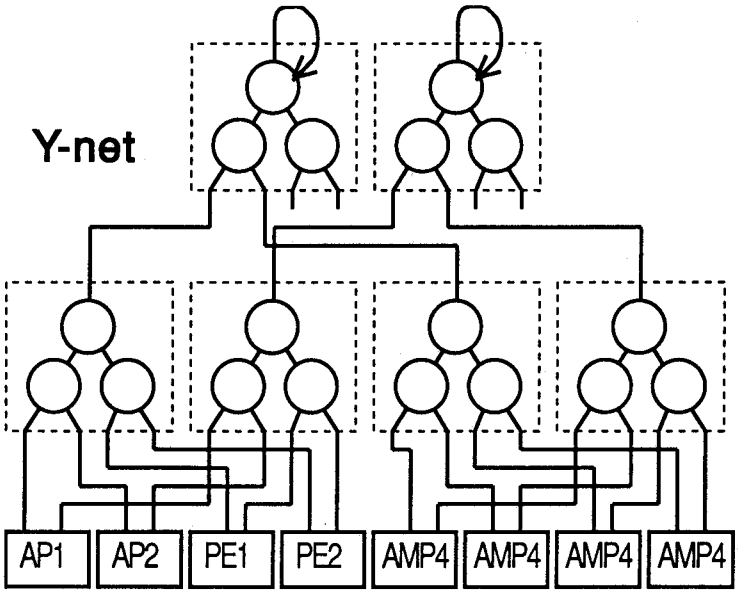


図 1.3: NCR3600 のアーキテクチャ

しなければならない.

これまで研究開発されたデータベースマシンでジョインを高速化しているものに DBC/1012[54], GAMMA[13, 15], Bubba[7] 等がある. これらのマシンに共通する特徴は, データベースを格納するディスク装置を接続したプロセッサをネットワークで接続した資源非共有型のマルチプロセッサシステムである.

DBC/1012 (現在は NCR3600) は, 図 1.3に示すように, AMP (Access Module Processor), AP (Application Processor), PE (Parsing Engine) の 3 種類の演算モジュールを Y-net と呼ばれる 2 重化された 2 進木構造のネットワークで接続している. この Y-net はマージ機能を有する高機能なノードで構成され, ネットワーク上でジョインやソートが行える. データベースは AMP 配下のディスク装置にハッシュを用いて分割格納されており, 上述のハッシュジョイン法や Y-net を用いたソートマージ法などが最適化機構によって使い分けられ, 高速なジョインを実現している.

図 1.4に示す GAMMA は, データベースを格納するディスク装置が接続されたプロセッサ (VAX11/750) と接続されていないプロセッサ (VAX11/750) とが, それぞれ複数台ネットワーク (トークンリング) で接続されている. ディスク装置が接続された

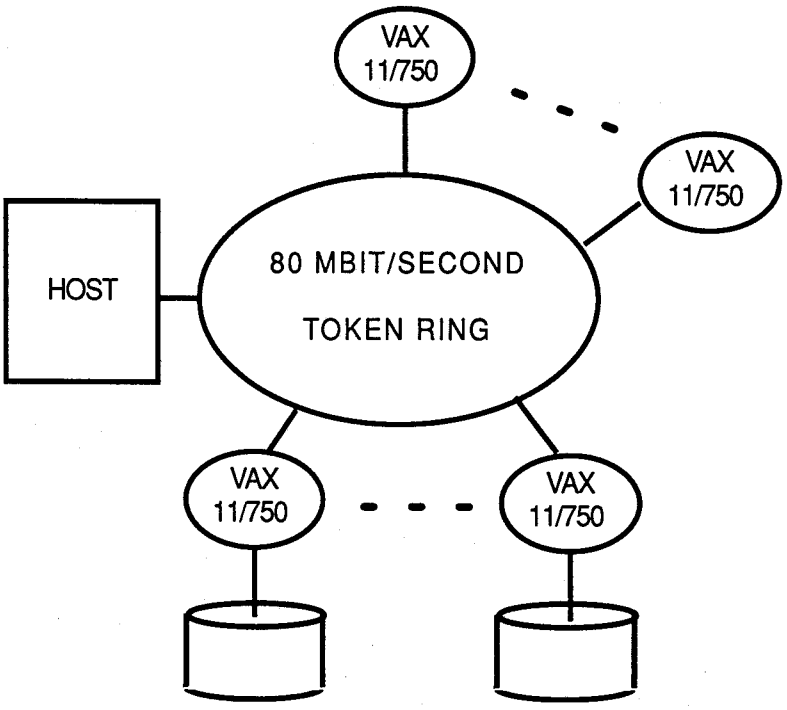


図 1.4: GAMMA のアーキテクチャ

プロセッサでサーチに相当する処理を行ない，その結果をディスク装置が接続されていないプロセッサにネットワークを介して転送し，そこでジョインを実行する．

このような資源非共有型のマルチプロセッサでは，プロセッサ間での負荷の不均衡が生じてプロセッサ全体の処理能力を十分に利用できなくなる可能性がある．負荷を均衡化させる為には，プロセッサ間で負荷の再配分を行なわなければならないが，このための通信オーバヘッドが全体の処理時間を長大化させる要因になる場合がある．負荷配分に関する従来法は文献 [86] で比較評価されているが，いずれの方法も均等な負荷配分を行なうには多くの通信回数が必要であり，通信オーバヘッドの増大が避けられなかった．

1.5 本研究における並列処理技術

以上述べたように，関係データベース処理は，行，ページ，ブロックなどの多様なレベルの並列性が内在し，どのレベルの並列性を抽出してシステムを構成するかが重

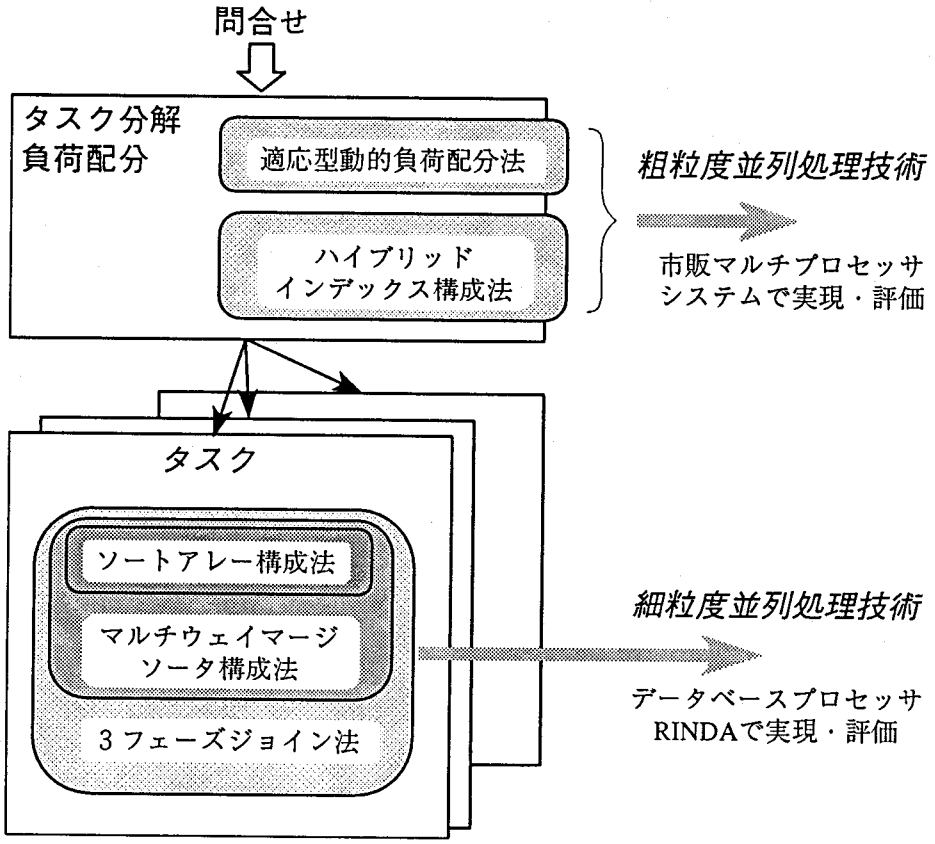


図 1.5: 問合せの並列処理方法と提案法の位置付け

要な問題となる．これまで述べてきたように，大規模データベースに対する複雑な問合せの応答時間を悪化させる要因がサーチ，ソート，ジョインであることから，本研究では，これらの3種類の基本演算を並列処理によって高速化することを考える．

並列処理の基本となる考え方は，複数の演算回路 (下限はゲート，上限はプロセッサノードを考える) を並列に動作させ，必要な処理ステップ数 (時間) をハードウェアの面積 (演算回路のゲート数 × 並列度) に置き換えることである．扱うデータが大量かつその個数が不定なデータベース応用では，面積を決定するシステム構成がデータの個数や長さ，属性に依存しない柔軟なアーキテクチャであることが重要となる．

本研究で対象とする問合せの並列処理方法を図 1.5に示す．問合せは複数のタスクに分解され，それぞれのタスクが並列実行可能な個々のプロセッサで処理される．プロセッサは，専用ハードウェアと汎用 (マイクロ) プロセッサの2種類の実現形態が考

えられ，要求される速度性能や実現時期等の要因に基づいて，いずれかの形態あるいは両者を組み合わせた形態のプロセッサを実現することになる．

プロセッサ間での並列処理を実現するには，並列に実行可能なタスクを生成するタスクの分解，分解されたタスクをプロセッサに割り当てる負荷配分，共有データであるデータベース (表やインデックス) へのアクセス競合の低減が課題となる．一方，サーチやソート，ジョインの各演算は，行に対する操作の繰り返しで目的とする処理を実行できる．実行順序に依存関係があるジョインは，いくつかのフェーズに分解することでフェーズ内ではデータ並列が実現可能である．どのような並列処理が実現できるかは，対象とする演算に大きく依存する．以下では，まず，サーチ，ソート，ジョインの各演算に対する並列処理方法を述べ，次に，負荷配分法等の粗粒度並列に関する手法を述べる．

本論文では，プロセッサ間で実現する並列処理を**粗粒度並列**，プロセッサ内部で行あるいは行と同程度の粒度で実現する並列処理を**細粒度並列**と言う．なお，特に断らない限り並列処理は，異なるデータに同一の演算を施すデータ並列と，あるデータに異なる演算を時系列的に施すパイプラインの両方を指す．

1.5.1 サーチの並列処理

本研究では，ページ単位でアクセスする汎用のディスク装置に手を加えることなく，RAP[57] 等で実現されているオンザフライを実現することを目指す [21]．利用者が指定する様々な問合せ条件は，述語と呼ばれる真偽判定の最小単位の条件式をブール演算子 AND, OR, NOT で結んだ論理式で与えられる．この論理式を完全にハードウェアで実行できれば極めて高速なサーチが実現できるが，ハードウェア規模と開発コストの増大を招くので現実的でない．そこで，一般的なアプリケーションで頻繁に使用され，ハードウェア化による高速化効果が顕著な述語を抽出し，それらの述語と述語に関する論理式だけをハードウェアで直接実行し，ハードウェアで実行されない部分はホスト計算機上のソフトウェアが処理を継続する方式とする．サーチプロセッサ (専用ハードウェア) のために抽出した述語は，整数とデシマルの 2 つの属性に対する比較演算，および，文字列に対するワイルドカードを含めた一致判定である．

ディスク装置からのデータ転送速度 (3M バイト / 秒) に追従したオンザフライを実現するために，ページからの行の抽出，行から述語で使用された属性の抽出，述語判定と論理式の評価を，行単位で完全にパイプライン化し高速化を図る．文字列の一致判定は，複数の比較器を用いたデータ並列 (セルラアレイ法) を実現し，他の述語判定と同等の時間で処理可能としてパイプラインに組み込む．また，SQL で規定された NULL の扱いを 2 値論理で実行できるように論理式を書き換え，ハードウェア量の増大を抑える．

オンザフライを実現する第 2 の課題は，ディスク装置から間断なくデータをサーチプロセッサに供給することである．ディスク装置のヘッドを移動せずに読み出せる範囲 (シリンダ) 内で，ページを連続的に読み出すマルチトラックリード機能と，複数面のバッファを用意してデータ書き込みと読み出しに交代で使用するダブルバッファ技術を組み合わせて，ディスクから複数ページを連続に読み出す操作と，パイプライン化されたサーチを重畳し，極めて大量に連続読み出しされたデータに対するオンザフライを実現する．

以上示したサーチの高速化において，行や列 (属性値) の抽出，述語の判定等，行単位でのパイプライン化，および，文字列の一致判定のためのデータ並列は，本論文で示す細粒度並列を実現する部分である．データの連続供給を実現するための処理は，ページを単位としたパイプライン処理であり，並列処理の単位はやや大きいが完全にスケジューリングされた固定的な負荷配分であるから細粒度並列の範疇とする．

1.5.2 ソートの並列処理

関係データベース処理に適用するソータは，数個から数 100 万個の広範な個数のレコードを効率よく扱えることが要求される．また，ソートできるレコードの長さは，1 回のソートでは固定であったとしても，毎回異なる長さ (数バイトから数 100 バイト) が効率よく扱えることが重要な要件となる．そこで，本研究では，マージウェイ数の大きい k ウェイマージ回路を専用ハードウェアで実現し，k ウェイマージの繰り返しで任意の個数のレコードをソートできるマルチウェイマージソータ構成法を提案する [62, 65].

まず，マージウェイ数に依存しない高速な k ウェイマージ回路を実現する並列比較回路となるソートアレーの構成法を考える．ソートアレーは，比較ユニットを 1 次元アレイ配置した構造からなり，全ての比較器でシストリックに比較と転送動作を繰り返す．これは，k ウェイマージで必要となる最大値 / 最小値を持つレコードの抽出と，k 個のレコードのソートを待ち時間なしで実現できる．比較ユニット間の接続を可変とし，複数ユニットでユニット群を構成する．ユニット群内の比較ユニットを連動させることで，ソート可能な最大レコード長の拡大やソート処理速度 (スループット) を向上できる柔軟な構成を実現する．さらに，全体が繰り返し構造からなるソートアレーを大規模一括集積した巨大チップを実現する．比較ユニットをビットスライス化したセル，比較ユニット，ユニット群のそれぞれの階層に予備回路を設け，巨大チップを高い良品率で実現する．

次に，レコードの (並列) 比較を行なう上記ソートアレー，レコードを格納するワークメモリ，逐次的なマージ操作を繰り返すマージ制御回路からなるマルチウェイマージソータの構成法を考える．ワークメモリに格納された k 個のレコード列のいずれかからレコードを読み出して，そのレコードが属していたレコード列を識別するバンクタグを付与してソートアレーに入力し，ソートアレーから出力されたレコードの持つバンクタグから次に入力するレコードの属するレコード列を決定してレコードの読み出し操作を行なう．ソートアレーではレコードの入力と出力を同期した並列比較が行なわれており，上記操作の繰り返しでマージ操作を継続できる．

k ウェイマージの繰り返しによるソートでは，ワークメモリの格納された k 個のレコード列を読み出して，マージした結果を同じワークメモリに書き戻す操作を繰り返す．このためワークメモリを有効に利用できるメモリ管理法が必要となる．本論文では，マージウェイ数が大きいほどメモリの利用率を高くできるエリア格納法を提案し，実装する．

ソートアレーを構成する比較ユニットの個数を増やすことで，マージ速度を低下させることなくマージウェイ数を拡大でき，マージ段数削減によりソート処理時間を短縮できる．一方，ソートできる最大レコード数は，ワークメモリの容量のみに依存する．したがって，ソータ容量とソート処理速度 (ソート処理時間) の 2 つの利用者要

求に応じた構成のソータを柔軟に実現でき，ソートするレコード数が大きく変動するデータベース応用に適したソータを実現できる [76].

以上示したソータの構成において，マージ操作におけるレコードの読み出しとソートアレーへの入力，ソートアレーからの出力と次の読み出しレコードの決定はパイプライン化されており，更に，これらの操作とソートアレーにおけるレコードの並列比較は完全に重畳されており，細粒度並列を実現しているといえる.

1.5.3 ジョインの並列処理

上記のマルチウェイマージソータを利用したソートマージ結合法を発展させ，ハッシュ化ビットアレイによるふるい落とし法を組み合わせた3フェーズジョイン法を提案する [63]. 本方法は，結合可能性のない行を削除するフィルタフェーズと，ソートフェーズ，マージ結合フェーズの3フェーズからなり，ハードウェア化による高速化効果が大きく処理の柔軟性が損なわれない前2者を専用ハードウェアで実現する.

3フェーズジョイン法は，結合対象とする2表のサイズ (行数) によって，ふるい落としを行なう表を片方にするか両方にするかの2種類の方式が実現できる. 前者を片ハッシュ結合法，後者を両ハッシュ結合法と呼び，いずれの方法でもふるい落としで残った行数がソータの容量以下であればジョインが行なえる.

ふるい落とし法の課題であるハッシュ関数は，乗算法と折り畳み法を組み合わせて，専用ハードウェア化に適した乗算重ね合わせ法を提案する. 可変長のキーを固定長のキー切片に細分化して，各キー切片に対する乗算をあらかじめ作成した乗算表を用いて行ない，その結果をビットシフト (回転) と排他的論理和により重ね合わせる. この方法は，種々のデータ属性に対してハッシュ値の衝突率を十分小さくできる. 提案するハッシュ関数は専用ハードウェアでの実現に適しており，ハードウェア化することで3フェーズジョインを行単位でパイプライン動作させることができる.

以上述べたサーチ，ソート，ジョインを高速化する細粒度並列は，データベースプロセッサ RINDA[26, 27, 67] として実現し，種々の問合せを用いて性能向上効果を評価する.

1.5.4 粗粒度並列処理

これまで述べてきた細粒度並列は，必要な処理 (タスク) を細かく細分化して，それぞれの処理を独立したハードウェア割り当てて全体をあらかじめ決められた実行順序 (スケジュール) で動作させている．このようなタスクの分割とスケジュールは，極めて難しい設計技術が必要であり，比較的単純な画一的処理を実現するにも膨大な設計工数を必要とする．そこで，タスクの分割や割り当てを比較的粗い粒度で行う粗粒度並列処理が重要視されてきている．

粗粒度並列では，並列に実行可能なタスクを生成するタスクの分解や，分解されたタスクをプロセッサに割り当てる負荷配分を，問合せの内容やプロセッサ台数，稼働状況等に応じて問合せの実行時に動的に行う必要がある．このため，タスク分解や負荷配分のオーバヘッドによって，プロセッサ台数を増やしても十分な高速化 (スケーラビリティ) が達成できない場合がある．特に，関係データベースの並列処理では，大容量のデータベースの格納 (配置) 法，オーバヘッドが小さい均等な負荷配分法，インデックス等の共有するデータに対するアクセス競合の低減法が課題となる．

個々のプロセッサがディスク装置を備える資源非共有型のマルチプロセッサシステムでは，種々の問合せに対してプロセッサの負荷が均等になる様にデータベースを分散格納するのは困難である．そこで，本研究では，複数のプロセッサから共通にアクセスできるディスク装置にデータベースを格納する資源共有型マルチプロセッサシステムを前提に，動的な負荷配分法およびアクセス競合の低減法を提案する [66, 64].

マルチプロセッサによる並列処理では，特定のプロセッサに負荷が集中すると，他のプロセッサはそのプロセッサの処理が終了するまで処理の完了を待たされ，全プロセッサの処理が終了するまで応答時間が引き伸ばされる．このような負荷の不均衡による応答時間の悪化要因をスキューと呼ぶ．このスキューを小さくするには，1 回に配分するタスクのサイズ (粒度) を小さくして負荷の均等化をすれば良い．しかし，負荷配分の粒度を小さくすることは，同期実行のためのオーバヘッドを増やす原因となる．すなわち，配分タスクのサイズを小さくすると配分回数が増大し，ほぼ配分回数に比例して負荷配分のためのオーバヘッドが増大する．本研究ではこの問題を回避

するために，プロセッサに配分するタスクのサイズを未配分のタスクサイズの関数として算出する適応型動的負荷配分法を考案した [22]．この方法は，配分タスクのサイズを可変とすることで，スキューを増大させることなく大量のタスクを少ない回数で配分することができる．

データベース処理を高速化するソフトウェア的な従来技術である木構造のインデックスは，アクセスが常にルートノードから開始されるのでアクセスの偏りが大きい．また，木構造の再構成中は他の検索 / 更新処理が中断される等の問題があり，並列処理との親和性が低い．そこで，複数のサブ木とサブ木に負荷を分配するハッシングとを組み合わせたハイブリッドインデックス構成法を提案する [24]．インデックスを用いる 1 件検索 / 更新トランザクション間の並列処理と，インデックスを用いる範囲検索トランザクション内での並列処理が可能である．

1.6 本論文の構成

本論文は，第 2 章以降を以下のように構成する．

第 2 章は，マルチウェイマージソータの主要構成要素である，1 次元アレイ構造に比較器を配置したソートアレーの柔軟な構成法と，大規模一括集積による巨大チップの設計法を論じる．ソートアレーは，データベースの表を構成する行から抽出したデータ (レコード) が処理の対象であり，本論文で示す最も細かい粒度での並列処理を実現している．本ソータのように繰り返し構造を持つ論理回路では，予備回路を階層的に設ける階層化冗長構成法を適用することで複数の比較ユニットを一括集積した大規模ソートアレーチップが高歩留りで実現できることを示す．

第 3 章は，第 2 章に示したソートアレーを基本構成要素とし，ソートできる最大レコード数とソート処理速度に柔軟に対応できるマルチウェイマージソータの構成法を論じる．

第 4 章は，ソートマージ結合法の前処理として，ハッシュ化ビットアレイを用いたふるい落とし法を組み合わせた 3 フェーズジョイン法を論じる．

第 5 章は，上記のソータおよびジョイン法を適用した専用ハードウェアの構成法

を論じる．更に，ディスク制御装置に付加したサーチ専用ハードウェアの構成法を示し，これらを組み合わせて実現したデータベースプロセッサ RINDA の性能向上効果を述べる．RINDA システムは，関係データベースの問合せ処理を高速化することを目的として開発した専用ハードウェアおよびその制御ソフトウェアである．

第 6 章は，大量データを複数プロセッサに配分する際のオーバヘッドを削減することを狙いとした，問合せ条件の変化やデータ分布の偏りに強い適応型動的負荷配分法を提案し，データフロー型とデータ駆動型制御を組み合わせたデータベースの並列処理法を論じる．市販のメモリ共有型マルチプロセッサシステム上に負荷の重い問合せ処理を実装しスケーラビリィティを評価する．

第 7 章は，本研究の総括であり，第 6 章までで述べた本研究の技術範囲と成果についてまとめる．

1.7 用語の定義

本論文で用いる用語の定義を以下に示す．なお「列」「行」「表」の定義は，JIS 規格 [55] に従っている．

列 (Column)： 時間がたてば変わる値のマルチ集合 (必ずしも互いに異ならない対象の順序づけられていない集まり) とする．同じ列のすべての値は，同じデータ型 (表現可能な値の集合) とし，同一の表中の値とする．1 つの列の 1 つの値は，表から選択できるデータの最小単位 (項目) であり，更新できるデータの最小単位である．

行 (Row)： 値の並びとし，その並びは空ではない．同じ表のすべての行は，同じ基数を持ち，その表のすべての列の値を含む．表のすべての行の i 番目の値は，その表の i 番目の列の値とする．行は表に挿入でき，表から削除できるデータの最小単位である．

表 (Table)： 行のマルチ集合とする．表の次数は，その表の列の数とする．表の次数は，あらゆる時点において，それの各行の基数と同じとし，表の基数は，それの各列の基数と同じとする．

データベース (Database)： 論理的には，表および表に付帯するすべてのデータの集まりである．表に付帯するデータには，表の構造を定義したスキーマや，表へのアクセスを高速化するためのインデックス等が含まれる．

インデックス (Index)： 表を構成する1個以上の列に対して，各行が持つ属性値を入力として，その行の格納位置，例えばディスク上の物理的な格納位置を出力とするための変換を行う手法で，属性値と格納位置を組にしたデータを例えば木構造に格納しておくことで実現できる．他の実現例としてハッシュを用いる方法もある．

射影 (Projection)： 1つの表に対する演算であり，指定された列だけから構成される表を導出する．一般に，導出された表は，元になる表より列の数が少ないので，全ての列の値 (属性値) が等しい複数の行が含まれる場合が多く，このような重複した行 (重複行) は1つの行とする重複除去を行う必要がある．この重複除去処理は，グループ化操作と同等な演算量を必要とするため，列の削除だけを行う準射影をハードウェア化する場合が多い．

選択 (Selection)： 1つの表に対する演算であり，指定された条件を満たす行だけから構成される表を導出する．この場合は，列の数が変わることがないので，元の表に重複行がなければ導出した表にも重複行はない．

結合 (Join)： 2つの表に対する演算であり，指定された条件を満たす行を結合して得られる行から構成される表を導出する．CAFS の例では，第1の表の結合条件でビットアレイを設定し，第2の表でビットアレイを参照することで結合に相当する操作を実現している．このため，結合の結果には，第1の表にだけある列は含まれない．これを準結合といい，通常の結合演算で導出される結果に，第2の表に関する射影を施したものと同じ結果が得られる．

サーチ (Searching)： 表の全ての行から，問合せ条件で指定された列の属性値を抽出し，問合せ条件を満足する属性値を持つ行だけを出力する操作であり，上記の射影と選択を同時に実現できる．一般に，表は磁気ディスク装置等の2次記

憶装置に格納 (1.3参照) されており，2 次記憶装置から逐次的にデータを読み出して条件判定を繰り返し適用することから，この操作をスキャンと呼ぶ場合もある．

ソート (Sorting)： 表を特定の行 (属性) の値で並べ換える操作であり，属性値が小さい行を先頭にする昇順ソートと，属性値が大きい行を先頭にする降順ソートがある．一般には，属性値の比較だけを行なう順序付け (Ordering) もソートと呼ぶが，本論文では，順序付けされた並び順に行を入れ換える (Exchanging) 操作を含めてソートという．

ジョイン (Joining)： 結合を実現する処理形態の 1 つであり，インデックスを使用しないで第 1 表と第 2 表を結合する処理を指す．すなわち，第 1 表と第 2 表に対して上記のサーチを行ない，その結果の一時表同士の対応する行を結合属性で連結する処理をジョインという．

粒度 (Granularity)： 粒度とは並列処理の単位，プロセッサの規模や性能などを意味する．一般には，ソフトウェアおよびアルゴリズム上の粒度とハードウェア上の粒度 (プロセッサ粒度) の 2 種類がある．粒度は，粗粒度 (coarse grain)，中粒度 (medium grain)，細粒度 (fine grain) に分類される場合が多いが，これらの間に明確な区分はない．本論文では主にアルゴリズム上の粒度を扱うが，細粒度の並列処理は比較器等の単純なハードウェアを前提とし，粗粒度の並列処理はマルチプロセッサシステムでの実現を前提としている．このため，ハードウェア上の粒度を扱っていると見ることもできる．

第 2 章

大規模一括集積向き一次元ソートアレイの構成法

2.1 まえがき

本章では，前章で述べた細粒度並列処理の典型例であるソートアレー (並列比較回路) の構成法 [65]，および，階層化冗長構成を適用した大規模一括集積法 [61] を論じる．

ソートは，関係データベースの結合処理等で多用される非数値処理の基本演算であり，多大な回数の比較演算が必要なため専用ハードウェアを用いた種々の高速化手法が提案されている [1]．関係データベースでは，数個から数百万個の広範な個数と広範な長さのレコードを効率よくソートすることが求められる．したがって，回路規模が $O(n)$ 以下 (n はレコード数) でハードウェア制限を超える個数でもソフトウェア的方法との併用でソートできること [32]，および，種々のレコード長に適応できる柔軟なハードウェア構成であることが望まれる．特に，ソートできるレコードの長さは，1 回のソートでは固定であったとしても，問合せの内容やデータベースの構成に依存して数バイトから数 100 バイトまで大きく変動するため，種々の長さのレコードを効率よく扱えることがハードウェア化する際の重要な要件となる．

筆者らが提案しているマルチウェイマージソータ [62] (第 3章参照) は，$\lceil k/2 \rceil$ 個 ($k \gg 2$) の比較器を 1 次元配列したソートアレー [60] と大容量ワークメモリで k-way マージャを構成し，逐次的な k-way マージの繰り返しでワークメモリに格納されたレコードをソートする．回路規模を決定するマージウェイ数 k はレコード数 n と独立に設定でき，逐次的な繰り返し操作で大容量ソートを行なうので，ソートできる最大レ

コード数にハードウェア構成上の制約がない.

ソートアレーは，2個のレコードを格納するメモリとそれらを比較・転送する比較転送回路からなる比較ユニットを1次元アレイ状に配置した構成からなる．シストリックアーキテクチャ [45] に基づく同期した並列比較動作により，$\lceil k/2 \rceil$ 個の比較ユニットで構成されたソートアレーは，k 個のレコードを入出力時間 $2k$ でソートできる．レコードを比較ユニット内のメモリに格納し転送するため，ソートできるレコード長は，比較ユニット内のレコードを格納するメモリの容量に依存している.

本章では，上記のレコード長に関する制限を解消することを目的として，複数の比較ユニットを連結動作させることができる柔軟なソートアレー構成法を述べる．ソートできる最大レコード長を大きくするために比較ユニット内のメモリ容量を大きくしたのでは，レコード長が短い時にハードウェア (メモリ) を有効に利用することができない．このため，比較ユニット内のメモリ容量は比較的小さくしておいて，メモリ容量を超える長さのレコードは，複数ユニットに分割格納してそれらのユニットを連結させることでソートする．また，比較転送のデータ幅を拡大する方向に比較ユニットを結合することで，1回の比較転送のデータ幅を拡大しソート処理速度を向上することもできる．比較ユニット間の接続を可変構造として必要とされるレコード長やソート処理速度に応じて複数ユニットを連結あるいは結合させた場合には，ソートアレーを構成する比較ユニットの個数が見かけ上少なくなり，ソートできるレコード数が少なくなるが，いずれのレコード長およびソート処理速度に対してもハードウェアを有効に動作させることができる.

ソートアレーの基本アルゴリズムは奇偶転置法 (odd-even transposition sort) を並列化したバブルソート法 [38] である．これまでに反転ソータ [10]，昇降ソータ [46]，ゼロタイムソータ [52] 等の実現例が報告されているが，前2者は磁気バブリメモリを前提に，最後のゼロタイムソータは半導体メモリを前提にして，メモリセル中に比較機能を組み込む Logic-in-memory の考えに基づいている．このため，比較機能を実現するセルがメモリのワード方向とビット方向に強く接続され，レコード数やレコード長に対する十分な拡張性を持たせた柔軟な構成を実現できなかった.

奇偶転置法と異なるアルゴリズムで動作する1次元アレイ構造のソータに並列計数ソータ [84] がある．これは，レコードの入力時にランク (順位) を決定し，入力完了時に求まるランクに基づいてレコードを置換し出力する．このソータは，入力レコードを全てのユニットにブロードキャストしてランクを決定するため，全ユニットにつながるバスが必要となる．このため，並列計数ソータは，上述のレコード長やソート速度を拡張する可変構造を実現するのが難しかった．

比較ユニット間の接続を可変としたソートアレーは，複雑な接続配線を含めて多くの比較ユニットを一括集積することで，より多くのレコードを1チップでソートでき，装置の小型化と高信頼化を実現することができる．ソートアレーは，規則的な繰り返し構造を有しLSI化した時の入出力端子数が一括集積するユニット数に依存しなため，大規模一括集積 (WSI) に適している [81]．筆者らは，こうした繰り返し構造を持つ論理回路の階層構造に着目して，それぞれの階層に予備回路を配置することで，チップ全体としては少ない予備回路量で高い良品率を達成できる階層化冗長構成法を提案している [60, 61, 82]．ソートアレーにおける階層構造には，比較ユニット，複数の比較ユニットを連結 / 結合させるユニット群がある．さらに，比較ユニット内の主要回路をビットスライス構成とした比較セルの階層を設けることができるため，容易に3階層までの階層化冗長構成法を適用することができる．しかし，高い良品率を得るために予備を増やすと相対的にチップ上に占める基本回路の比率が低下する問題があった．本論文では，ソートアレーチップの大規模一括集積を例に，階層数や個々の階層における予備の配置量を最適化する方法についても述べる．

以下，**2.2** ではソートアレーの基本構成，比較ユニット間の接続を可変としてレコード長拡大と比較速度向上を実現する柔軟なアレー構成法を示す．また，従来型ソータとの相違点を述べる．**2.3** では階層的に予備を設ける階層化冗長構成法を適用することで複数ユニットを一括集積したソートアレーチップが高い良品率で実現できることを示す．さらに，良品率と冗長度の比からなる有効利用率を最大にする冗長構成を求めることでWSI規模を最適化できることを示す．最後に **2.4** では得られた結果をまとめる．

2.2 ソートアレーの構成法

2.2.1 基本アーキテクチャ

シストリックアーキテクチャ [45] は，単純な機能を実現したセルを規則的に配置したアレーを用いて，逐次的なデータ入力と並行に，セルでの演算とセル間のデータ転送を交互に，かつ，全セルで同期して行なうことを特徴とする．1 次元アレーの場合，各セルは隣接する 2 個のセルとだけ結合され，直接通信できるのはこれらの隣接セルに限定される．レコードの比較と交換の繰り返しからなるソートは，アレーを構成する個々のセルで比較操作を，セル間のデータ転送で交換操作を行うことで実現できる．アレーの端からレコードを逐次入力し，各セルで比較と比較に基づく転送を同期して並列に実行する．このように，全体をパイプライン動作させることでシストリックアーキテクチャを自然に実現できる．

ソートアレーの基本構成を図 2.1に示す．ソートアレーは，セルに相当する比較ユニットの 1 次元アレー構造からなり，各比較ユニットは，比較器 (COM)，2 個のメモリ (MEM A/B)，入出力回路 (I/O) で構成される [60]．レコードは各ユニットのメモリに保持され，固定サイズ (例えばバイト) を単位とする比較と転送の繰り返しで 1 レコードを処理する．1 個のメモリ (MEM A あるいは MEM B) 容量以下の長さを持つレコード k 個は，$\lceil k/2 \rceil$ 個の比較ユニットからなるソートアレーでソートできる．

2 個の比較ユニットからなるソートアレーを用いて，4,1,6,3 の 4 個の数値レコードを降順ソートする例を図 2.2に示す．初期状態では，比較ユニットのメモリ (およびレジスタ) は 0 に初期化しておく．各比較ユニットは，メモリに格納された 2 個のレコードのいずれか一方 (入力操作時 (Step1 $\cdots$ Step4) は小さいレコード，出力操作時 (Step5 $\cdots$ Step8) は大きいレコード) を隣接する比較ユニットに出力すると同時に，他方の隣接ユニットから転送されて来るレコードを入力し，出力によって空いたメモリに格納する．レコードの比較 (Compare) はレコードの入力時に行ない，比較結果に基づいて転送 (Transfer) を行なう．これらの操作は，交互にかつ全ユニットで同期して行なう．

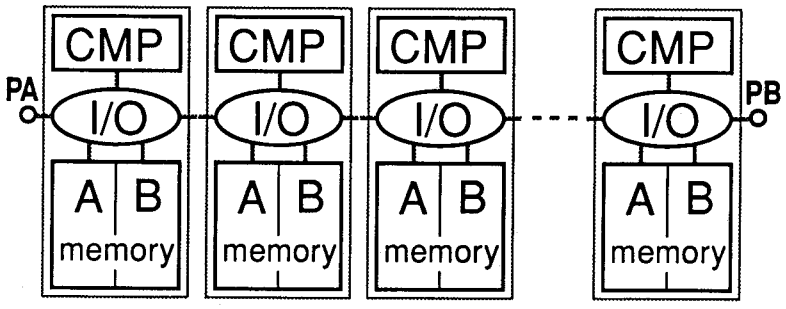


図 2.1: ソートアレイの基本構成

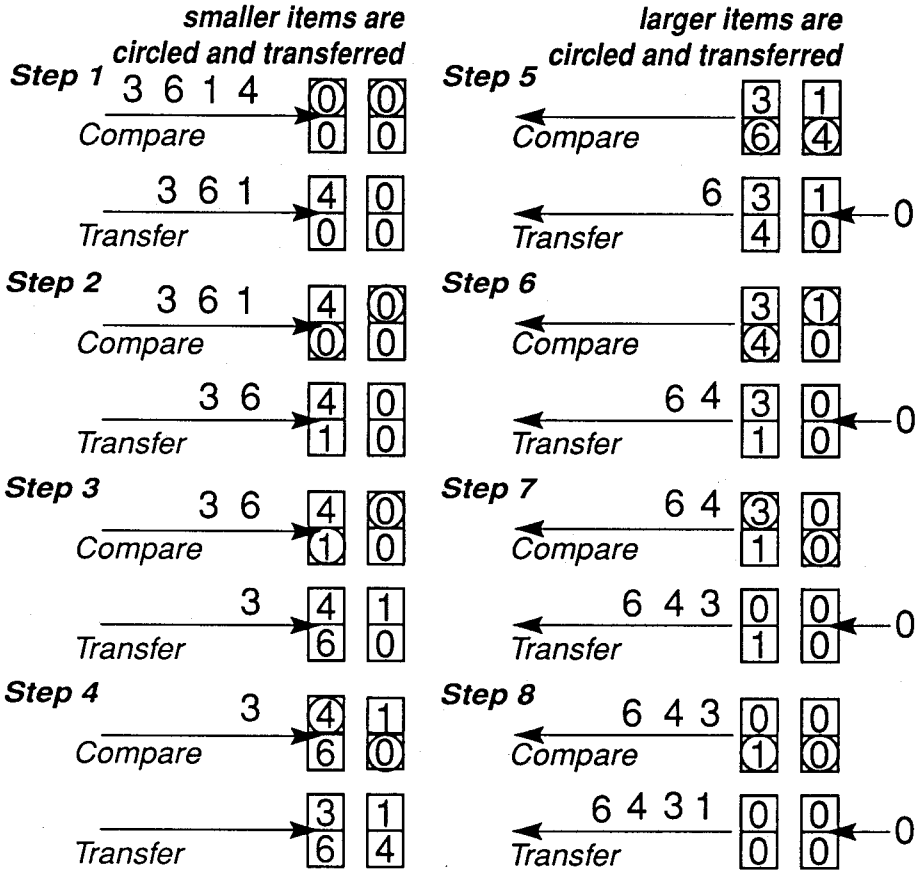


図 2.2: 降順ソート処理の様子

2.2.2 比較ユニットの構成と動作

データベース処理におけるソートは，文字列等の比較的長いレコードもソートできなければならない．このため，比較ユニット内のメモリを多バイト化して長いレコードを扱えるようにする．メモリを多バイト化したソートアレーの構成を図 2.3に示す．各比較ユニットは，比較器 (COMPARATOR)，2 個のレジスタ (REG A/B)，2 個のメモリ (MEM A/B)，入出力回路 (SW A/B/C)，転送回路 (SW L/R) と，フラグレジスタ $(F_{EQ}, F_{GT}, F_{SEL})$ で構成される．$\lceil k/2 \rceil$ 個の比較ユニットからなるソートアレーは，比較ユニットのメモリ (MEM A あるいは MEM B) 容量以下の長さを持つレコード k 個をソートできる．

各ユニットのメモリに保持されたレコードは，固定サイズ (例えば 1 バイト) を単位とする比較と転送を繰り返しながらレジスタ F_{GT} に比較結果を累積する．1 レコードの比較転送が完了したら比較結果をレジスタ F_{SEL} に移し，この値を用いて入出力回路を制御する．転送回路は，レコードの入力操作時 (Step1 $\cdots$ Step4) と出力操作時 (Step5 $\cdots$ Step8) で，レコードの転送方向を切り替える．比較ユニット間のデータ転送を 1 バイト単位で行う場合，長さ l バイトのレコードは l 回の比較・転送操作の繰り返しでソートされる．個々のレコード長が異なる場合は，それらの最大長 l に長さを揃えてからソートする．タイミングチャートを図 2.4に示す．隣接ユニットからの入力データ DATA IN はクロック t_2 でレジスタ REG A にフェッチされる．同時にメモリ MEM B のデータはレジスタ REG B にフェッチされる．一方，メモリ MEM A のデータは出力データ DATA OUT として隣接ユニットに送出され，隣接ユニットのレジスタ (REG A か REG B のいずれか) にフェッチされる．クロック t_2 でレジスタにフェッチされた 2 個のデータは，比較器の入力となると同時にメモリへの書き込みデータとなる．以上の操作でメモリ MEM A から読み出されたデータは隣接ユニットのメモリに書き込まれるので，1 サイクル ($t_0 - t_0$ 間) でユニット間のデータ転送が完了する．比較結果はクロック t_3 でレジスタ F_{EQ}, F_{GT} に設定され，比較と転送は完全に重畳して 1 サイクルで処理を完了する．

第 $i(1 \leq i \leq l)$ バイトの比較において，$F_{EQ} = 1$ は第 $(i-1)$ バイトまでの比較結

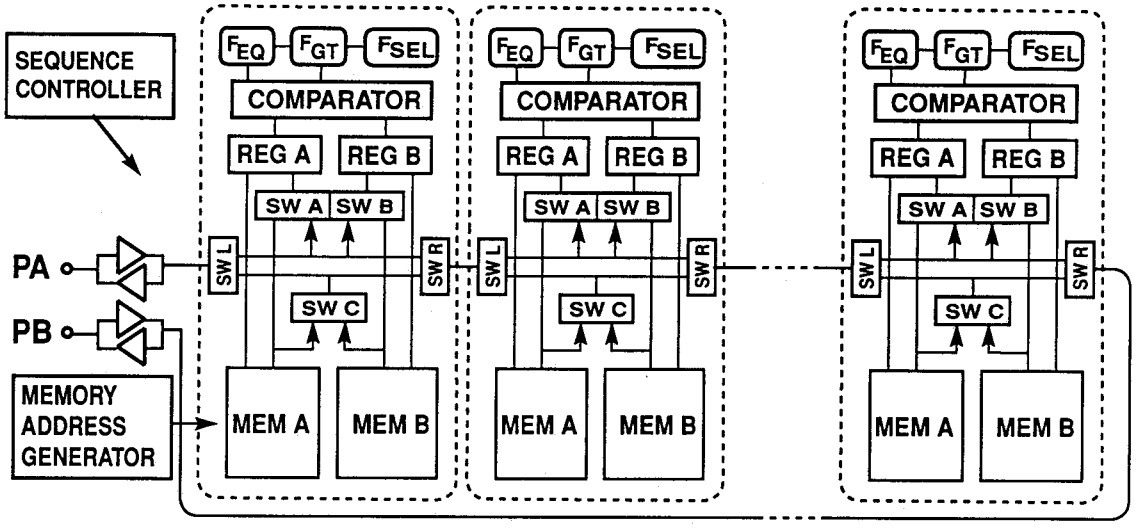


図 2.3: 多バイト化した比較ユニットとソートアレイの構成

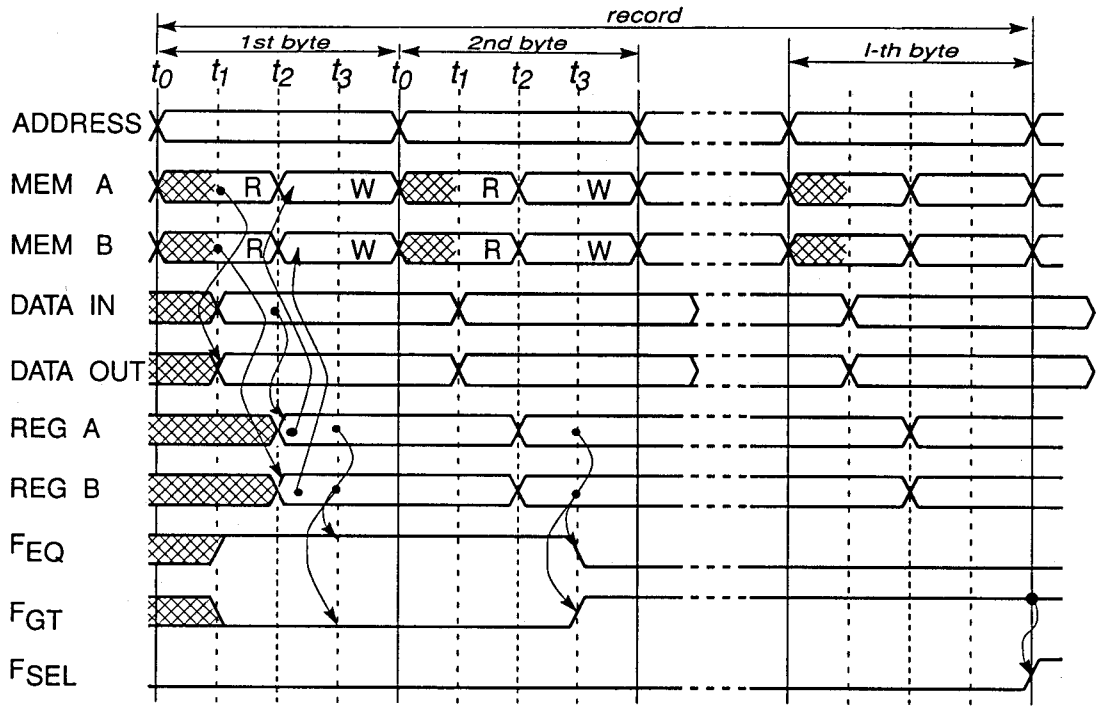


図 2.4: ソートアレーのタイミングチャート

果が一致していることを示している．第 $(i-1)$ バイトまでの比較結果が一致 ($F_{EQ}=1$) していて第 i バイトの比較で初めて大小関係が決まった時に，その比較結果をレジスタ F_{GT} に設定する．すなわち，レコードの先頭からの比較の繰り返し過程で，初めて大小関係が決定した時の結果，図 2.4の例では第 2 バイトの比較完了時の値を F_{GT} に設定する．レジスタ F_{GT} の値はレコードの比較が完了した時点，すなわち，第 l バイトの比較が完了して次のレコードの比較 (と転送) が開始される直前にレジスタ F_{SEL} に移す．レジスタ F_{SEL} の値で入出力回路を制御し，メモリ MEM A と MEM B に格納されたレコードの一方を選択して転送する．なお，転送回路 (SW L, SW R) は，レコードの比較結果とは関係なく，レコードの入力操作時と出力操作時とでレコードの転送方向を切替える．

以上示したように本ソートアレーでは，直前のレコードの比較結果で入出力回路を切り替え，切り替えた入出力回路を使った l 回の転送と比較とでレコードの大小関係を決定する．レコードの比較を l 回に分割して行なっているから，レコードの一部分だけをキーとしてソートすることも容易である．この場合は，キーとなる部分の比較結果だけをレジスタ F_{EQ}, F_{GT} に反映させるように比較器を制御する．

2.2.3 比較ユニットの連動法

上記のソートアレーでは，ソートできるレコードの最大長は比較ユニットのメモリ容量で，ソート速度は比較転送のデータ幅とクロック周波数で決定される．ここでは，比較ユニット間の接続を可変とし，複数ユニットを連動させてレコード長を拡大する方法と比較速度を向上させる方法を示す．複数ユニットを一括集積した大規模ソートアレーは，これらの拡張のために必要なユニット間の接続回路も同一チップ上に実装でき，アレーの再構成を容易に実現できる．

レコード長の拡大

ソートできるレコード長の上限は，比較ユニットを構成するメモリの容量に依存する．長大レコードをソートするために各ユニットのメモリ容量を大きくしたのでは，レコード長が短い場合にメモリを有効に利用できない．このため，ユニットのメモリ

容量は増やさずに、レコード長に応じて複数ユニットを連結して動作させる．長大レコードは，複数の連結されたユニットに分割格納され，それぞれを格納しているユニットで独立に比較を行い，比較結果をユニット間で累積しレコード全体の大小関係を決定する．

ユニット連結のために必要な回路は，ユニット群の中から１つのユニットを選択して順番に比較・転送動作をさせる活性化回路，ユニット群内で比較結果を合成する累積回路，活性化されていないユニットを飛び越してデータ転送するためのバイパス回路である．バイパス回路は，図2.3の転送回路(SW L/R)で実現できる．活性化回路と累積回路は，ソートアレーに入力されているレコードを分割して，対応する部分を比較転送するユニットだけを動作させる回路であり，このために付加する回路は僅かである．

比較速度の向上

比較速度の向上はレコード長ほど動的に変更できる必要は無いが，LSIとして実現したソートアレーの適用性を高めるには重要な機能である．ここでは，データの比較転送幅を可変として比較速度を変更する．例えば，１バイト幅の比較器を持つ比較ユニット２個を結合して，見かけ上(チップの外から)は２バイト比較を行っているように動作させる．

レコードは複数バイトが同時に入力され，連動しているユニットに例えば１バイトづつ分配される．従って，１個のレコードはユニット群を構成するユニットにサイクリックに格納され，連結の場合より細かい単位で比較結果をユニット間で累積する必要がある．このため，累積のタイミングは変える必要が出てくるが，累積回路自体の構成はレコード長拡張の場合と同一である．全ユニットは同時に動作しているから，活性化回路やバイパス回路は必要としない．ソートアレーの入出力端(PA, PB)は，十分な幅のデータ入出力回路と，連動するユニット数を変更した時のデータ転送経路を変更する回路が必要となる．転送経路の変更は，ソートアレーの端でのみ必要であり，比較ユニット間の接続に変更はない．

以上示した２つの拡張では，比較ユニット内部の構成および制御は変更する必要

がなく，連動するユニット間に，上述の比較結果累積回路と連動制御回路を追加するだけで実現できる．

2.2.4 ソートアレーの特徴と従来法との相違点

ソートアレーは，比較器の個数がレコード数に比例する $O(n)$ 比較器のソートアルゴリズムである奇偶転置法 (odd-even transposition sort) を並列化したバブルソート法 [38] を基本アルゴリズムとしている．小規模な回路 (比較ユニット) の規則構造で実現できるため回路の繰り返し性が高く，複数ユニットを一括集積する場合に以下の特徴が得られる．

(1) 全体 (チップ) の設計工数が一括集積するユニット数に依存せず，ほぼ比較ユニット 1 個分に等しい．

(2) データ転送が局所化されているから，単体ユニットの動作を検証すれば全体の動作を保証できる．

(3) チップの入出力端子数 (ピン数) が，集積規模に依存せず一定である．

一般に，集積可能なゲート数はチップ面積に比例するが端子数は辺長に比例するので，ゲート数より端子数を増やす方が困難を伴う．また，LSI の設計コストは非常に高く，上記ソートアレーの特徴 **(1)**, **(2)** は LSI 化する際に極めて有利な条件となる．一方，専用ハードウェア化する際の要件であるソフトウェアとの親和性の観点からは，

(4) レコード数，レコード長が変わってもハードウェアが有効に動作する．

ことが重要である．比較ユニット間の接続を可変にしてレコード長拡大やソート速度向上を実現するソートアレーは，この要件を満たしている．

以上の観点から，これまで提案されている $O(n)$ 比較器のソータを比較する．ソートアレーと同様に奇偶転置法を並列化したソータに反転ソータ [10] と昇降ソータ [46] がある．これらのソータは磁気バブリメモリに比較機能を組み込むことを想定しており，条件 (4) のレコード数やレコード長に関する拡張性は考慮されていない．

Miranker らは，筆者らとほぼ同時期にゼロタイムソータ [52] を提案している．これは Logic-in-memory の考えに基づいて，比較器と 2bit のメモリセルからなる 'Dibit' セルを単位として繰り返し構造を実現している．'Dibit' セルを幅方向と深さ方向とに接続することで，所望のレコード数とレコード長でのソートを実現している．この方法ではセル間の接続が密 (2 次元) になり，条件 (4) を満たす柔軟な構成を実現することは困難といえる．

奇偶転置法とは異なる $O(n)$ 比較器のソータに並列計数ソータ [84] がある．これは，レコードの入力時にランク (順位) を決定し，入力完了時に求まるランクに基づいてレコードを置換し出力する．ランクを決定する際に入力レコードを入力済の全レコードと比較するので，全てのユニットに入力レコードの値をブロードキャストするバスが必要である．これは大規模一括集積の条件 (2), (4) を十分に満たすことができない．

2.3 大規模ソートアレーの実現法

規則的な繰り返し構造を持つソートアレーは，LSI 化した時の入出力端子数が一括集積するユニット数に依存しなため，冗長構成による欠陥救済により大規模一括集積が行ない易い．大量にあるユニット間接続配線を含めて多くの比較ユニットを一括集積することで，装置の小型化と高信頼化を実現することができる．しかし，論理回路が主体のソートアレーに従来から知られた k-out-of-n 冗長構成を単純に適用した場合，切り替え単位となる回路を任意に細分化できない問題や，予備数の増加に伴って切り替え回路量が増加する問題があり，巨大集積を良好な良品率で実現することは困難であった．

そこで，繰り返し構造を持つ論理回路の階層構造に着目して，後述する階層化冗長構成法 [60, 61, 82] を用いてそれぞれの階層に予備回路を配置し，チップ全体としては少ない予備回路量で高い良品率を達成することを考える．比較ユニット間の接続を可変としたソートアレーの機能上の階層構造には，比較ユニット，複数の比較ユニットを連結 / 結合させるユニット群がある．さらに，比較ユニット内の主要回路をビットスライス構成とした比較セルの階層を設けることができるため，容易に 3 階層までの階層化冗長構成法を適用することができる．しかし，高い良品率を得るために予備

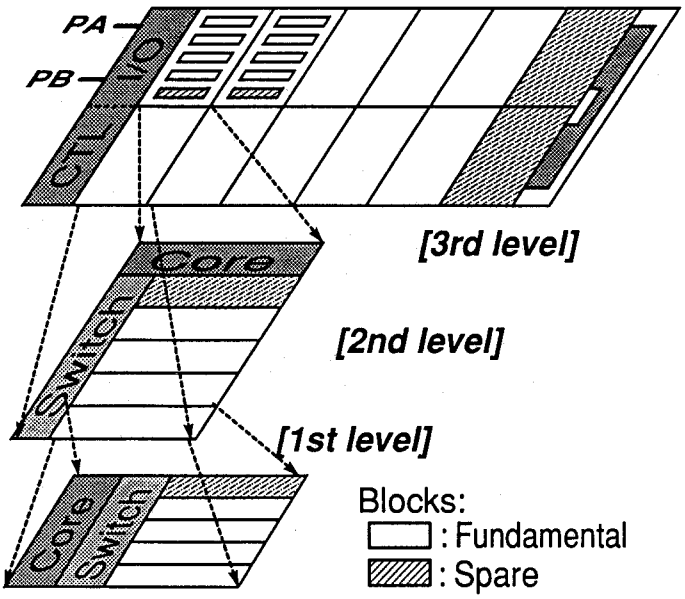


図 2.5: 階層化冗長構成法を適用したソートアレーの構成

を増やすと相対的にチップ上に占める基本回路の比率が低下する問題があった．以下では，$4cm \times 2cm$ の巨大チップに基本回路として 40 個の比較ユニットを大規模一括集積する場合を例に，階層数や個々の階層における予備の配置量を最適化する方法について述べる．

2.3.1 階層化冗長構成法

筆者らが提案している階層化冗長構成法 [60, 61, 82] は，LSI を構成する論理回路の繰り返し構造に着目して k-out-of-n 冗長構成を階層的に設定し，欠陥を含む切替単位を予備に切り替える操作を各階層毎に実施する方法である．3 階層の冗長構成をソートアレーに適用した場合の例を図 2.5に示す．本方法では，下位階層における切替操作によって上位階層の切替単位が無欠陥と等価になる確率が高くなるため，全体として少ない冗長度 (全体に占める予備の割合) で高い欠陥救済効果 (良品率改善効果) が得られる．また，下位階層の切替回路は上位階層の切替単位に含まれるので，最上位階層を除く下位層の切替回路も欠陥救済の対象となる．

チップ内に発生する欠陥がランダムかつ孤立欠陥であると仮定すると，面積 A の

回路の良品率 Y は欠陥密度 λ を用いて

$$Y = \exp^{-\lambda A}$$

で表される．ゲート密度 γ をチップ内で一様とすると $A = G/\gamma$ であるから，上記の良品率 Y はゲート数 G の関数となる．m 階層の階層化冗長構成を適用した LSI の良品率 Y_m は，第 $i(1 \leq i \leq m)$ 階層の良品率 Y_i の漸化式で与えられる．

$$Y_i = \sum_{j=k_i}^{n_i} \binom{n_i}{j} Y_{i-1}^{j}(1 - Y_{i-1})^{n_i - j} \times Ycore_i \times Ysw_i$$

ここで，Σ の項は第 $i-1$ 階層までの冗長切替で良品率 Y_{i-1} となった切替単位を k_i-out-of-n_i 冗長構成した良品率，$Ycore_i$ は第 i 階層で新たに追加されたコア回路 (制御回路や入出力回路等) の良品率，Ysw_i は k_i-out-of-n_i 冗長構成を実現するための切替回路の良品率である．最下位階層の切替部品の良品率 Y_0，各階層の $Ycore_i$，Ysw_i は，それぞれを構成する回路のゲート数の関数として求めることができる．以上により，LSI の良品率 $(= Y_m)$ は，最下位階層の切替単位のゲート数，および，各階層のコア回路と切替回路のゲート数から算出できる．

階層化冗長構成では，予備の配置，すなわち，階層数の決定と各階層における基本ブロック数と予備ブロック数を最適にする方法が課題となる．ここでは，チップの冗長度 η を次式で定義し，より少ない冗長度で高い良品率を達成できる冗長構成を求める方法を提案する．

$$\text{冗長度}\eta = \frac{\text{チップ内の予備回路の面積}}{\text{チップ内の基本回路の面積}}$$

冗長度 η は，階層的に配置されたコア回路と切替回路の面積を無視すれば，最下位階層の切替単位が，チップ内で予備回路として配置された個数と基本回路として配置された個数の比で近似できる．冗長構成の目的は，単位面積中に最も多くの良品となる基本回路を実現することであり，ウェハ上に占める良品チップの割合を示す収率に相当する有効利用率を最大とする構成が最適冗長構成といえる．

$$\text{有効利用率} = \frac{\text{チップの良品率}}{\eta + 1}$$

2.3.2 大規模一括集積の適用

比較ユニット間の接続を可変としたソートアレーに階層化冗長構成を適用する．最初に，ソートアレーの機能に着目して冗長構成を適用できる階層を抽出する．比較ユニットは，ソートアレーの機能を実現する主要な回路ブロックであり，複数の比較ユニットを連動させるユニット群も回路ブロックとして抽出できる．更に，個々の比較ユニットは，メモリ，レジスタ，比較器をビットスライス構成として実現できることから，比較セルとして抽出することができる．したがって，比較セル，比較ユニット，ユニット群の 3 階層で構成できる考えられる．

次に基本ブロックの配置を考える．3 階層の冗長構成をソートアレーに適用した場合の構成を 図 2.5に示す．第 1 階層 (最下位層) は，比較セルを切替単位として k_1-out-of-n_1 冗長切替を適用する．メモリのアドレス生成回路や比較結果の累積回路等は，上位階層のコア回路とする．比較ユニットの入出力データ幅を 1 バイトとすると，比較ユニットを構成する比較セル数 k_1 の取り得る値は 1, 2, 4, 8 である．第 2 階層は比較ユニットを切替単位とするユニット群であるから，連動するユニット数を考慮して k_2 の取り得る値は 2, 4, 8, $\cdots$ である．レコード長の拡張や比較速度の向上のために付加する制御回路は上位階層のコア回路とする．第 3 階層以上の構成に基本的な制約はないが，ここでは，4 インチウェハを仮定し面積 4cm × 2cm チップ 6 個を同一ウェハに実装するとして，一括集積する WSI 規模を設定する．チップ面積と (平均的な) ゲート密度から，搭載できる比較ユニット数を概算できる．

以上の制約のもとで LSI チップの良品率を算出した結果を図 2.6に示す．図は，ソートアレーチップの良品率を階層数をパラメータとして示してある．横軸 $\eta = 1$ の時に基本ブロックと予備ブロックの面積が等しい．この結果から，階層数が小さい場合は冗長度を大きく，すなわち予備を増やしても良品率の向上が期待できないのに対して，階層数を増やすと高い良品率が得られることがわかる．例えば，1 階層の冗長構成で達成できる最大の良品率は 30% 程度であるが，3 階層とすることで 90% 近い良品率を達成できることがわかる．

図 2.6の結果から明らかなように，階層化冗長構成法は冗長度が大きい領域で高い

良品率を達成する方法といえる．しかし，冗長度を大きくするとチップ上に占める予備の割合が増え，搭載できる基本ブロック数の減少を招く恐れがある．そこで，一定面積のウェハ上に実現できる良品ブロック数の割合を示す有効利用率を評価する．この値が高いほど一定面積 (チップ) 上に良品となる比較ユニットが多いことを示している．図 2.7は冗長度を変化させて求めた有効利用率である．図中の破線は，冗長構成によって良品率 100% が達成された場合でも超えられない有効利用率の上限値を示している．例えば，冗長度が 1 の場合，良品率 100% でも基本ブロック数に等しい予備ブロックがあるので有効利用率は 50% となる．図 2.7の結果から，冗長度 $\eta = 1.1$(3 階層) が有効利用率を最大とする最適冗長構成といえる．この時の良品率は図 2.6から約 80% である．

チップの形状を 4cm × 2cm に近くする等のレイアウト上の制約を考慮すると，最適な冗長構成が取れない場合がある．比較ユニットを 2 ビットスライス構成として，40 個の基本ユニットを一括集積する際の冗長構成は，

第 1 階層 : 4-out-of-5

第 2 階層 : 4-out-of-5

第 3 階層 : 10-out-of-12

であり，この時の冗長度は 0.88 である．本構成に基づいて設計したソートアレーは，基本回路として 40 個の比較ユニットを，予備を含めて $37mm \times 21mm : 780mm^2$ に実現できた [80, 61].

2.3.3 考察

ソートアレーでは，比較ユニットのビットスライス構成，比較ユニットの連結・連動を実現する階層に着目して冗長構成を階層的に適用した．この場合，各階層の基本ブロック数として取り得る値が制限されているので組み合わせ数が少なく，短時間で良品率あるいは有効利用率を最大にする構成を決定できた．有効利用率は，図 2.7の破線で示すように上限値が明らかであるから，予備ブロック数を増やしていって冗長度がある値を超えたら計算を打ち切ることができる．階層数についても同様に上限値

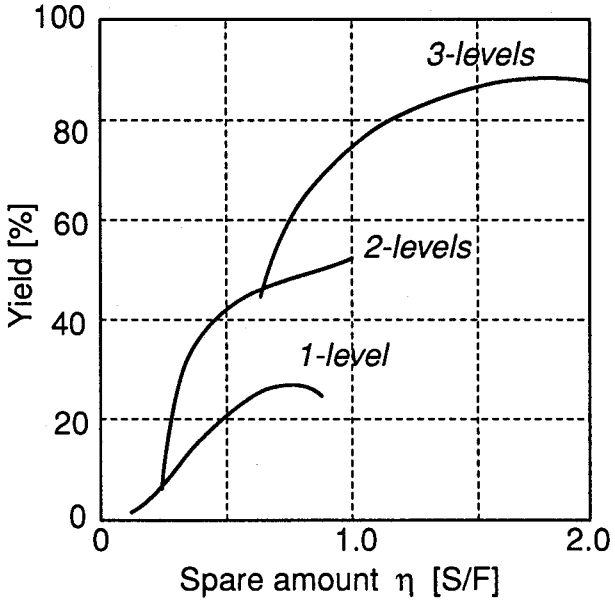


図 2.6: ソートアレーチップの良品率

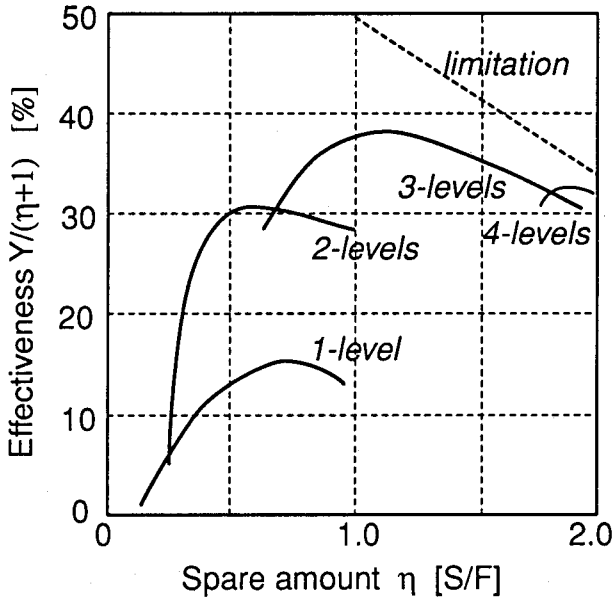


図 2.7: 階層化冗長構成法の欠陥救済効果

が決まるため，計算すべき冗長度の上限を簡単に求めることができる．可能な冗長度の範囲内で所望の有効利用率，あるいは，良品率を達成することができない場合は，チップに搭載しようとする基本ブロック数が多すぎると考え，一括集積する回路規模を削減して，再度，有効利用率を評価する．

以上示したように，有効利用率を用いた冗長構成の設計法は，階層的に配置する基本ブロックの数にある程度の制約があると，比較的少ない組み合わせ数の冗長構成を評価することで最適化を図ることができるが，基本ブロックの配置に全く制約がないと組み合わせ数が増大して最適化に時間がかかることが予想される．一方，冗長切替を行なう為には，各階層の回路ブロックが正常に機能するかを短時間で判定することが重要になる．この問題に関して筆者らは，個々の回路ブロックの機能に着目した試験回路をブロック内に内蔵し，下位階層のブロックの簡単な試験を並列に行なった後に冗長切替を行ない，更に，冗長切替で構成した上位階層のブロックを詳細に試験する2段階テスト法 [81] を実現し，短時間で行なえる効率的な試験法を報告している．

階層化冗長構成法を一般的な論理回路に適用した場合には，最上位階層のコア回路を欠陥救済の対象とできないことが問題になることもある．ソートアレーで問題となるコア回路は，1バイトあるいは2バイト幅のレコード入出力回路，複数ユニットを単位とした飛び越し接続のための回路 (バイパス) とその制御回路であるが，設計結果では 1k ゲート以下と少なく大きな問題とならなかった．この部分の回路量が多い場合は，コア回路に2重化等の他の冗長構成を適用するなどの改良が必要になる．

2.4 まとめ

本章では，典型的な細粒度並列を実現する1次元アレイ構造からなるソートアレーの構成法を述べた．シストリックアーキテクチャに基づいて奇偶転置法を並列化したソートアレーは，1組みのメモリと比較器とからなる比較ユニットをアレイ状に接続し，全てのユニットで同期した比較と転送動作を繰り返すことでソートを行なう．1バイト幅の比較転送を行うソートアレーでは，l 回の比較転送動作によって長さ l バイトのレコードをソートできる．

このような機能を実現した比較ユニットを，比較ユニット間の接続を可変構造として一括集積し，複数ユニットが連動するようにユニット群を形成する．比較ユニットのメモリ容量が増大する方向と比較器の幅が増大する方向の 2 方向にユニットを連動させ，ソートできる最大レコード長とソート処理速度を拡張可能とする．

規則的な繰り返し構造からなるソートアレーは，冗長構成による欠陥救済が容易であり，冗長構成のための再構成可能な構造と上記の機能拡張のための可変構造を同時に満足するソートアレーの構成法を明らかにした．論理回路の機能的な階層構造に着目して個々の階層に予備を設ける階層化冗長構成法は，予備を増やして冗長度を上げると WSI の良品率を向上できるが，相対的にチップ上に占める基本回路の比率が低下する問題があった．予備回路量と基本回路量の比である冗長度と達成できる良品率とから算出したチップ面積の有効利用率を用いて，有効利用率を最大にすることで最適な予備の配置と WSI 規模の最適化が図れることを示した．3 階層の冗長構成を適用した冗長度 0.88 のソートアレーチップは，基本ブロックとして 40 個の比較ユニットを良好な良品率で大規模一括集積できた．

第 3 章

大容量データベース処理に適したソータ構成法

3.1 まえがき

本章では，前章で述べたソートアレーを基本構成要素とし，ソートできる最大レコード数とソート処理速度に柔軟に対応できる，大容量データベース処理に適したソータ構成法 [62, 65] を論じる．

前章でも述べたように，ソートは関係データベースの結合処理等で多用される非数値処理の基本演算であり，ソートを高速化する種々の専用ハードウェアの構成法が提案されている [1]．関係データベースでは，数個から数百万個の広範なデータ量を効率よくソートすることが求められるため，専用ハードウェア化するにあたり以下の要件を満足する必要がある [32]．

a) 計算時間が回路サイズの関数でなく，データサイズの関数となるような回路であること．

b) 高速化要求はデータサイズ n が大きい領域で大であるから，回路サイズは n 以下 $(n, \sqrt{n}, \log n,$ 定数等$)$ であること．

c) どんなに大きなハードウェアを用意しても扱いきれない場合が必ずあるので，ソフトウェア的方法との併用が容易であること．

専用ハードウェアによるソータの研究は盛んに行われていたが，従来のソータは，並列比較による処理時間の短縮に主眼が置かれていたため，ソータのハード量が，ソート対象とするレコード数 N に対して $O(\log_2 N)$[79, 77, 30, 35]，または $O(N)$[44, 52,

84] となる構成，すなわち，ハード量がレコード数に依存して増加する構成が一般的であった．これまで報告されている，関係データベース処理への適用を主な狙いとしたソータ [30, 35] は，2-way マージを基本アルゴリズムとしてハードウェア化したマージ回路を多段接続している [38]．この構成は，計算時間が $O(N)$ で回路サイズが $\log_2 N$ であるから，上記の要件 a)，b) を満足しているといえる．しかし，$\log_2 N$ の回路サイズでは，現実的な回路規模で実現できるソータ容量は高々 2^{20} 程度であること，要件 c) のソフトウェアとの併用で回路の複雑さが増大することが課題として残されていた．

例えば，100 万個のレコードをソートするには，$O(\log_2 N)$ 系の代表的なソータであるパイプラインマージソータ [35] でも 20 段のマージ回路が必要となり，各マージ回路ごとに容量の異なるワークメモリを分散して実装する必要もあって，小形化を阻む要因となっていた．マージ回路数で決まる個数を超えるレコードをソートするための制御を実現したソータ [30, 2] や，種々のレコード長に対応できるソータ [35, 19] の構成が報告されているが制御の複雑化が避けられなかった．同様の狙いでトリーセレクションソート (トーナメント法) をハードウェア化したソータにシストリックソータ [17] があるが，データベース処理への応用を考えるとレコード数やレコード長に柔軟に対処するためのロジックが必要となり，回路の複雑化は避けられなかった [39]．

本章で提案するソータ構成法は，従来のソータが 2-way マージ回路を多段化していたのに対し，$\lceil k/2 \rceil$ $(k \gg 2)$ 個の比較器を 1 次元配列したソートアレー [60] で k-way マージャを実現することを考える．この構成では，回路サイズを決定するマージウェイ数 k はレコード数 n と独立に設定でき，逐次的な繰り返し操作で大容量ソートを行なうので要件 c) のソフトウェアとの親和性を高くできる．本方法を適用した k-way マージソータは，ソート対象となるレコードを格納するワークメモリと，1 次元アレー構造の並列比較回路 (ソートアレー) で構成され，マージするレコード列の選択にデータ駆動形制御を適用することで，数十ウェイのマージ処理をウェイ数に依存しない速度で高速に行える．このマージ処理を逐次的に繰り返すことで大量なレコードをソートする．

ソートできる最大レコード数は，集中実装された 1 つのワークメモリの容量で決

まり，比較回路や制御回路の構成に影響されない．ワークメモリの容量とレコード長で決まるレコード数をソートできることから，短いレコードはより多くソートできる．一方，ソート可能な最大レコード長は，第2章に示したように，比較ユニットの連結によって拡張する．1次元アレー構造のソートアレーは，ＶＬＳＩ技術との適合性が高く，複数の比較器を一括集積することで大幅な小形化がはかれる．

以下，**3.2**では，マルチウェイマージ法とハードウェア構成について述べる．**3.3**では，大容量ソータを実現するための逐次多段マージソート法と繰り返し使用するワークメモリへのレコード列格納法，および並列比較を行なうソートアレーの構成法を示す．**3.4**では，試作ソータの構成と性能について評価結果を示す．なお本章では，データベース処理におけるソート条件となる列の属性値を抽出して連結したキーをレコードと称する．

3.2 マルチウェイマージ法のハードウェア化

3.2.1 ハードウェア化の課題

データベース処理におけるソートは，磁気ディスク等の二次記憶装置に格納された表に対して，選択等の処理を行った結果に対して行う．このため，ソートするレコードの個数をあらかじめ知ることができない．ソートすべきレコードの個数は，データベースの規模や問合せの内容に大きく依存するので，レコード数が少ない場合でもソータのハードウェアが有効に機能する必要がある．また，ソートの高速化を必要とするのは大規模データベースであり，極めて大量のレコードをソートできなければならない．

ソータは従来より種々の形式のものが提案されているが，データベース処理でよく用いられるものは，レコード数Nに対して回路サイズが$\log_2 N$に比例する$\log_2 N$型ソータ [79, 77, 30, 2] である．しかし$\log_2 N$型ソータをデータベース処理に適用するには以下の問題点がある．

(1) ハードウェア量: $\log_2 N$型ソータは，ソートできる最大レコード数がハードウェア構成に大きく依存し，実用的な規模のソータを実現するにはハードウェア量

が多い．例えば，100 万個のレコードのソートを可能とするには，1LSI/ 比較器として 20 個の LSI が必要である．1 枚の基板に搭載可能な LSI 数を考慮すると，LSI 数を数個に抑えることが望ましい．

(2) **拡張性:** 製品化を前提に考えると，製品ランクによってソート可能なレコード数を容易に変えられる必要がある．$\log_2 N$ 型ソータでソート可能なレコード数を変更するには，ソート用ワークメモリ量の変更に加え比較器 LSI 数の変更も必要である．種々のランクの製品への対応を容易にするには，ワークメモリ量の変更のみに抑える必要がある．

これらの問題を解決するため，単純マルチウェイマージ[1]を繰り返して行うことによりソートを実現する．本ソート法は，従来のソフトウェアでのソートマージ法をベースとしており，マージ準備段階でのレコードのソート，中間マージ，出力マージ段階でのマージを同一のハードウェア (ソートアレー) で高速に実現できるところに特徴がある．本ソート法を用いることで上記の問題は以下のように解決できる．

(1) に対して： マージの繰り返しでソートするので，大量レコードのソートも一定規模の比較ハードウェア (ソートアレー) で実現できる．ソートアレーは，ハードウェア規模の小さい比較ユニットの 1 次元アレーであるから，容易に複数ユニットを一括集積できる．この結果，数個の LSI で数 10 ウェイのマージを実現でき，これらの LSI を用いてワークメモリが許す限り，いかなる大量レコードでもソートできる．

(2) に対して： マージを繰り返してソートするから，ソート可能なレコード数の変更は，ワークメモリ量の変更だけで対応可能である．ワークメモリは集中して実装できるから，大容量のダイナミック RAM チップを用いて高密度に実装できる．

以下では，レコードの比較回路 (ソートアレー) とレコードの格納回路 (ワークメモリ) を分離することで，ソートできる最大レコード数とソート速度の 2 つの独立した

[1]本章のマージはソートの手段として複数のソート済みレコード列を 1 本にする操作であり，次章で述べる 3 フェーズ・ソートマージ結合法におけるマージとは異なる．

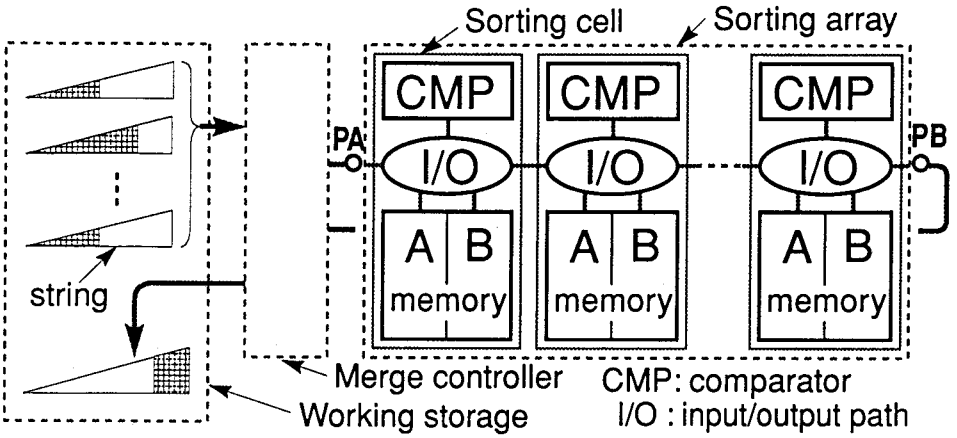


図 3.1: マルチウェイマージソータの基本構成

要求条件を同時に満足するマルチウェイマージソータのハードウェア構成とアルゴリズムを詳述する.

3.2.2 ハードウェア構成

レコードを並列比較するソートアレー，レコードを格納するワークメモリ，およびマージ制御回路からなるソータの基本構成を図 3.1に示す．ソートアレーは，2個のレコードを格納して大小関係を判定する比較ユニットを1次元アレー状に配置してある．ワークメモリは，$\log_2 N$ 型ソータと異なり全体を集中実装できることから，市販の大容量 RAM チップを用いて汎用計算機の主記憶装置と同様に極めて高密度に実装できる．マージ制御回路は，$k(k \geq 2$ の整数) 本のレコード列をマージする k ウェイマージ制御と，最終的に1本のレコード列するための k ウェイマージ処理の逐次繰り返しを制御する回路である.

ソートアレーの並列比較動作は，図 2.2(第 2章参照) に示したように，全ての比較ユニットで同期して比較操作と比較に基づく転送操作を行なっている．この結果，ソートアレーに入力されたレコードの内で最大なレコードは，常に，ソートアレーの最左端のユニットに保持され，レコードの入力 / 出力回数，あるいは，それらの順序関係に関わらず待ち時間なしで取り出せる.

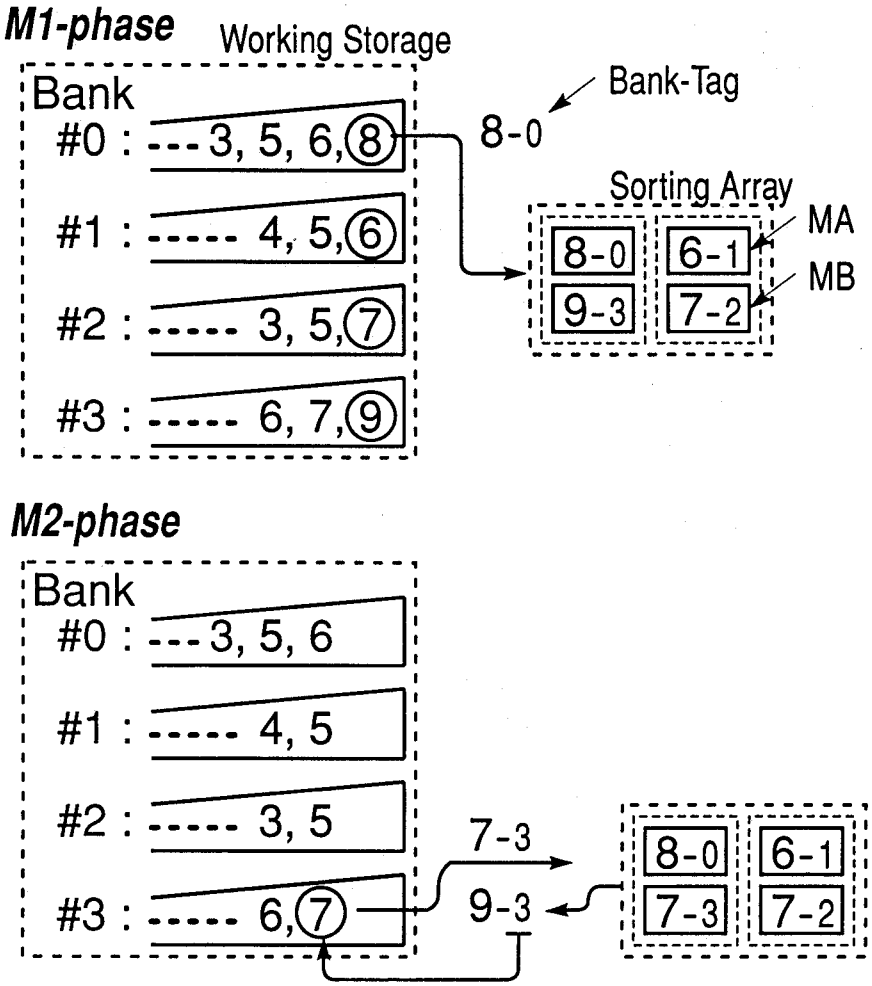


図 3.2: バンクタグを用いたマルチウェイマージ制御

3.2.3 マージアルゴリズム

マージウェイ数を大きくすることによって，少ないマージ段数で大量のレコードをソートできる．マージウェイ数を拡大しても制御が複雑にならず，高いスループットを維持できるバンクタグを用いたデータ駆動形マージ制御法について述べる．

バンクタグを用いた k ウェイマージ制御を，4 ウェイマージを例として図 3.2を用いて説明する．ワークメモリに格納された 4 本のレコード列には，#0 から #3 のバンク番号が付与してある．

M1 フェーズ： マージ対象とする各レコード列の最大レコード，すなわち各レコード

列の先頭レコード (図で○印を付けたレコード) に，そのレコードが属していたレコード列を識別するバンクタグを付与して，ソートアレーに入力する．

M2 フェーズ： ソートアレーから 1 個のレコードを出力し，出力したレコードのバンクタグが指すレコード列から次のレコードを読み出してソートアレーに入力する．このマージ操作を全ての入力レコード列が空になるまで繰り返す．

最初の出力レコードは，M1 フェーズの終了時点で最左端のユニットに保持されているレコード，すなわちマージするレコード列内の最大レコードである．出力された最大レコードの次に大きいレコードの候補は，ソートアレー中に残存する $(k-1)$ 個のレコード，あるいは，出力した最大レコードが属していたレコード列中の次のレコード (現時点での先頭レコード) であるから，上述のマージ操作の繰り返しで全体をマージできる．

k ウェイマージ処理の流れを制御するバンクタグは，レコードの大小比較に影響を与えないように，例えばレコードの最後部に付与することで，ソートアレー内ではレコードと一体として扱うことができる．任意のウェイ数 k でマージするために必要なバンクタグの最小ビット数は $\lceil \log_2 k \rceil$ と小さく，$\lceil k/2 \rceil$ 個の比較ユニットを持つソートアレーを用いて，大きなウェイ数 k のマージ処理を少ない制御オーバヘッドで容易に実現できる．

3.3 逐次多段マージソータの構成法

データベース処理に適用するソータは，検索結果の一時表を入力とすることから，ソート対象となるレコードの個数および長さに柔軟に対応できなければならない．本節では，このような要求を満足できる逐次多段マージの制御法を示す．本ソータでは，ワークメモリに格納された複数のレコード列をマージして再度ワークメモリに格納する処理を段階的に繰り返す．ワークメモリに効率よくレコード列を格納する方法，および，ソートアレーを有効利用するための往復ソート機能の実現法を詳述する．

3.3.1 多段マージ制御法

マルチウェイウェイマージ法は，3段あるいは4段のマージ処理で実用上十分な個数のレコードをソートできる．このため，従来のパイプラインマージソータのようなハードウェア的な多段化ではなく，逐次的なマージ処理で高速なソータを小形に実現できる．逐次多段マージソートは以下の3段階からなる．

マージ準備段階 (Pre-merge Stage)：入力される一連のレコードを，k 個ずつソートアレーに入力／出力して，大きさ k のソート済みレコード列としてワークメモリに格納する．入力が完了すると，ワークメモリには $\lceil N/k \rceil$ 本のレコード列が格納される．ここで，レコード列の大きさとは，レコード列に含まれるレコードの個数である．

中間マージ段階 (Intermediate Merge Stages)：ワークメモリに格納されたレコード列を k 本ずつマージして，再度ワークメモリに格納する k ウェイマージを段階的に繰り返す．1段の中間マージ処理で，レコード列の大きさは最大 k 倍に，レコード列の本数は最大 $1/k$ となる．レコード列数が k 以下となるまで中間マージ段階の処理を継続する．

出力マージ段階 (Output Merge Stage): 中間マージ段階までの処理で，k 本以下となったワークメモリ中のレコード列をマージして出力する．

以上示したように，逐次的なマージの繰り返し処理でソートを実現することで，ソートできる最大レコード数と無関係にソータのハードウェアを構成できる．N 個のレコードをソートするのに必要な全マージ段数 (ステージ数) は $\lceil \log_k N \rceil$ である．マージウェイ数とレコード数の関係を，ステージ数をパラメータとして図3.3に示す．この図から，マージウェイ数を増やすとより少ないステージ数で大量のレコードをソートできることが判る．また，N 個のレコードのソート処理時間，すなわち，最初のレコードの入力開始から最後のソート結果出力までの時間は，各ステージの処理がソートアレーによる並列比較で $O(N)$ で実現されることから，この時間はステージ数に比例する時間 $O(N \log_k N)$ となる．

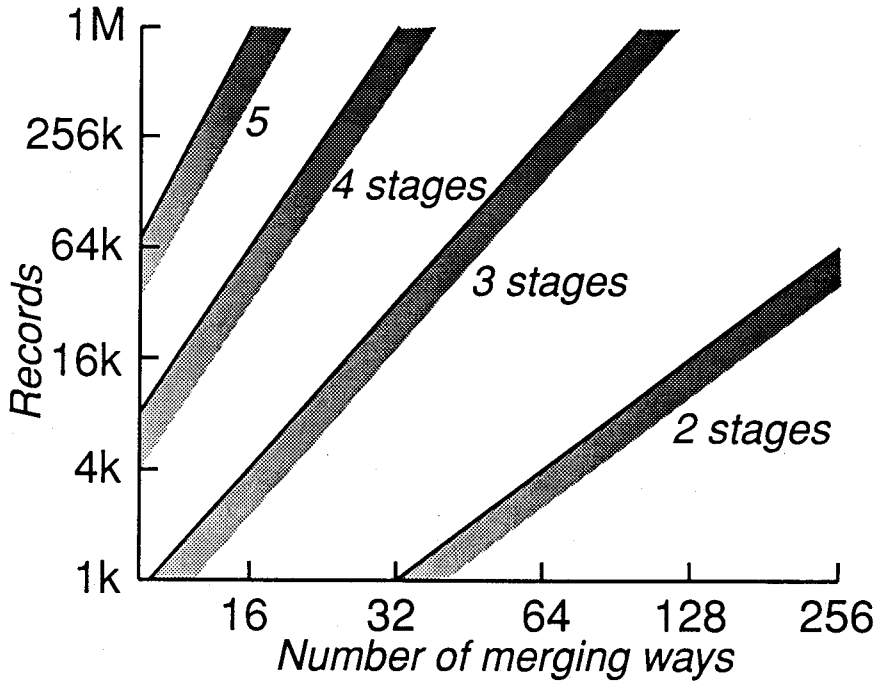


図 3.3: 逐次多段マージによる大容量ソート

一般に，マージ準備段階および出力マージ段階の処理は，データベースを管理しているホスト計算機等とソータとの間のレコード転送時間に重畳して処理できことから，レコードの入力完了から出力開始までのソータ保留時間は $N \times (\lceil \log_k N \rceil - 2)$ となる $(N \geq k^2)$．図 3.3から明らかなように数十ウェイのマージであれば，このソータ保留時間は 1 段，最悪でも 2 段の中間マージ処理時間である．

3.3.2 レコード列の格納法

中間マージ段階では．ワークメモリから読み出した複数のレコード列をマージして再度ワークメモリに格納する．このため，ワークメモリを有効に利用できるメモリ管理法が必要となる．以下では，ソートするレコードの個数を N，長さを L として，メモリ利用率 $(= LN/\text{メモリ容量})$ を比較評価する．

これまで知られているメモリ管理法には，2 重メモリ法，ポインタ法，ブロック分割法がある [36]．ワークメモリを 2 つの領域に分けて使用する 2 重メモリ法は，制御は簡単であるがメモリ利用率は最大で 1/2 である．各レコードにポインタを持たせ

て空き領域を管理するポインタ法は，固定長レコードであれば制御は極めて簡単であるが，レコード長が短いと利用率が低下する．メモリ利用率は $L/(L+$ ポインタ長$)$ である．ブロック管理法は，固定サイズのブロック単位で空き領域を回収・再利用し，ブロック内はポインタチェインで管理する方法であり，2 重メモリ法では 2 分割としていたワークメモリの分割を細分化することでメモリ利用率を向上した方法といえる．この方法は，マージウェイ数 k に等しい補助ブロックを必要とし，ウェイ数が小さい領域で有効な手法であった．

そこで，逐次多段マージソータのメモリ管理法として，マージウェイ数を大きくした場合にメモリ利用率を $k/(k+1)$ に向上できるエリア格納法を検討した．エリア格納法を用いた逐次多段マージ処理の手順を図 3.4に示す．マージ準備段階では，入力レコードをワークメモリの下位方向から，大きさ k のレコード列として順次格納していく．1 つのレコード列を連続領域に格納することで，メモリ管理を容易にしている．マージ準備段階の終了時には，レコード列を格納した上方に補助領域がある．中間マージ段階では，補助領域が分断されないように奇数段と偶数段とで交互にワークメモリの上端と下端から連続的にレコード列を格納する．

補助領域は，k 本の入力レコード列とは別の領域に，出力レコード列を格納するために使用する．その大きさは，ワークメモリに格納するレコード列の最大の大きさに等しい．すなわち，最終の中間マージ段階の出力レコード列で最大なものの大きさに等しい．

補助領域の大きさを最小にする条件は，最後の中間マージ段階で生成する出力レコード列を大きさが均一な k 本にすることである．中間マージの最終段で k 本の大きさが均一なレコード列を生成するには，その 1 つ前のマージ段階で，大きさが均一な k^2 本のレコード列を生成すればよいから，結局，初段の中間マージ処理で k のべき乗本の大きさが等しいレコード列をワークメモリ上に生成することに帰着する．マージ準備段階の終了時点で，入力レコードの個数は確定しているから，初段の中間マージ処理で，最悪でも大きさが k だけ異なる k のべき乗本のレコード列を生成できる．一般に $N \gg k$ であること，レコード列の大きさの差 k を保ったままで中間マージが行えるように初段の中間マージを制御することができるから，これ以上の均一化は不要

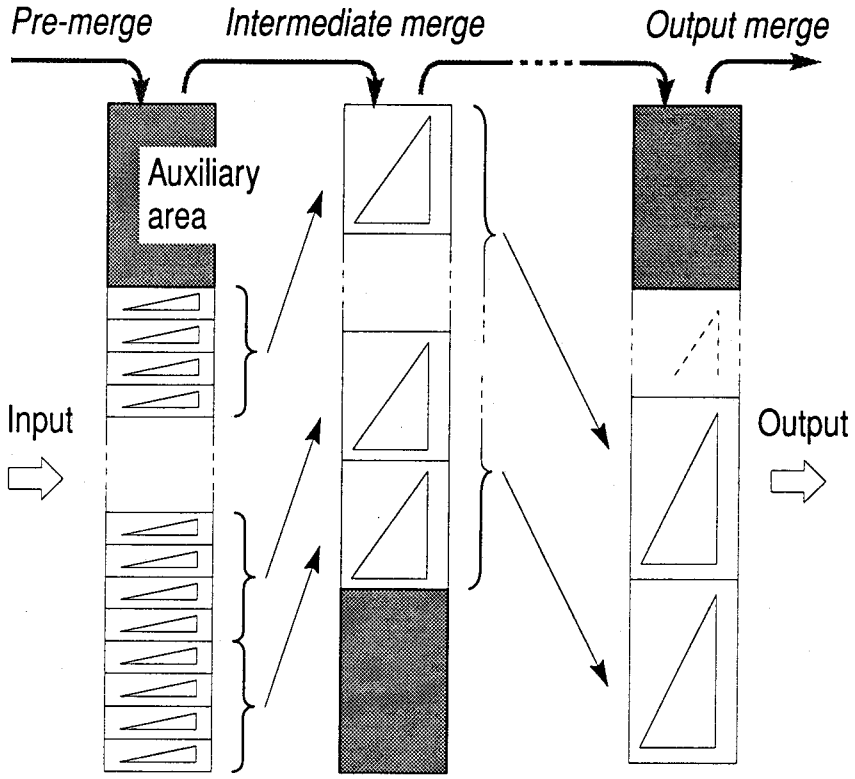


図 3.4: エリア格納法を用いた逐次多段マージ処理

といえる.

3.3.3 往復ソート法

マージ準備段階の処理は，k 個のレコードを連続してソートアレーに入力した後に，k 個のレコードを連続に出力する操作の繰り返しである．このため，レコードの入力と出力が間欠的になる．この問題を解決する，マージ準備段階の処理時間を半減する往復ソート法を示す.

往復ソートとは，ソートアレー (図 3.1参照) のポート A(左端)/ ポート B(右端) からソート結果を出力する操作と，ポート B ／ポート A に新たなレコードを入力する操作とを，交互にかつ重畳して行なうことである．この機能を実現するには，ソートアレー内でポート A から入力したレコードとポート B から入力したレコードとを分離する必要がある．このための従来技術として，入力レコードにポート A，B のいずれか

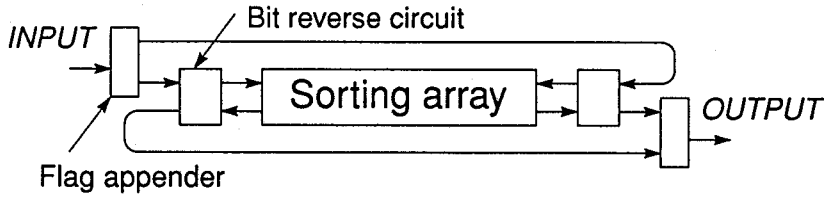


図 3.5: 往復ソート機能を実現するハードウェア構成

表 3.1: 往復ソート機能を実現するレコード修飾法

ソート順	制御	ビット反転回路	
	フラグ	ポート A	ポート B
昇順	0	反転	非反転
降順	1	非反転	反転

ら入力したかを示すタグを付与し，1 次元アレー状に配置した各比較回路で判定する手法がある [52] が，比較回路の制御が複雑になりスループットの低下が避けられなかった．このため，ソートアレーの入出力端に，レコードを修飾するハードウェアを付加する事で往復ソート機能を実現する．

上記機能を実現するハードウェアの構成を図 3.5に示す．レコードは，その先頭位置 (MSB 側) に制御フラグを付与した後に，ビット反転回路を介してソートアレーに入力する．ビット反転回路は，ソートアレーの入力／出力レコードを制御フラグを含めてビット反転する機能を持つ．制御フラグとビット反転回路によって，ポート A から入力したレコードを，ポート B から入力したレコードより，常にその値が大きくなるようにする．この結果，両者のレコードは，ソートアレー内で分離され往復ソート機能を実現できる．往復ソート機能を実現するためのレコード修飾法を表 3.1に示す．本方法では，往復ソート機能と同時に，昇順 / 降順のソート順を制御できる．このため，ソートアレー内の比較転送機能をソート順に依らず一定とでき，比較ユニットの制御回路を簡略にできる．また，レコードの先頭位置に付与したフラグと，前述したマージ制御のためのバンクタグの長さを同一とすることによって，マージ準備段階，中間マージ段階および出力マージ段階の全ての処理段階で，ソートアレーで扱うレコー

ド長を同一にできる.

3.4 ソータの実現と性能評価

大量レコードを k ウェイマージ処理の繰り返しでソートするアルゴリズムとハードウェアによる実現法を示した. その特徴は,

(1) ソートアレーによる並列比較とバンクタグを用いたデータ駆動型制御とによって, 数十ウェイのマージ回路を容易に実現できる. ウェイ数が大きいマージ処理をウェイ数に依存しない処理時間で実現できたことから, 少ないステージ数で大量のレコードを高速にソートできる. 現実的な応用では, 数十ウェイのマージ回路を用いて $3N$ あるいは $4N$ に比例する時間で処理できる.

(2) ソート可能な最大レコード数はワークメモリの容量で決まり, ソートアレーや制御回路の構成に依存しない. また, ソートアレーの規模を大きくしてマージウェイ数を拡大することで, マージ段数を削減してソート処理時間を短縮できる. この結果, ソート速度と最大レコード数の 2 つの独立した要求条件に応じた柔軟な構成のソータを実現できる.

(3) レコードの比較を行うソートアレーと, レコードを格納するワークメモリとを分離して実装できるため, ハードウェア化の範囲や規模を柔軟に設定できる. 例えば, 専用ハードウェア化したソートアレーを汎用マイクロプロセッサに付加して, 従来ソフトウェアで行っていたマージソートを高速化する構成や, マージ制御回路も含めて全体を専用ハードウェア化し, 小形・高速なソータを実現することもできる.

である. 次章に示す関係データベースプロセッサ RINDA では, 本アルゴリズムに基づくハードウェアソータを, マージ制御部を含めて専用 LSI で実現しているが, 以下では, マージ速度を決定づけるソートアレーだけを専用ハードウェアで実現した試作ソータの構成とその性能評価結果を示す.

表 3.2: 試作ハードウェアソータの概要

逐次多段 k ウェイマージソータ	
レコード長	$L : L < 64$ バイト
レコード数	N (ワークメモリ容量で制限)
マージウェイ数	k (レコード長に依存)
	$k = 80(L < 16$バイト$), 40(L < 32), 20(L < 64)$
処理速度	
マージ準備段階	3.0 MB/ 秒 (往復ソート機能により連続出力)
中間マージ段階	1.5 MB/ 秒
出力マージ段階	1.5 MB/ 秒
ハードウェア構成	
制御回路	汎用 μ プロセッサ (68020)
ソートアレー	大規模一括集積 LSI × 1 個
ワークメモリ	8 MB (1 M ビット ダイナミック RAM × 64 個)
ボードサイズ	B4 判× 1 枚

試作ソータの仕様を表 3.2に示す．ソートアレーチップは，図 2.3に示した構成であり，前章に示した 3 階層の階層化冗長構成法を適用して，40 個の比較ユニットを基本回路として一括集積してある．従って，本ソートアレーチップ 1 個で 16 バイトまでのレコード 80 個を 3M バイト / 秒のスループットで並列に比較できる．マージはレコードの入力操作と出力操作が対になることから，その速度は 1.5M バイト / 秒である．多段化したマルチウェイマージ制御は汎用マイクロプロセッサで行なう．8MB のワークメモリを搭載したシングルボードソータを試作し，500-K レコード (レコード長 15 バイト：1 バイトは制御用に使用) を，3 段マージによって約 13.4 秒でソートできることを確認した．さらに，本試作ソータを用いて **2.2.3**節に示したレコード長の拡張機能と，**3.3**節に示した往復ソート機能の実現性およびその有効性を検証した．さらに，エリア格納法を実装することでワークメモリの容量までソートできる大容量ソータを実現できた．

3.5 まとめ

本章では，前章に示したソートアレーを基本構成要素とし，ソートできる最大レコード数とソート処理速度に柔軟に対応できるマルチウェイマージソート法を提案した．本方法を適用したソータは，レコードを格納するワークメモリ，レコードを比較(並列比較)するソートアレー，逐次的なマージ操作を制御するマージ制御回路で構成される．データ駆動形マージ制御によるレコード列の選択と，並列比較を行なうソートアレーの採用により，ウェイ数を拡大してもマージ速度が低下しないマルチウェイマージ回路を構成し，逐次的なマージ操作の繰り返しでワークメモリに格納された任意個数のレコードを高速にソートできる．

逐次多段マージソートのためのワークメモリ管理法として，メモリ利用率を十分高くできるエリア格納法を提案した．従来のメモリ管理法では，マージウェイ数を大きくするとメモリ利用率が低下するのに対し，本方法ではマージウェイ数を大きくするに伴ってメモリ利用率を向上できることを明らかにした．

提案したマルチウェイマージソータは，レコードの比較回路(ソートアレー)とレコードの格納回路(ワークメモリ)とを分離できるため，それぞれを高密度に実装できる．また，マージ段数に依存するソート処理速度と，ワークメモリの容量で決まる最大ソート可能なレコード数を独立に設定できるため，利用者の要求条件に応じて最適な構成のソータを柔軟に実現できる．

前章で実現したソートアレーチップと汎用マイクロプロセッサを用いて，最大80ウェイ，マージ速度1.5MB/秒のハードウェアソータを試作し，提案法の検証と有効性を確認した．本試作シングルボードソータは，500-Kレコード(レコード長15バイト：1バイトは制御用に使用)を，3段マージによって約13.4秒でソートできた．

第 4 章

専用ハードウェア化を考慮した3フェーズジョイン法

4.1 まえがき

本章では，大規模関係データベースの問合せ応答時間を長大化させるジョインを並列処理によって高速化する手法 [63] を論じる．

ジョインは2つ以上の関連する表を結合して1つの表にする操作である．結合する2つの表の行数を M, N とすると，単純なアルゴリズムであるネストループ法 [5] は $O(M \times N)$ の演算量が必要であり，大規模データベースには適用できない．ネストループ法の比較回数を削減する方法として，あらかじめ結合キーの値で2つの表をソートしておくソートマージ結合法 [5] がある．この方法は，ソートされた2表をマージ操作によって結合するためソートを除く演算量が $O(M + N)$ であり，高速なソートが可能であれば大規模データベースに適用できる．結合操作の演算量を削減する別の方法としてハッシュジョイン法 [34] が提案されている．この方法は，2つの表を同一のハッシュ関数によりバケットに分割 (スプリット) した後に，対応するバケット間に限定して結合操作を行うので，演算量を $O(M+N)$ 近くまで削減できる．ただし，結合するバケットの一方を全て主記憶上に展開できない場合は，ネストループ法と類似な処理が必要となり演算量が増加する．このハッシュジョイン法については，マルチプロセッサシステムにおける関係データベースの並列処理 (第6章) で論じる．本章では，ソートマージ結合法を基本アルゴリズムとして前章に示したマルチウェイマージソータを用いることを前提に，ソートの前処理としてハッシュ化ビットアレイを用いたふるい落とし法 [50] を適用した3フェーズジョイン法 [63, 76] を提案する．本結合

法は，フィルタフェーズ，ソートフェーズ，マージ結合フェーズの3フェーズからなり，フィルタフェーズ，ソートフェーズのレコード入力，ソートフェーズのレコード出力とマージ結合フェーズの処理を行単位でパイプライン化して重畳する．ふるい落とし法は，一方の表のジョイン属性に関するハッシュ値をビットアレイに登録した後に，他方の表に関して各行でジョイン属性のハッシュ値を求めてビットアレイを参照し，対応するビットがセットされていない場合は結合の可能性はないとしてその行を除去する手法である．

データベースの表を構成する各行について，結合条件となる列の属性値を連結したデータを結合キー(キー)という．このキーに基づいて結合する表から，結合可能性のない行を除去するふるい落とし，残った行をキー順に並べ替えるソートは，処理対象となる行あるいはキーの個数が多く，これらの処理を専用ハードウェア化してパイプライン動作させ高速化を図る．最後のマージ結合は，ユーザが指定する種々の条件に基づいて行を連結する処理が含まれハードウェア化に適さないため，ホスト計算機上のソフトウェアで実行する．

提案する3フェーズジョイン法は，結合する2つの表の片方だけから結合の可能性のない行をふるい落とす片ハッシュ結合法と，2つの表をふるい落とす両ハッシュ結合法の2種類の実現が考えられる．いずれの方法でもフィルタフェーズにおいて十分に結合可能性のない行をふるい落としておくことが重要である．すなわち，ハッシュ化ビットアレイを用いたふるい落とし処理によって，以後のソートフェーズとマージ結合フェーズで対象となる行の数を削減する．このためには，ハッシュ化ビットアレイを設定・参照するハッシュ関数を，種々の属性および長さからなるキーに対して，ハッシュ値の衝突がないように設計しなければならない．衝突が起こると，結合相手のない行が誤ってふるい落とされず以後の処理の対象となるからである．ふるい落とし処理に適したハッシュ関数とは，種々のジョイン属性(キー)に対してハッシュ値が衝突しないことである．文献[48, 42]では，短い固定長キーを前提として，これまでに提案されているハッシュ関数の衝突の発生を比較している．キー値の集合が未知な場合は，除算法が比較的良い結果をもたらすことが報告されているが，ふるい落とし処理では，キー長が長い場合や可変長の場合があり，単純に除算法を適用することが

できない．長いキーを扱う方法に，折り畳み法 [48] と排他的論理和法 [6] がある．また，除算法より少ない演算量で同等の効果が得られる乗算法 [42] も知られている．これらの手法を組み合わせて，ふるい落とし処理で要求される条件を満足するハッシュ関数を設計しなければならない．

このような特性を持つハッシュ関数として，折り畳み法 [48] の 1 つである回転重ね合わせ法と乗算法 [42] を組み合わせた乗算重ね合わせ法を提案する．可変長のキーを固定長のキー切片に細分化して，各キー切片に対する乗算をあらかじめ作成した乗算表を用いて行ない，その結果をビットシフト (回転) と排他的論理和により重ね合わせる．この方法は，ハードウェア化することで，ハッシュ値の衝突率が十分に小さい良好なハッシュを高速に行ない，3 フェーズジョイン全体を行単位でパイプライン動作させる．

以下，**4.2**では従来法の問題点と解決の方針を示す．**4.3**では，提案する 3 フェーズジョイン法を詳細に述べ，片ハッシュ結合法と両ハッシュ結合法の 2 種類の実現法を示す．次に，**4.4**では，種々のキー属性や長さに対応できるハッシュ関数として提案した乗算重ね合わせ法とその評価結果を述べる．十分に大量な個数の固定長整数および可変長文字列を用いて評価した結果を示す．

4.2 従来法の問題点と解決の方針

ジョインの代表的な手法として，ネストループ法，ソートマージ法，ハッシュジョイン法が知られている．これらのアルゴリズムを演算回数から比較する．なお，本節における演算回数の議論では，それぞれ N, M 個 $(N \geq M)$ の行 (レコード) からなる 2 個の表をジョインするとし，結合キーはユニークなものと仮定している．

(1) ネストループ法 [5]: 第 1 表の各行の結合キーに対して第 2 表の全行の結合キーを比較することにより結合する方法である．演算回数は，2 つの表の行数の積に比例する値 $O(N \times M)$ である．

(2) ソートマージ法 [5]: 結合対象となる 2 個の表をそれぞれ結合キーでソートした後に，各表の先頭行から順に結合キー同士を比較することにより結合 (マージ結

合) する方法である．マージ結合の演算回数は $O(N+M)$ であり，各表のソートに要する演算回数は各々 $O(N\,log_2\,N), O(M\,log_2\,M)$ である．従って，N, M が十分大きい，すなわち大規模データベースでは，マージ結合の演算回数はソートの演算回数と比較して十分に小さいといえる．

(3) ハッシュジョイン法 [34]: 結合対象となる 2 個の表を結合キーを用いて複数のバケットに分割し，対応するバケット間で結合を行う方法である．バケットへの分割は，結合キーに適当なハッシュ関数を適用して同じハッシュ値を持つ行をバケットに集める．対応するバケット間の結合は，上記のネストループ法あるいはソートマージ法を用いる．従って，バケット数が十分に多くバケットへの分割が均等に行なえれば，ほぼ線形時間 $O(N+M)$ でジョインできる．

上記のジョイン法で (1) のネストループ法は，他のアルゴリズムと比較して演算回数が多く，大規模データベースを対象とすることはできない．しかし，制御は簡単であることから，第 2 表 (行数:M) がランダムアクセス可能な主記憶装置に格納できる程度に小さい場合に多用される手法である．(2) のソートマージ法は，ソートを除けば線形時間でジョインが行なえるため，大規模データベースに適用することができる．これまで，ソートを高速化する並列処理アルゴリズムが種々提案 [1] されており，本章においても，第 3章に示した高速かつ柔軟な構成を取り得るマルチウェイマージソータを適用する．

(3) のハッシュジョイン法は，表をハッシュ関数で複数のバケットに分割し，対応するバケット間で結合を行うことから並列処理に向いたジョイン法といえる．並列処理を高いスケーラビリティ (プロセッサ数に比例した性能) で実現するには，バケットへの分割を均等化してプロセッサ負荷をバランスさせることが重要である．本論文では，このような負荷バランス問題の一例として並列ハッシュジョイン法を第 6章で論じる．

ジョインを高速化する手法には，上記 3 種類の基本アルゴリズムのいずれにも適用可能なふるい落とし法 [50] が知られている．第 1 表の各行を結合キーでハッシュし，ハッシュ表の対応するエントリに行を格納する．その後，第 2 表を同じハッシュ関数

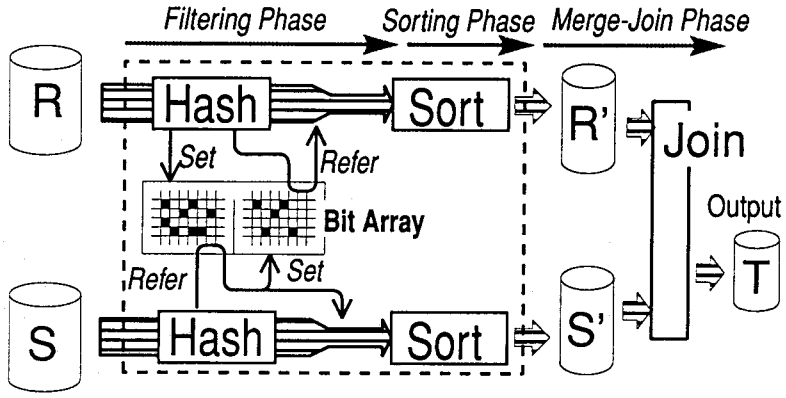


図 4.1: 3フェーズジョイン法の基本動作

でハッシュし，ハッシュ値に対応するハッシュ表のエントリに格納されている行と比較し，対応する行が存在すればそれらを結合する．ハッシュ表をビットアレイで構成しておき，第1表のハッシュ値でビットアレイを設定し，第2表のハッシュ値でビットアレイを参照し，対応するビットが設定されている場合に結合を行なうことで実現できる．このようなビットアレイをハッシュ化ビットアレイと呼ぶ．この方法を用いることで，結合の可能性がない行を線形時間でふるい落とすことができる．

4.3 3フェーズジョイン法

4.3.1 基本アルゴリズムとハードウェア化の方針

前節で示した従来法の問題点を考慮して，ソートマージ法を基本にふるい落とし法を効果的に融合させた3フェーズジョイン法を提案する．本方法の基本的な動作を図4.1に示す．結合する2個の表から，結合キーに基づいて結合可能性のない行を除去するフィルタフェーズ，残った行をキーの順序に並べ替えるソートフェーズ，ソートされた表の行を順次マージして連結するマージ結合フェーズの3段階からなる．フィルタフェーズとソートフェーズ，ソートフェーズとマージ結合フェーズの間は，行を単位としたパイプライン処理が可能であり，専用ハードウェア化することで極めて高速なジョインを実現できる．

次に，専用ハードウェア化の方針を示す．最初のフィルタフェーズにおいて，不要

な行を除去する手段としてハッシュ化ビットアレイを用いたふるい落とし手法 [50] を実現する．まず，第1表の各行から結合のためのキーを作成し，そのキーをハッシュ関数を用いてハッシュし，ビットアレイの対応する位置に設定する．次に，第2表の各行から作成したキーを，第1表で用いたと同一のハッシュ関数でハッシュし，ハッシュ値に対応するビットアレイの位置が設定してなければ，その行は結合の可能性がないとする．第1表に関する操作をビットアレイの設定，第2表に関する操作をビットアレイの参照と称し，ハッシュ値の一致，不一致によって第2表から結合可能性のない行をふるい落とす．従って，フィルタフェーズで用いるハッシュ関数は，種々の属性および長さからなるキーを処理できなければならない．また，ハッシュ値に衝突 (Collision) が発生すると，結合可能性のない行が残る．

以上示したフィルタフェーズは，結合する2つの表の全行に対してハッシュ値を算出し，ビットアレイを設定，参照するフェーズであり，以後のソートフェーズ，マージ結合フェーズと比較して最も多くの行を処理しなければならない．また，結合に用いられるキーの個数や長さ，キーの属性や分布の片寄りはあらかじめ知ることができないから，より高度な，すなわち計算量の多いハッシュ関数を使用する必要がある．以上の理由により，フィルタフェーズは専用ハードウェアで実現することとした．

ソートフェーズは，多くのソータの研究例を挙げるまでもなく計算機処理量が大きく，専用ハードウェア化による高速化効果が大きい処理である．データベース処理では，ソートするキーの個数や長さが大きく変動することから，これらの値に柔軟に対処でき，かつ，大容量のソータをコンパクトに実現できるマルチウェイマージソータを実現する．

マージ結合フェーズの入力は，既に，フィルタフェーズで結合可能性のない行が大部分除去され，かつ，ソートフェーズで結合対象となる2つの一時表がソートされている．このため，マージ結合の計算機処理量が十分に小さくなっていると期待される．一方，マージに基づく行の連結操作は，ユーザによって指定された出力行の構成に大きく依存する．以上の理由から，マージ結合フェーズの処理は，ホスト計算機上のソフトウェアで実行する．

以上のように，3フェーズジョイン法は従来から提案されているふるい落とし，ソートマージといった手法を効果的に組み合わせた手法であり，専用ハードウェア化を考慮して抽出した3つの処理要素を，データベース処理という処理形態を考慮してホスト計算機と専用ハードウェアの役割分担を明確にしたことに特徴がある．専用ハードウェアで実現した処理は，扱う行数が多く行を単位にパイプライン的に重畳させることができる処理であり，細粒度並列処理を専用ハードウェアで行なっているといえる．

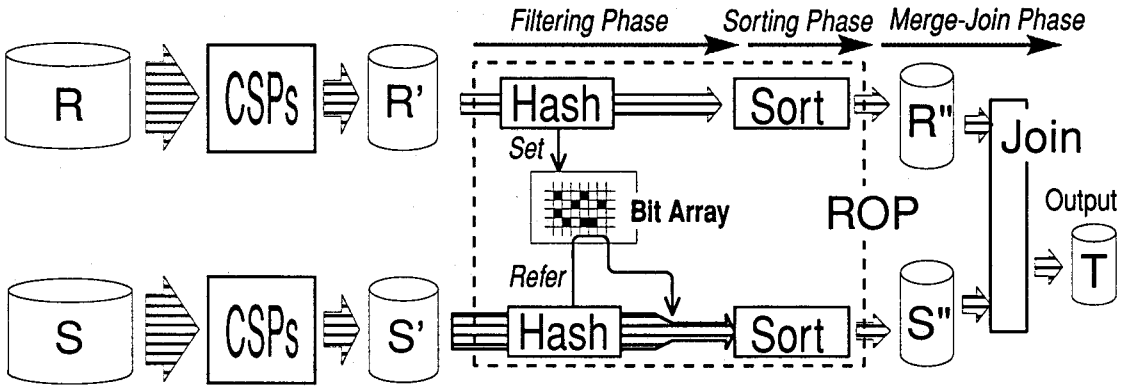
4.3.2 3フェーズジョイン法の実現

3フェーズジョイン法は，ふるい落としを行う表が結合対象となる2つの表の片方か両方かによって，片ハッシュ結合法と両ハッシュ結合法の2種類の方法が実現できる．処理の概要を，図4.2に示す． 図中のCSPとROPは，データベースプロセッサRINDAを構成する内容検索プロセッサと関係演算高速化プロセッサであり，詳細は第5章に示す．図中のR, Sはそれぞれ結合対象となる表であり，内容検索プロセッサCSPでサーチ(選択・射影)した結果の一時表がR', S'である．これらの一時表にフィルタフェーズおよびソートフェーズの処理を施した結果が一時表R", S"であり，最後のマージ結合フェーズによって結合した結果が一時表Tである．

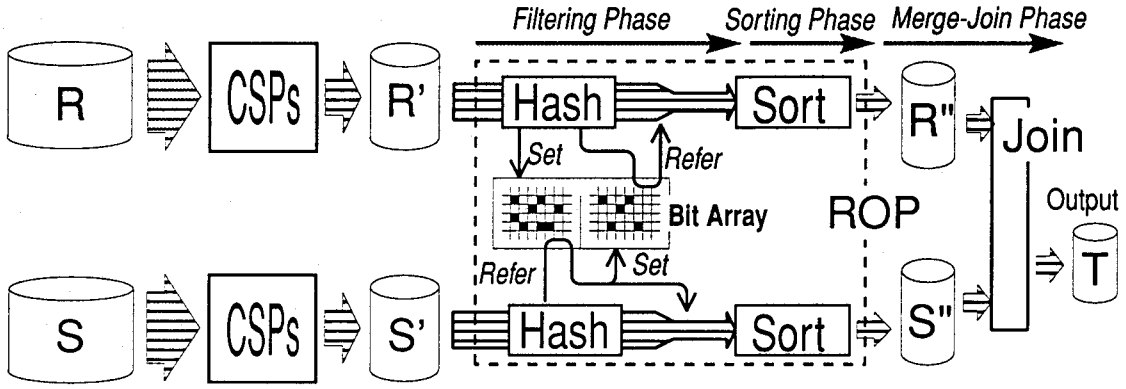
図4.2に示す片ハッシュ結合法と両ハッシュ結合法のいずれの方法でも，ふるい落とし処理で残った行数がハードウェア化したソータの容量以下であれば，ソータのオーバフローは起きない．このため，より大規模な表であっても結合処理の対象とすることができる．また，ふるい落としで出力される一時表が小さくなれば，一時表の転送に要する時間とマージ結合に要する時間を短縮できる．以下では，片ハッシュ結合法と両ハッシュ結合法について具体的に述べる．

片ハッシュ結合法

(1) 設定ソート: ビットアレイの全ビットを初期化した後に，第1表を入力しながらビットアレイの設定とソータへの入力を行なう．行の入力，ハッシュ値の計算とビットアレイの設定，ソータへの入力は行単位でパイプライン処理する．



(a) 片ハッシュ結合法



(b) 両ハッシュ結合法

図 4.2: 3 フェーズジョイン法の実現

ソート結果はソート済み一時表として出力する.

(2) **参照ソート:** 第2表を入力しながら各行について既に設定されているビットアレイを参照し，ビットアレイの対応する位置が上記設定ソートで設定されていれば結合可能性のある行としてソータに入力する．行の入力，ハッシュ値の計算とビットアレイの参照，ソータへの入力は行単位でパイプライン処理する.

ソート結果はソート済み一時表として出力する.

(3) **マージ結合:** 得られた2つのソート済み一時表をホスト計算機でマージ結合する．上述の参照ソートにおけるソート結果の出力と，ここでのマージ結合操作はパイプライン化する.

本方法では，第2表だけがふるい落とされている.

両ハッシュ結合法

(1) **設定:** ビットアレイの全ビットを初期化した後に，第1表を入力しながらビットアレイを設定する．行の入力，ハッシュ値の計算とビットアレイの設定は行単位でパイプライン処理する.

ここでの処理は，片ハッシュ結合法と異なりソートは行なわないので，ソータの容量を越える大規模な表であっても入力することができる.

(2) **参照ソート設定:** 第2表を入力しながら各行について既に設定されているビットアレイを参照し，ビットアレイの対応する位置が上記設定操作で設定されていれば結合可能性のある行としてソータに入力する．同時に設定段階で使用したとは異なる第2のビットアレイを，結合可能性のある行として残された，すなわちソータに入力する行で設定する.

行の入力，ハッシュ値の計算とビットアレイの参照は行単位でパイプライン処理し，さらに，参照の結果から結合可能性のある行と判定された場合は，第2のビットアレイの設定とソータへの入力までをパイプライン処理する.

ソート結果はソート済み一時表として出力する.

(3) 参照ソート: 再度，第1表を入力しながら各行について上記第2のビットアレイを参照し，結合可能性のある行を抽出してソータに入力する．片ハッシュ結合法と同様に行単位のパイプライン処理を行なう．

ソート結果はソート済み一時表として出力する．ソート結果の出力と次のマージ結合も行単位でパイプライン処理する．

(4) マージ結合: 得られた2つのソート済み一時表をホスト計算機でマージ結合する．本方法では，第1表の入力を2回行ない，第1表と第2表をともにふるい落としている．この場合，いずれの表もふるい落としの後にソートを行なっているから，ふるい落とし後の行数がソータの容量以下であればジョインすることができる．

4.4 ハッシュ関数の設計

4.4.1 要求条件

ふるい落としの効果を左右する主要な要素としてハッシュ回路で用いるハッシュ関数がある．ハッシュ関数は，互いに異なる2個以上のキーのハッシュ値が衝突 (Collision) する場合がある．関係データベース処理では，キーを生成する列の属性が整数であったり文字列であったり，時には異なる属性を持つ複数の列からキーを生成する場合もある．また，キー値の分布に片寄りが生じる場合もあり，単純なハッシュ関数では衝突率が高くなると考えられる．ハッシュ値が衝突すると，結合可能性のない行を十分に除去できなくなるため，より高度なハッシュ関数が要求される．

ハッシュ関数は，キーをハッシュ表のアドレス空間に一様 (ランダム) に割り当てるものが望ましい．文献 [48, 42] は，短い固定長キーを前提として，既に提案されているハッシュ関数の衝突の発生を比較している．代表的なハッシュ関数は，除算法，乗算法，折り畳み法，平方採中法，文字解析，ビット解析，基数変換法などであり，キー値の集合が未知な場合は，除算法が比較的良い結果をもたらすとしている．ふるい落とし処理では，キー長が長い場合や可変長の場合があり，そのまま除算法を適用することはできない．以下にふるい落とし処理で用いるハッシュ関数の特徴を示す．

- ハッシュ表は，ハッシュ値に対応するキーの有無を表示する1ビットのセルの集まりである．このため，以下に示すロードファクタを小さくできる．

$$\text{ロードファクタ} = \frac{\text{異なりキー値の総数}}{\text{ハッシュ表のセル数}}$$

- 衝突率の低減は重要ではあるが，衝突が起きた場合でもあふれ(オーバフロー)等の処理は不要である．
- 種々のデータ属性に対して，同一のハッシュ関数が適用できる必要がある．
- 比較的長い可変長のキーを扱える必要がある．

4.4.2 乗算重ね合わせ法

比較的長い可変長のキーを扱う方法に，キーを固定長の切片に分割して，それらを排他的論理和を用いて重ね合わせる排他的論理和法 [6] がある．しかし，文字列，特に日本語文字列の文字コードは，あるビットに1が出現する確率に偏りがあるため，単純に1バイトあるいは2バイトの切片を用いて重ね合わせると，結果のハッシュ値に偏りが生じる．

ハッシュ値の偏りを軽減する手法として，適当なビット数をずらして重ね合わせたり，ビット順を入れ換えて折り畳む方法が知られている．ここでは，ハードウェア化の容易性の観点から乗算法とビット回転を施した切片を排他的論理和法によって重ね合わせる乗算重ね合わせ法を実現する．乗算は文字コードの偏りを分散するため，回転重ね合わせは可変長キーを同一のハードウェアで扱うためである．

乗算重ねあわせ法を実現したふるい落とし回路のハードウェア構成を図4.3に示す．乗算は，固定長の切片にあらかじめ定められた定数を掛け合せる演算で精度の要求も小さい．そこで，乗算結果をメモリに展開した乗算表を作成しておき，切片の値をアドレスとしてアクセスすることで実現する．一方，一定ビットの回転はハードウェア上の配線をずらして結線するシフト回路で実現でき，排他的論理和も桁上り処理が必要ないことから，簡単な回路で高速に処理できる．ハッシュ回路内は，キーを分割した切片を単位にパイプライン処理され，キーの入力速度に完全に追従して動作する．

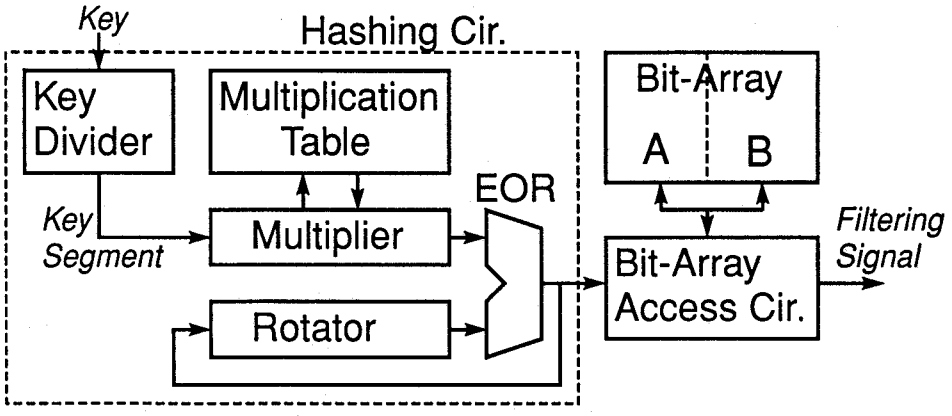


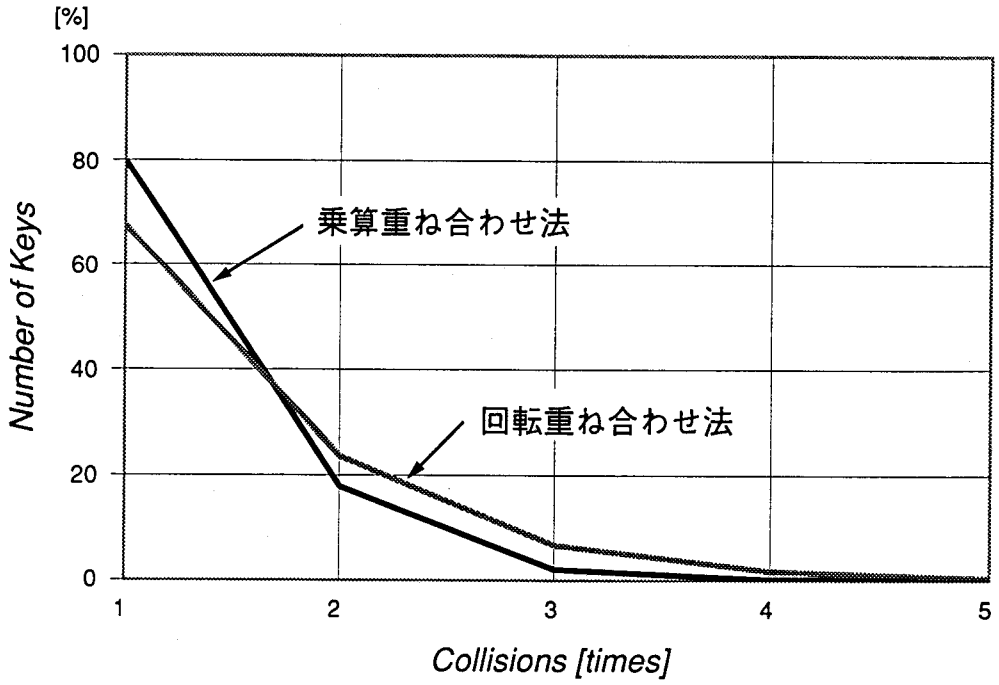
図 4.3: ふるい落とし回路の構成

ハッシュ回路の出力 (ハッシュ値) でビットアレイをアクセスして設定あるいは参照の各操作を行う．ビットアレイは，両ハッシュ結合法のために A と B の 2 面ある．ハッシュ回路内のパイプライン処理によって，キーの入力が完了した時点で遅れなくハッシュ値が作成され，ビットアレイをアクセスしてふるい落とし処理を行なえる．

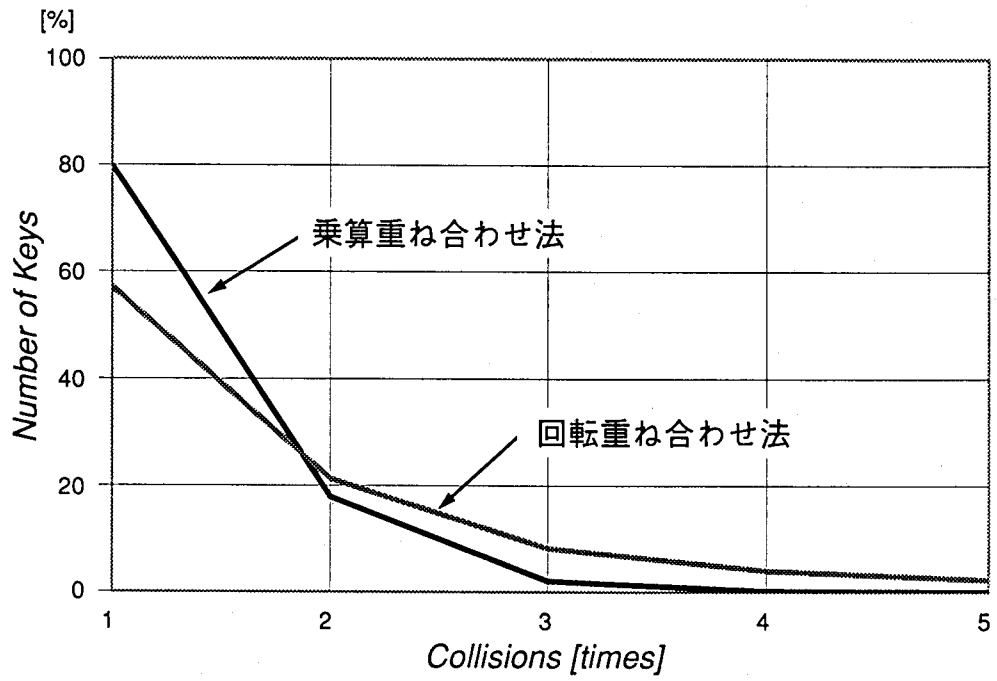
4.4.3 衝突率の評価

乗算重ね合わせ法の効果を定量的に評価した結果を図 4.4に示す．図の横軸は，同一のハッシュ値に衝突するキーの個数を示しており，従来法である回転重ね合わせ法 [48] は，回転数を 0 から 7 まで変えて最も衝突の発生が少なかった回転数 6 の場合を示している．図 4.4(a) は長さ 16 バイト固定長のランダム数字列を，図 4.4(b) は英語辞書の見出し語を用いた場合であり可変長の英単語である．評価したワード数は (a), (b) とも見出し語数に等しい約 25 万語である．ビットアレイの大きさは 100 万個である．

評価結果から，いずれの場合も乗算重ね合わせ法が優れているといえる．特に，キーの分布に偏りがある英単語の例では，回転重ね合わせ法の衝突無しの比率が 60% 未満に対して，乗算重ね合わせ法は 80% の単語が衝突無しである．また，キーの衝突回数で比較すると，乗算重ね合わせ法は，ランダム数字列と英単語でほぼ同一の結果が得られているのに対して，回転重ね合わせ法は，文字コードの分布に依存している



(a) ランダム数字列 (16バイト長)



(b) 英単語 (可変長)

図 4.4: 乗算重ね合わせ法におけるハッシング効果

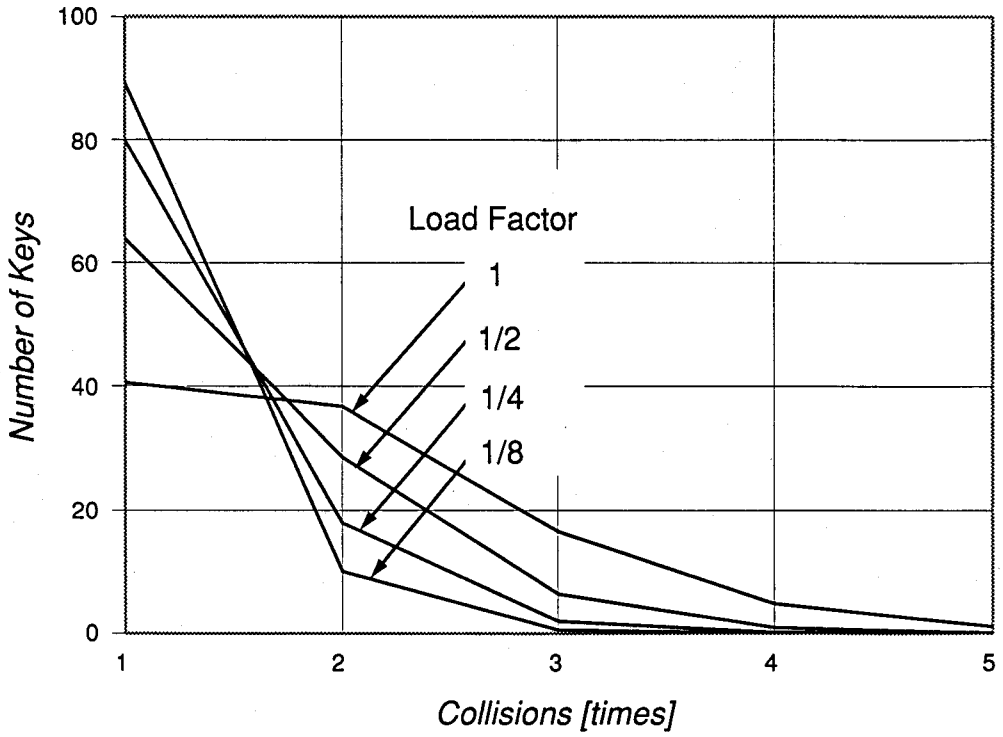


図 4.5: ロードファクタに対するハッシング効果の違い

ことがわかる．英単語のハッシュ結果では，同一ハッシュ値に衝突するキーの個数が 5 を越える割合が 10% 程度と高い．乗算重ね合わせ法では，いずれの評価でも 3 個以上のキーのハッシュ値はほとんど衝突していない．

次に，ビットアレイの大きさ (セル数) が衝突の度合いに与える影響を示す．ビットアレイの大きさを変えて所定の個数のキーをハッシュすることによって，前述のロードファクタを変化させ評価した．ハッシュ関数は上述の乗算重ね合わせ法を用い，ロードファクタを 1 から 1/8 まで変えた時の衝突の度合いを図 4.5に示す．評価に用いたキーは，図 4.4(b) と同一である．

ロードファクタが 1 の時，すなわち，キー値の個数とセルの個数が等しい時は，約 4 割のキーは衝突なし，残りの約 4 割のキーが 1 回の衝突，2 割のキーが 2 回以上の衝突を起こしている．ロードファクタが 1/4 以下になると，衝突なしの割合が 8 割を超え，2 回以上衝突する確率はほぼゼロとなる．

以上の結果より，ハッシュ化ビットアレイを用いたふるい落とし処理では，ロードファクタを小さくする，すなわち，ビットアレイのサイズを大きくすることによって，キーの衝突を回避することができる．すなわち，ハッシュするキーの個数が増加するに伴って，ビットアレイのサイズを大きくすると衝突率を一定に保つことができる．

4.5 まとめ

関係データベース処理の中で，特に，高速化が要求されているジョインの高速化手法について示した．第3章に示したマルチウェイマージソータを用い，さらに，ソートの前処理としてハッシュ化ビットアレイを用いたふるい落とし法を適用した3フェーズジョイン法を提案した．

本方法は，結合する表から結合可能性のない行をふるい落とすフィルタフェーズ，残った行を並べ替えるソートフェーズ，ソートされた行をマージしながら連結するマージ結合フェーズからなる．従来のソートマージ結合法の前処理としてふるい落とし処理を効果的に組み合わせた手法であり，ふるい落としの対象となる表が，1個(片方)か2個(両方)かによって片ハッシュ結合法と両ハッシュ結合法の2種類が実現できる．

ハードウェア化の容易性とシステム構成の柔軟性から，フィルタフェーズとソートフェーズを専用ハードウェア化する．フィルタフェーズ，ソートフェーズのレコード入力，ソートフェーズのレコード出力とマージ結合フェーズの処理を行単位でパイプライン化して重畳することができ，専用ハードウェア化することで極めて高速なジョインを実現できる．専用ハードウェア化することで，扱う行数が多い処理を行単位でパイプライン処理したといえる．

ハッシュ化ビットアレイを用いるフィルタフェーズに適用するハッシュ関数として，キーの長さ，属性が変わっても衝突が起こりにくい乗算重ね合わせ法を提案した．乗算は，固定長の切片に定数を掛け合せる演算で精度の要求も小さいことから，乗算結果をメモリに展開した乗算表を用いて行なう．乗算結果の重ね合わせは，シフト回路と排他的論理和回路で実現する．このように乗算重ね合わせ法を用いたハッシュ回路

は簡単な回路で実現でき，キーを分割した切片を単位とするパイプライン処理によって，キーの入力速度に完全に追従した動作を実現している．

第 5 章

データベースプロセッサの実現と評価

5.1 まえがき

これまで述べてきた細粒度並列によるソートやジョインの高速化手法を有機的に統合したデータベースプロセッサ RINDA[26, 27, 67] のアーキテクチャと性能評価結果を論じる．さらに，本章では，第 1 章で提起したサーチの高速化手法 [21] についても議論する．

近年，関係データベースを用いたシステムでは，ギガバイトを超えるデータベースの構築や数時間もかかる統計処理が実際に行われるようになり，処理の高速化・高機能化，データベースの大規模化の要求がますます増大してきている．特に，関係データベースに対するインデックスの利用が困難な非定型の検索処理は，処理自体は単純な比較に基づく条件判定やソートが主体であるが，磁気ディスク装置等の 2 次記憶装置に格納された大量のデータを処理対象とするため，比較的少量のデータに複雑な計算を行うことを前提とした汎用計算機では効率が悪い．このため，データベース処理向きのアーキテクチャを持つデータベースマシン [37, 40] が研究され，実用に供せられてきている．これは，関係データベースのデータ操作水準が高く，ソフトウェアのみの実現では高速化に限度があり，近年の半導体を始めとするハードウェアテクノロジーの進歩を背景として，専用ハードウェアの支援による高速化が有効となってきたことによる．

これまでに実現されているデータベースマシンは，主に，ディスクとプロセッサ間の IO ネックの解消と，ソートやジョインなどの繰り返し演算による CPU ネックの

解消をめざしている．例えば，CAFS[3] では，ディスクのデータ読み出しと同時にタプルの選択と射影を行うプロセッサをディスクのコントローラに付加して，IO ネックの解消を狙っている．専用のハードウェアソータを備えたGREO[2] や，汎用計算機のベクトルプロセッサをデータベース処理用に拡張したIDP[43] は，CPU ネックを解消しようとしている．また，入出力処理を行う複数の汎用マイクロプロセッサとデータベース処理向きに設計された専用プロセッサとで構成されるServer/8000[8] は，IO ネックとCPU ネックの両方に対処している．

著者らは現状のソフトウェアによるデータベース処理において弱点とされている，インデックスの利用が困難な非定型の検索処理を専用ハードウェアを用いて高速化することにより，データベース処理の総合的な性能向上を目指したデータベースプロセッサRINDA(Relational Database Processor)[26, 27, 67] を開発した．RINDA は，関係データベース処理において基本的な演算であり，かつ，従来のソフトウェアでは負荷の重かった，サーチ，ソート，ジョインを高速化している．RINDA のハードウェアは，サーチを高速化する内容検索プロセッサ (CSP：Content Search Processor)[21] と，ソートおよびジョインを高速化する関係演算高速化プロセッサ (ROP：Relational Operation Accelerationg Processor)[76] からなり，それぞれIO ネックとCPU ネックを解消することを狙いとしている．負荷の重い関係演算処理を直接ハードウェアで実行することによって，従来の汎用計算機上のソフトウェアシステムと比較して，最高で100 倍以上の高速化を実現している．

RINDA(CSP とROP) は，汎用計算機DIPS シリーズ [56, 85] の周辺装置として位置付けられ，汎用計算機上のデータベース管理システム (DBMS) の負荷を軽減するハードウェアである．これらのハードウェアは，DIPS 上のDBMS によって制御され，RINDA とRINDA を制御するDBMS(ソフトウェア) をRINDA システムと呼ぶ．

関係演算高速化プロセッサ (ROP) は，ホスト計算機とチャネルインタフェースで接続され，第3章と第4章に述べたソートとジョインを高速化するためのハードウェアを実現している．さらに，ROP では，従来のデータベースマシンであまり考慮されていなかった専用ハードウェアへのデータの入出力に伴う処理対象属性の切り出し (キー抽出) 処理，および，使用頻度の高い統計処理の一つであるグループ化計数処理

も専用ハードウェア化している．

一方の内容検索プロセッサ (CSP) は，非定型問合せのサーチを実行する専用ハードウェアで，インテリジェントなディスク制御装置としてホスト計算機とディスク装置の間にチャネルインタフェースで接続されている．RINDA システムではサーチを基本的には CSP が直接実行するアーキテクチャであるが，ハードウェア化を考慮して一部処理を DBMS が分担している．このため，DBMS は与えられた検索条件の論理式を乗法標準形に変換し，CSP が実行可能な部分のコマンド列を CSP に転送する．これにより CSP がサーチして絞り込んだ結果を DBMS が処理する．DBMS は上記論理式の変換過程で，ブール演算子 NOT を使用しない乗法標準形に変換することで，SQL 仕様に基づく 3 値論理の探索条件を等価な 2 値論理として CSP で処理できる．

CSP は汎用ディスク装置に対して高速サーチを実行するため，シリンダ単位のマルチトラックリードでページの連続読み出しを行い，ページ単位でデータ読み出しと条件判定をパイプライン処理する．さらに，RINDA システムでは一つの表を複数のディスク装置に分割格納し，複数 (p 台) の CSP で並列サーチし，サーチ時間を $1/p$ に短縮している．

以下，**5.2**では RINDA アーキテクチャと非定型問合せの処理方法を示す．**5.3**と**5.4**では，RINDA の基本構成要素である関係演算高速化プロセッサ ROP と内容検索プロセッサ CSP の実現法を述べる．**5.5**では，RINDA ハードウェアを付加することによる関係データベース処理の高速化効果を評価し，最後に **5.6**でまとめる．

5.2 基本概念

5.2.1 設計方針

関係データベース処理は，ソフトウェアによる高速化手法であるインデックスを利用することで問合せの応答時間を大幅に短縮できる場合がある．インデックスを付与していない列による検索，ソートやジョイン，集約演算を伴う問合せは，インデックスを有効に利用することができないことから，従来のソフトウェアのみによる処理では多大の時間を要していた [25]．主な原因は 2 つ考えられる．1 つは，磁気ディス

ク装置に格納された表を主記憶に読み出し，各行に対して逐次的に条件判定を行わなければならず，そのためのサーチに時間がかかることである．他の1つは，非定型処理では単一表のグループ化，または複数の表を対象とする結合処理，副問い合わせを伴うことが多く，それに伴うソートやジョインに時間がかかることである．

以上示したサーチ，ソート，ジョインに要する処理時間は，対象とする表のサイズ(行数)に比例して増大する．データベースマシンRINDAの開発目標は，上記のインデックスを使用しない問合せ処理にかかるホスト計算機の負荷を軽減し，問合せ応答時間を短縮することである．磁気ディスクの制御装置に検索用のハードウェアを設け，検索結果だけをホスト計算機に返却することでサーチの負荷を軽減する．一方，大容量メモリを持つハードウェアソータをホスト計算機の周辺装置として備え，ソートとジョインにかかるホスト計算機の負荷を軽減する．

RINDAの設計で考慮した主要な要求条件は，以下の2項目である．

- アプリケーションプログラムあるいはプログラマにRINDAの存在を意識させないこと．アプリケーションプログラムを変更することなく，RINDAによる性能向上を享受できる．
- データベース規模が大きくなっても応答時間が悪化しないスケーラブルな構成であること．データベース規模が小さい時は，わずかなハードウェアの付加で十分であり，規模が大きくなるに伴ってより強力なハードウェアを付加できる．

第1の要求条件である利用者プログラムの変更を回避するために，RINDAシステムではSQLインタフェース[28]を実現する．RINDAハードウェアで実現する機能は，SQLで必要な機能のサブセットとし，ホスト計算機上のソフトウェアによって，RINDAのサブセット機能を使ってフルセットのSQL機能を実現する．RINDAハードウェアを使用するための問合せ最適化や，RINDAに特化したアルゴリズムも実現する．第2の要求条件であるスケーラブルな構成とするために，RINDAを内容検索プロセッサCSPと関係演算高速化プロセッサROPの2つの独立したハードウェアで構成する．個々の利用者やアプリケーションプログラムが要求する性能を満たすように，必要な個数のCSPとROPをシステムに組み込むことができる構成とする．

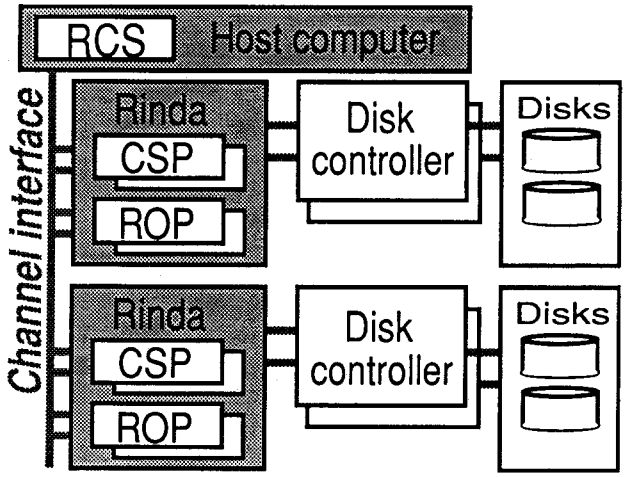


図 5.1: RINDA システムの構成

5.2.2 ハードウェア構成

ホスト計算機に RINDA を接続したシステム構成を図 5.1に示す．システムは NTT によって開発された DIPS シリーズ汎用計算機，標準的なディスク制御装置，ディスクユニットと RINDA ハードウェアで構成される．RINDA は，複数の内容検索プロセッサ CSP と関係演算高速化プロセッサ ROP からなり，それぞれ独立の入出力インタフェースでホスト計算機と接続されている．接続する CSP と ROP の個数は，データベースの規模と要求される応答性能によって決める．例えば，極めて大規模なデータベースを対象とした検索だけでソートやジョインが必要ない問合せを高速に処理したい場合は，複数の CSP だけで RINDA を構成し CSP 間の並列処理で応答時間を短縮する．

CSP と ROP は，ホスト計算機上のデータベース管理システム (DBMS) の RINDA 制御部 RCS(Rinda Control Subsystem) で作成されたチャネルコマンドとそのパラメータで制御する．CSP はディスク装置に格納されたデータベースの表を直接検索し，問合せで指定された条件を満たす行の必要な列だけを，一時表の形でホスト計算機に返却する．このため，ホスト計算機は，ディスク読み出しと条件判定の繰り返し処理

表 5.1: CSP の機能概要

Function	Description
Predicates	Specified in a WHERE clause
Comparison	< column > < comp-op. > < value >
In	< column >[NOT] IN < value list >
Like	< column >[NOT] LIKE < pattern >
Null	< column > IS [NOT] NULL
Boolian expression	Any combination of predicates
Column selection	SELECT < column list >
Set function	COUNT(*)

表 5.2: ROP の機能概要

Function	Description
Sorting	ORDER BY < column [ASC|DESC] list > (also used for joins, subqueries, and GROUP BY clause)
Filtering	Removal of unnecessary rows (used for joins and nested queries)
Duplicate removal	DISTINCT < column list >
Set function	COUNT(*) with a GROUP BY clause

から開放され大幅に負荷が軽減される．CSP が実現している主な機能を表 5.1に示す．CSP では，単一の表に関するほとんどの機能を実現しており，ディスクからの読み出し速度に追従して処理できる．

ROP は，ホスト計算機から送られた表の行をソートし結果を返却する．その際，ジョインやネストした問合せの高速化に寄与する，不要な行のふるい落し処理を行なう．ROP が実現している主な機能を表 5.2に示す．ROP では，チャネルの転送速度に追従してこれらの機能をパイプライン処理する．

5.2.3 ソフトウェア構成

ホスト計算機上の DBMS ソフトウェア構成を図 5.2に示す．既存のソフトウェアだけで処理する DBMS に，RINDA 制御部 (RCS) を組み込む構成とした．COBOL

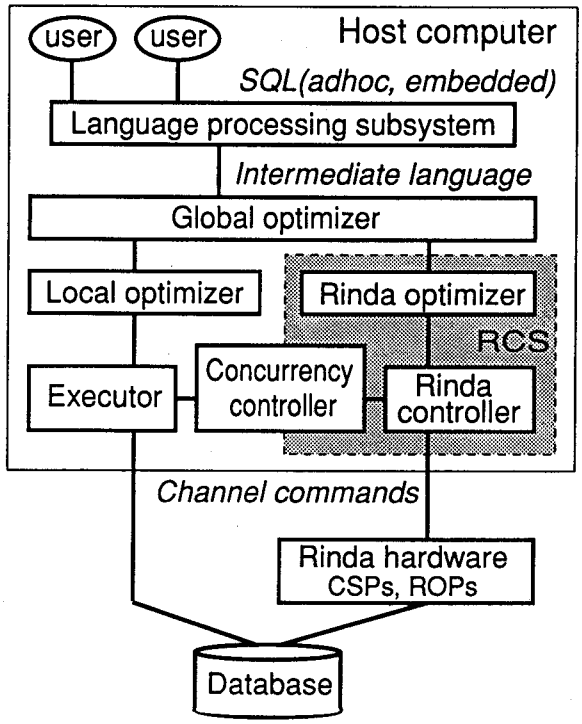


図 5.2: DBMS のソフトウェア構成

等で書かれたアプリケーションの埋め込み型 SQL 文や，利用者が発行するアドホック SQL 文は，インデックスの使用・未使用に関係なく同一の書式で記述できる．これらの SQL 文は，最初に，DBMS の言語解析部 (Language processing subsystem) で内容を解析される．次に，大域最適化部 (Global optimizer) で，付加ハードウェアを使用する RINDA 方式と，インデックスを使用するソフトウェア方式のいずれで実行するかを判定する．これをアクセスパスの決定という．インデックスを使用しない問合せ (以下では非定型問合せと呼ぶ) であれば，RINDA 方式を選択し RCS を呼ぶ．RCS は，RINDA 最適化部 (Rinda optimizer) と RINDA 駆動部 (Rinda controller) からなり，RINDA 最適化部が最適化した問合せ実行手順に従って RINDA 駆動部が RINDA ハードウェアにオーダ (制御情報) を発行する．ソフトウェア方式では，局所最適化部 (Local optimizer) で問合せ実行を最適化する．したがって，RINDA システムでは，アクセスパスの決定と，アクセスパス内での最適化による 2 フェーズの最適化を行う．アクセスパスの決定で考慮する要因を以下に示す．

- 一般にトランザクションは複数のSQL文を含み，SQL文単位でRINDA方式とソフトウェア方式のいずれかの実行方式を選択できる．このため，アクセスパスの決定は，個々のSQL文毎にRINDA方式とソフトウェア方式のいずれかを逐次的に選択する．RINDA方式で処理する場合は，ソフトウェア方式と共通に設けた並行処理制御部 (Concurrency controller) をRCSが制御して，複数利用者環境におけるデータベースの一貫性を維持する．

- アクセスパスは，CPU演算量がディスク入出力回数に比例すると仮定し，ディスク入出力回数だけを評価して決定する．この方法では，最適パスを正確に選択できない場合が生じるが，パスの選択は短時間で行える利点がある．データベース処理のコストはデータ分布等に大きく依存し，厳密な実行コストの評価は困難であることを考慮すると，正確よりも迅速なパス選択が重要であると考えた．アクセスパスの選択誤りによる応答時間の大幅な悪化を避けるために，確実な判定が下せない時はRINDA方式を選択することにした．その理由は，RINDA方式がディスク入出力処理と重畳して検索処理等を行うために，ディスク入出力回数でほぼ正確な予測が行え，しかも，ソフトウェア方式ほど処理時間に幅が生じないためである．

RINDAを用いた典型的な非定型処理の実行例を図5.3に示す．8月1日にサンフランシスコSFOに向けて出発した乗客数PSNを求める問合せである．それぞれの表にはインデックスが付与されておらず，大域最適化部はRINDA方式を選択したとする．以下，図5.3を参照しながら処理の流れを示す．

手順 (1)–(3)： RCSは，“Schedule”表を検索するコマンドをCSPに送る．検索条件はDST='SFO'である．検索結果は，RCSの管理するバッファに返却される．

手順 (4)： ROPに対して片ハッシュ結合法 (4.3参照) を実行するための 設定ソートコマンドを送る．引き続いて“Schedule”表の検索結果を送る．ROPは受け取った検索結果をソートして返却する．(図を簡略にするために結果の返却は表示していない．)

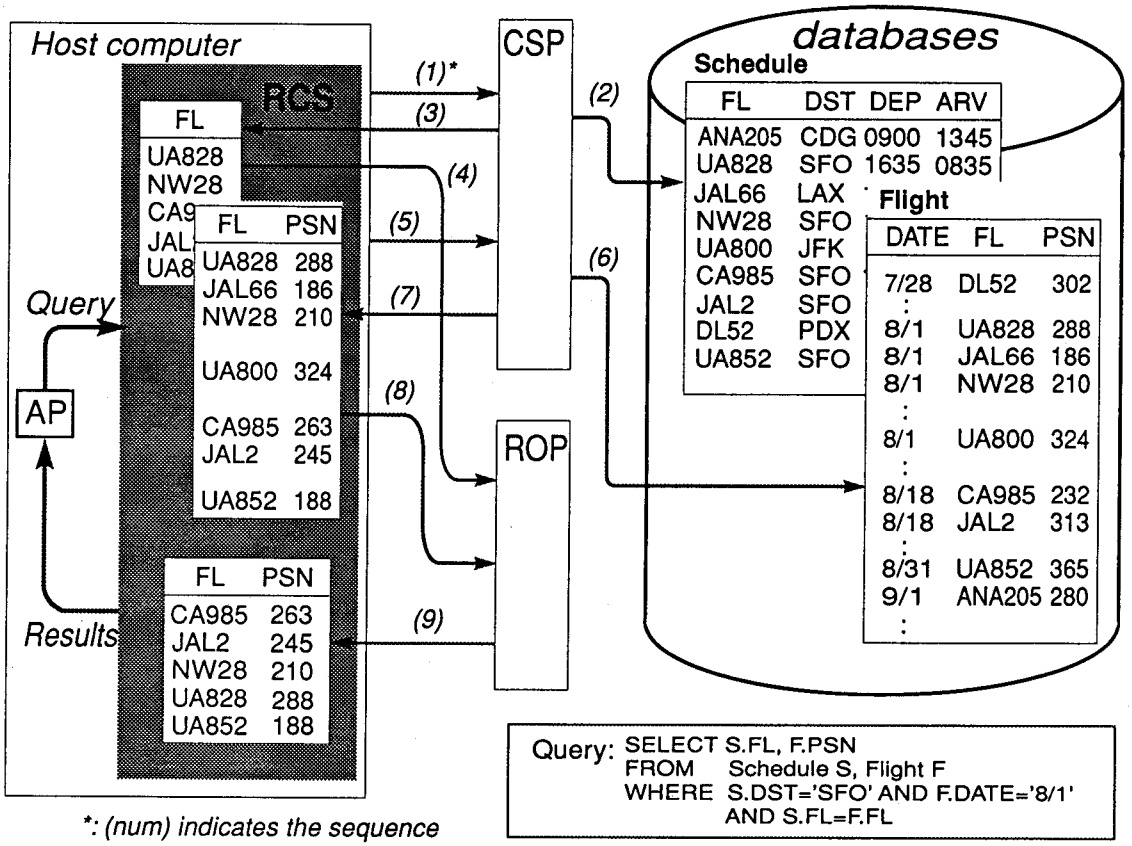


図 5.3: RINDA を用いた問合せ実行例

手順 (5)–(7): "Flight" 表を条件 DATE='8/1' で検索するコマンドを CSP に送る．検索結果は，RCS の管理するバッファに返却される．

手順 (8)–(9): ROP に対して 参照ソート コマンド (4.3参照) と，"Flight" 表の検索結果を送る．ROP は結合の対象とならない行をふるい落としてソートした結果を返却する．

手順 (9): 以上の処理によって，RCS は少なくとも問合せ条件を満足する，"Schedule" 表と "Flight" 表の検索結果をフライト番号 FL でソートした形で受け取る．RCS は，これら 2 表をフライト番号 FL でマージ結合し，結果を SQL で指示されたアプリケーションプログラムに返却する．

RCSは，1つの表を複数ディスクに分散格納して，それらに対する検索を複数のCSPで並列に実行するように制御できる．この並列処理により，手順(2)と(6)の検索時間を短縮できる．さらに，手順(4)と(5)–(7)はROPとCSPに対する独立した処理命令であるから，これらを並列に実行することもできる．複数のCSPがあって“Schedule”表と“Flight”表が異なるCSPの下に格納されている場合には，手順(1)–(3)と(5)–(7)を同時に実行することもできる．この様に，種々の並列処理が実現できることが，CSPとROPを分離してそれぞれホスト計算機と接続した理由である．現在のRINDAでは，複数CSPによる並列検索がRCSの機能として実現されている．

5.3 関係演算高速化プロセッサ (ROP)

5.3.1 基本設計

結合対象となる表を予めソートしておくことによって，結合演算自体の処理量を削減できることを第4章で示した．結合演算以外の関係演算でも対象となる表(一時表)がソートしておくことで全体の処理を大幅に高速化できる場合がある．例えば，SQL文でDISTINCTが指定された場合の重複行を除去する処理は，ソートされている場合は行数に比例する線形時間で処理できる．また，検索結果が複数行の場合に，特定の列(属性)に関してソートして出力したいという利用者要求も高い．これらを考慮して関係演算高速化プロセッサでは，2個の表の結合処理のような完結した処理を実現するのでは無く，表5.2に示すようなソートを中心とする関係演算の基本機能をハードウェア化することとした．ROPで実現した基本機能の概要を示す．

ソートに関して： 第3章で述べたマルチウェイマージソータをハードウェア化する．大容量のワークメモリを半導体メモリで実現することで，高速・大容量ソータを実現できる．

ジョインに関して： 第4章で述べた3フェーズソートマージ結合法をハードウェア化する．片ハッシュ結合法，両ハッシュ結合法のいずれも高速化するために，ふるい落としのためのビットアレイ操作とソートの組み合わせによって，設定ソート コマンドや 参照ソート コマンド等，種々のコマンドが派生する．

グループ化に関して: ソート機能の派生機能として，重複除去機能とグループ化計数を実現する．ソートした結果である行の並びの前後を比較して，重複している行を削除するのが重複除去，重複除去した上で重複している行の数を計数して，その値を属性として返却するのがグループ化計数である．ここで，重複とは，ソートするために指定された列の属性値の並び (キー) が一致していることである．

グループ化機能は，ソートの後処理として ROP のハードウェアで実現するのが容易なこと，DBMS 側のソフトウェア処理ではソートされたデータを再びスキャン (走査) する処理量が大きいことからハードウェア化することとした．

関係データベース処理は，二次記憶およびホスト計算機上でのデータの格納単位は行であるのに対し，専用ハードウェアでの処理の対象はキーであるという処理対象の 'ミスマッチ' が存在する．このため，行とキーとの相互変換が必要となる．行とキーでは，構成している属性の書式および属性の順序が異なる．ROP では，行とキーとの相互変換処理を，独立した専用ハードウェアとして搭載し，行の入出力と内部処理がパイプライン処理できる構成とした．また，ROP 内部に搭載した大容量メモリ (ワークメモリ) に，キーに変換する前の行を格納しておくことにより，変換作業に伴う二次記憶へのアクセスをなくした．

5.3.2 ROP のブロック構成

ROP のブロック構成を図 5.4に示す．ふるい落としブロック，ソートブロックは各々 3 フェーズ・ソートマージ結合法での対応する機能を実現している．ソートブロックは，グループ化計数機能も実現している．

ROP の構成で特徴的なことは，キー抽出→ふるい落とし→ソート，および，ソート→出力編集は各々行単位のパイプラインで処理するために，キー抽出も専用ハードウェア化していることである．キー抽出回路は，行から ROP での処理対象となる属性を抽出し，ソート / 結合の重みの順に属性を並べてキーを構成する．さらに，キーを後述する内部キー形式に変換する．以上のキー抽出機能に加えて，ROP で処理した結合キーあるいはソートキーに対応する行をページ形式に編集する出力編集機能も

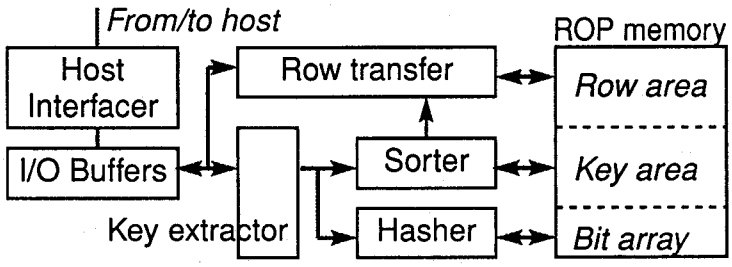


図 5.4: ROP のブロック構成

備えている．以下に ROP の特徴を示す．

- マージ準備段階と出力マージ段階の処理は，本体装置とのデータ転送に追従して処理する．ROP への入出力データは、ナルカラムや可変長カラムを含むデータベースの一時表であり、専用ハードウェアを用いてカラム抽出、コード変換等を行って固定長の内部キーに変換する．
- マージの繰り返しでソートするため，ソータ (ソートブロック) へのキー入力完了後，キーの出力を開始するまでに ' 遅れ時間 ' がでる．この ' 遅れ時間 ' は，ソートするレコード長が一般に行長より短かいこと，中間マージ段数が最悪でも 2 段と少ないことから，ROP のデータ転送時間の 1 割以下にできた [76]．
- 比較ユニットの連結機能により，ソートできる内部キー長の上限を 250 バイト程度まで拡張した．また、大容量の DRAM チップを用いて 64MB を超えるワークメモリを高密度に実装した．

5.3.3 内部キー方式

一般に，表は複数の異なる属性をもつ列からなり、属性値のタイプには、整数、符号有り／符号無しの十進数や文字列等がある．また、属性値として値が未定のナルが許されている場合もある．これらの多種多様な属性の集まりからなる行を、直接ソートする試み [29] も成されているが、限られた属性の 1 つのフィールドを対象にするのでハードウェアの複雑さから実用上の限界があった．

ROP では、フィルタフェーズにおけるハッシュ関数の入力キー，および，ソート

フェーズで使用するキーを、ROPの内部キーとして抽出する専用ハードウェア(キー抽出回路)を設けている．この内部キーは、キーの先頭位置からの比較で大小関係が判定できる絶対値数で，可変長文字列やナル値は定義長で固定長化している．内部キーを専用ハードウェアで作成することにより，異なる属性を持つフィールドを複数個連結したキーを，高速にふるい落としあるいはソートできる．

5.3.4 ジョインの動的最適化方式

結合対象となる表をCSPを用いてアクセスする際に、選択・射影後の一時表に含まれる行数をカウントし，この値を基に最適な結合方式を決定する動的最適化機構を実現している[23]．第1および第2の一時表が十分に小さく、ディスク上にワーク域を作成せずに結合できるときは，ROPを使用せずにCSPで得られた検索結果の一時表をホスト計算機上でネストループ結合する．

上記以外の場合は，ROPを用いて3フェーズ・ソートマージ結合する．第1表のサイズが第2表のサイズより十分に小さいときは片ハッシュ結合法を、上記以外、すなわち第1および第2の表のサイズがともに大きく均衡している場合，および，ソータ容量を超えてオーバフローが起きる可能性がある場合は，両ハッシュ結合法を使用する．

5.3.5 作業メモリ方式

ROPでは、行データから抽出した内部キーを用いてふるい落としおよびソートを行う．処理結果は一時表として返却することから、内部キーとは独立に行データを格納している．内部キーの最後部に行データの格納位置を示すポインタを付与し，ソート後にポインタから行データを読み出して一時表を作成する．したがって，ROP内部には内部キー、行データ，ふるい落としで使用するビットアレイを格納するための記憶部が必要となる．これら3種類のデータを独立のメモリ装置に格納したのでは、メモリの使用効率が低くなるばかりかメモリの実装密度も低下し、小型化・低価格化を阻害する要因となる．

そこで，大容量メモリチップを高密度に実装した1つの作業メモリ(単一のアドレ

ス空間) を、行格納域、内部キー格納域、および，ビットアレイ格納域に動的に分割して使用する．ビットアレイ域の大きさは、ハッシュ値のロードファクタを一定として衝突率を低く保つために，作業メモリの一定比率の領域を割り当てる．残りの領域を行格納域と内部キー格納域に分割して使用する．行の長さは可変長で，最悪条件では行ごとに長さが異なる場合がある．このため，行を格納する毎に作業メモリのオーバフローを検出する回路を設けて，真に作業メモリが満杯になるまで処理できるように設計してある．

5.4 内容検索プロセッサ (CSP)

5.4.1 基本設計

SQL の検索条件は，述語と呼ばれる真偽判定の最小単位の条件式をブール演算子 AND，OR，NOT で結んだ論理式に変換される．CSP は内容検索の実行において，SQL の検索機能を基本的にそのまま実行することにより，ホスト計算機の負荷を極力オフロードする．しかし，全ての属性に対する全ての述語をハードウェアで実行することは無理がある．このため，一般的なアプリケーションで頻繁に使用される述語に限ってハードウェア化し，使用頻度が低い一部の機能はホスト計算機上の DBMS で実行する分割実行を基本的な方針とした．表 5.1に示した CSP の機能で，<value> と <value list> は整数とデシマルの 2 つの属性，<pattern> は 1 バイト文字列 (ASCII) と 2 バイト文字列 (日本語) のための可変長ワイルドカード文字 (SQL の '%' と '％') をサポートしている．

SQL 文の中に CSP が実行できない述語が含まれる場合は，CSP が実行できるとこまで実行して絞り込んだ結果に対して RCS が残りの機能を実行する．この方式に適した検索条件の論理式の形式は乗法標準形[1]である．DBMS の言語処理部は，非定型問合せを乗法標準形に変換し，CSP で実行可能な述語のみからなる論理和項を論理積で結合した部分を CSP へ転送する．CSP が処理した結果に対して RCS は残りの論理和項を実行する．以下の例で，述語 $p_1, p_2, \ldots$ とそれらに関する論理式は CSP

[1]Conjunctive canonical form: 論理変数 (ここでは述語) を論理和で結合する項 (論理和項) を論理積で結合する形式

が実行し，CSP が実行できない述語 $p_x, p_y, ...$ と残りの論理式は RCS が実行する．

$$\underbrace{(p_1\ OR\ p_2\ OR\ ...)\ AND\ (...)}_{CSP\ executes}\ \underbrace{AND\ (p_x\ OR\ p_y\ OR\ ...)\ AND\ (...)}_{RCS\ executes}$$

5.4.2 2値論理演算への縮退方法

SQL ではデータ項目の値が存在しない時は，ゼロやスペースでなく NULL と表現する．これに伴い述語の判定結果は，真 (true)，偽 (false) 以外に述語の中のデータに NULL が含まれた場合の不定 (unknown) の 3 値の真偽値となり，ブール演算子 AND, OR, NOT は表 5.3の真理値表で定義された演算となる．ASLM[51] は，3 値論理を扱うために問合せを NULL を含むサブ問合せと含まないサブ問合せに分解し，サブ問合せの繰り返し実行とすることで 2 値論理で評価できるようにしている．サブ問合せの繰り返し実行を高速に行うために，ASLM は連想モジュールと呼ぶ専用ハードウェアを設けている．

RINDA では，ハードウェア量を増やさずに NULL を扱うアプローチをとる．すなわち，3 値の論理演算を 2 値の論理演算に縮退させることを考える．検索条件の判定結果により行を選択するときには，検索条件の判定結果が真の行のみが選択される．すなわち，最終的には不定は偽に置換される．表 5.3の真理値表で特に注目すべきは，NOT(不定) はやはり不定となることである．即ち，行の選択において NOT(不定) は偽であるのに，不定を単純に偽に置き換えてしまうと，NOT(偽) は真となり不合理を生ずる．したがって，検索条件の論理式に NOT がないことが，述語判定結果の不定を偽に置換し，検索条件判定における 3 値の論理演算を 2 値に縮退可能な条件である [20].

任意の検索条件を上記検索条件に変換する方法は，検索条件の論理式を NOT が 1 つの述語だけにかかる標準形に変換し，その結果 NOT がついた述語があれば，NOT を述語中に組み込むことである．RINDA システムでは，RCS が論理式を乗法標準形に変換する際に，上記 NOT の述語への組込みを行う．即ち，比較述語の場合は比較演算子を逆にし，他種の述語の場合は述語中に入れる．例えば，(NOT(列 1 > 10))

表 5.3: 3 値論理の真理値表

AND	True	False	Unknown
True	True	False	Unknown
False	False	False	False
Unknown	Unknown	False	Unknown
OR	True	False	Unknown
True	True	True	True
False	True	False	Unknown
Unknown	True	Unknown	Unknown
NOT	True	False	Unknown
	False	True	Unknown

は (列 1 <= 10) に，(NOT(列 1 LIKE '%A%')) は (列 1 NOT LIKE '%A%')) に変換する．

5.4.3 データベースの格納

データベースは，DB スペースと呼ぶ論理的な単位で格納される．DB スペースの物理的な格納は複数ディスクに分散している．個々のディスクは，割り当てられた DB スペースを複数の物理的に連続した領域に格納する．ディスク入出力は，DB スペース単位で設定されたサイズのページを単位として行う．したがって，論理的にページの集まりとして構成される DB スペースは，物理的には複数ディスク上の複数の連続領域として扱われる．

データベースに表を作成する際は，1 つの DB スペース内で物理的に連続したページ，あるいは，連続したページが確保できない時は隣接したページが割り当てられる．行を追加することによって表のサイズが大きくなった時は，一定数の隣接するページを表に追加する．結果として，表は複数ディスクの複数領域に分散して格納される．1 ページに複数の行を格納するが，1 つの行を複数のページに跨がって格納しない．各表は，表を格納しているディスクとディスクの領域に関する情報を持つ制御レコードを持ち，この情報に基づいて RCS は RINDA の検索を制御する．

RINDA は，ディスクの連続領域を単位として複数 CSP による並列サーチを実現

している．図5.1に示すシステム構成で並列サーチを高スケーラビリティで実現するための条件は，検索対象となる表に関して，

- 個々の CSP 配下に格納されたページ数が等しいこと
- 個々の CSP がサーチした結果のデータ量が等しいこと

である．RINDA システムが対象とする非定型検索は，検索条件を予め知ることができないので後者の条件を満たすことは一般に困難である．しかし，検索条件に合致する行数が検索対象の行数と比較して十分小さい，すなわちヒット率が小さい場合には，サーチ時間は前者の条件でほぼ決まる．このため，前者の条件を満たすように，行を格納要求順にページに格納し，ページ単位で循環的に各ディスクの連続領域に格納することとした．

5.4.4 入出力の非同期化

サーチの実行は，ディスクの回転に同期して，読み出されるデータの流れに沿って検索条件を判定するストリーム処理方式 (同期転送方式) と，ディスク装置からの読み出しデータをページ単位でバッファに格納した後，検索条件を判定するページサーチ方式 (非同期転送方式) に分類できる．CSP では以下の理由で CSP の入出力を非同期にできるページサーチ方式を採用した．

- 汎用ディスク制御装置はページ単位で，記録データのエラー検出・訂正機能を有している．本機能はページの全データを読み出した後に有効であり，エラー訂正の為にはページがバッファメモリ上に存在している必要がある．
- ストリーム処理方式では，ページの先頭から有効データを格納するなど処理方式に相応しい格納構造を採る必要があるが，ページサーチ方式ではページ内の格納構造を自由に採ることが可能である．RINDA システムでは既存 DBMS のデータベース格納構造を変更しない．
- 古典的なストリーム処理方式が提案された当時は，メモリ素子が高価でページバッファを複数個設けることはコスト的に無理であった．現状はメモリ素子の

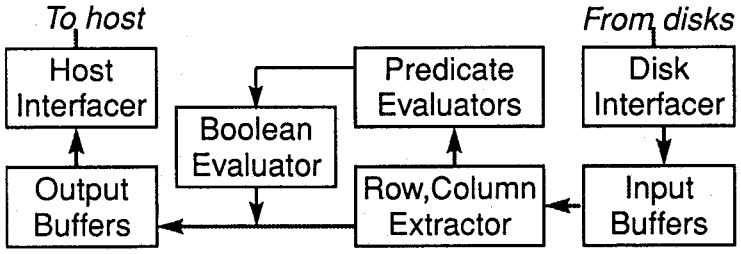


図 5.5: CSP のブロック構成

低価格化が進んでおり，ページバッファ用メモリのコストに占める割合は大きくない．

CSP はディスクからシリンダ単位のマルチトラックリードでデータ読み出しを行い複数の入力バッファに格納し，ページ単位でデータ読み出しと検索条件の判定をパイプライン処理する．このため，ストリーム処理方式と実効的に同等なデータ転送時間内のサーチ (オンザフライ) を実現している．

5.4.5 CSP のブロック構成

CSP のブロック構成を図 5.5に示す．CSP はホスト計算機／ディスク装置とのインタフェースを制御するホストインタフェース部 (Host Interfacer) ／ディスクインタフェース部 (Disk Interfacer)，入出力ページを一時格納する入／出力バッファ (Input/Output Buffers)，入力バッファ中から検索対象行の必要列を抽出する行・列抽出部 (Row,Column Extractor)，述語を判定する述語判定部 (Predicate Evaluators)，検索条件の論理式を判定する論理式判定部 (Boolean Evaluator)，から構成される．

CSP は，RCS が作成しチャネルコマンドで転送してきたオーダ (制御情報) に従ってサーチを実行する．オーダ中には，サーチ対象の DK アドレス，ページアドレス，ページ容量などの物理制御情報と，表 5.1の検索機能を指定した論理制御情報が含まれている．CSP は受け取ったオーダの物理制御情報をもとにディスク装置に対しチャネルコマンドを発行し，サーチ対象のページをマルチトラックリードにより連続的に読み出す．同時に，論理制御情報に基づき検索条件の判定と出力列の抽出を実行する．

この動作を間断なく行うために，ディスクから読み出したページを格納する入力バッファと検索条件合致行の抽出列をホスト計算機に転送するための出力バッファは複数個を交代で使用する．出力バッファ中に検索結果のデータがページ単位にまとまった段階で，CSP 内部の処理とは非同期に検索結果をホスト計算機に転送する．

RINDA システムのデータベースは可変長データを含んでいるため，行ごとに行中の長さ情報に基づき，述語の対象列や出力列の相対アドレスを計算する必要がある．行・列抽出部でアドレス計算をしながら，対象列を抽出して述語判定部と出力バッファへ転送する．なお，計算した相対アドレスは，出力行ごとに付加情報として行データに添付する．これにより，CSP の処理結果を DBMS が操作する時のアドレス計算を容易にする．

述語判定部は，LIKE 述語判定用の文字列比較回路と他の述語を判定するデータ比較回路から構成される．文字列比較回路は，パタンマッチ LSI で構成され，複数の任意長の文字列キーと入力データの部分一致を判定する．データ比較回路は，キーと入力データの大小 (一致) 比較を判定する．データ比較回路では，ほとんどのデータ型をサポートしているが，使用頻度の低い浮動小数点型はサポートしておらず，DBMS のソフトウェアで処理する．

論理式判定部は，述語判定部から受け取った結果を論理式に適用すると同時に，行・列抽出部が抽出した必要列を出力バッファに格納する．探索条件が偽の場合は実行中の (または完了した) 列抽出処理をアボートし，次の行の抽出列を格納途中の前行と同一アドレスに上書きしていく．同一行の述語判定と列抽出を同時処理することにより，入力バッファから行読み出しの重複を避けている．

5.5 評価

5.5.1 評価条件

RINDA による非定型検索処理の高速化効果を拡張ウィスコンシン・ベンチマーク [4, 14, 72] により評価した．このベンチマークでは、単一ユーザ環境で問い合わせの応答時間を計測してシステム性能を評価する．

表 5.4: 評価に用いた問合せ

Query	Type	SQL statement
Selection	1 row	SELECT * FROM THUK WHERE UNIQUE2 = 999
	1 %	SELECT * FROM THUK WHERE UNIQUE2 >= 500000 AND UNIQUE2 < 510000
	10 %	SELECT * FROM THUK WHERE UNIQUE2 >= 500000 AND UNIQUE2 < 600000
Join	AselB	SELECT * FROM HUNKA A, HUNKB B WHERE A.UNIQUE1 = B.UNIQUE1 AND A.UNIQUE1 < 10000
	CselAselB	SELECT * FROM HUNKA A, HUNKB B TENKA C WHERE C.UNIQUE1 = A.UNIQUE1 AND C.UNIQUE1 = A.UNIQUE1 AND A.UNIQUE1 < 10000 AND B.UNIQUE1 < 10000
Minimum	Scalar	SELECT MIN(UNIQUE1) FROM HUNKA
	Group-by	SELECT MIN(TWOTHOUS) FROM HUNKA GROUP BY HUNDRED
Count	Scalar	SELECT COUNT(*) FROM HUNKA
	Group-by	SELECT COUNT(*) FROM HUNKA GROUP BY HUNDRED

評価に用いた表は，表 HUNK のサイズが 10 万行，THUK のサイズが百万行であり，行の長さはシステムの制御情報を除いて 208 バイトである．評価システムの構成は，ホスト計算機が小型汎用機の DIPS-V30E[85]，CSP: 2 台，ROP: 1 台からなる RINDA，および，容量: 1.3G バイトでデータ転送速度: 3M バイト / 秒のディスクおよびディスク制御装置が各 2 台である．ホスト計算機と RINDA 間は 3 Mバイト / 秒のチャネルで接続されている．

問合せ実行に要する経過時間は，問合せ文の実行開始から，処理結果をディスクに一時表として格納し終えるまでの時間をホスト計算機側で測定した．実際のアプリケーション実行環境での性能を測定するために，SQL で規定された 1 行ごと検索結果を返却するフェッチを繰り返している．評価に用いた問合せ文の一覧を表 5.4に示す．問合せは，単純検索，結合，最小値関数，計数関数の 4 種類である．このうち計数関数は，RINDA ハードウェアの性能を評価するために追加した問合せであり，検索結果の行を転送せずに行数をカウントしてアプリケーションに返却する．

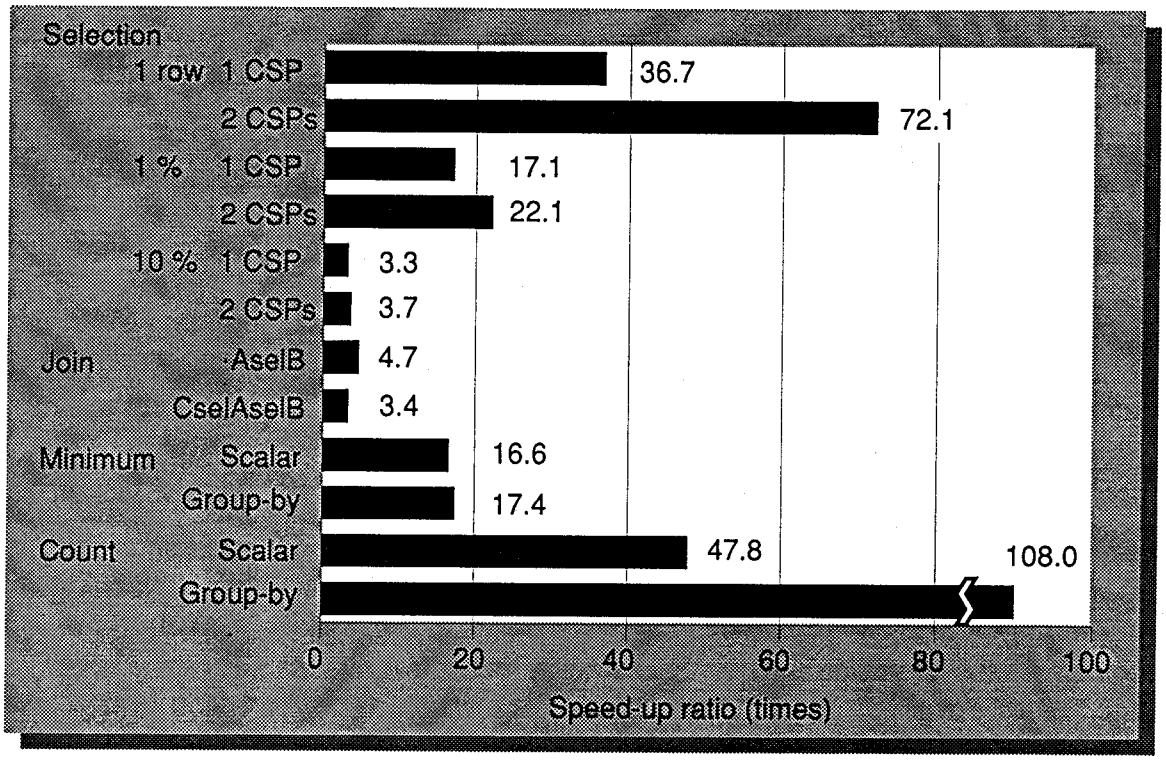


図 5.6: RINDA を用いた速度向上効果

5.5.2 評価結果

RINDA を使用することによる処理速度の向上度を図 5.6に示す．処理速度の向上度は，RINDA を使用しないソフトウェアだけの処理で得られた応答時間を，RINDA を使用した場合の応答時間で割った値である．単純選択 (1 行，1%，10%) は，1 台あるいは 2 台の CSP を使用して，最小値関数および計数関数のスカラーは 2 台の CSP を使用している．それ以外の問合せは，2 台の CSP と 1 台の ROP からなる RINDA システムを使用している．

表 5.5は，RINDA の問合せ応答時間の実測値を示しており，DeWitt 等によって測定された他の市販データベースマシンの実測値 [4, 72] と比較している．ここでは，検索結果をフェッチするのでは無く，作業用のディスク装置に一時表として格納するまでの時間を測定している．DBC/1012 と Gamma は，それぞれ 24 個と 17 個のプロセッサで構成 (第 1章の従来技術参照) されている．これらのデータベースマシンでは，

表 5.5: 問合せ実行時間の比較 (単位: 秒)

Query	Type	Rinda	DBC/1012	Gamma
Selection	1 %	52.5	213.1	134.9
	10 %	92.0	1106.9	181.7
Join	AselB	136.8	235.6	35.8
	CselAselB	128.5	95.7	37.9
Minimum	Scalar	31.7	18.3	15.5
	Group-by	47.3	27.1	1934

問合せ対象となる表を複数のプロセッサに分割格納し，単一の問合せを複数プロセッサで処理する粗粒度並列 (第 6章参照) を行なっている．

5.5.3 考察

図 5.6によれば，CSP を用いた単純選択の速度向上度は，検索結果の行数に依存していることがわかる．2 台の CSP を用いて，検索結果が 1 行の場合は 72.1 倍の速度向上が達成されているが，検索結果が 10% と多い場合は 3.7 倍の速度向上しか達成できていない．この理由は，図 5.2に示した RCS が，検索結果を 1 行づつアプリケーションプログラムに渡す転送処理で多くの CPU 処理が必要とされているからである．つまり，検索結果が多い時は，ホスト計算機で逐次処理するフェッチに多くの時間が消費されるため，RINDA ハードウェアの高速化効果が相殺されている．スカラ計数関数は，検索結果の行数を計数する機能を CSP が直接実行しているため，スカラ最小値関数と比較して高い速度向上度を達成できている．

ROP による速度向上度は，図 5.6の最小値関数と計数関数のスカラとグループ化を比較することで明らかになる．グループ化のためには，スカラの場合の処理に加えてソートが必要になるが，ソートによる速度向上度の低下はなかった．つまり，ROP によるソートは，ホスト計算機によるソートより数十倍高速である．AselB 結合は 10% の単純選択と比較して同等の速度向上が達成されている．これら 2 つの問合せは，同数の行をアプリケーションに返却していることから，両者の速度向上度を比較することで ROP の高い性能を確認できる．この時の結合処理は，第 4章に示した片ハッシュ

結合法を用いている．結合対象の第1表はCSPで1/10に選択され，第2表はROPのふるい落としで1/10程度に削減され，それぞれソートされていることから，ホスト計算機での結合処理の演算量が大幅に削減されているといえる．

表5.5でRINDAを他のDBC/1012やGammaと比較すると，選択性能は十分に高速であると言えるが，結合や最小値関数の性能が低いことがわかる．これらの問合せでは，ホスト計算機の処理量が多く，評価に用いた小型汎用計算機では処理時間が長くなるのが原因である．

RINDAシステムの残された課題は，ハードウェアの高い性能向上効果を相殺しているフェッチ処理の高速化である．RINDAハードウェアは，サーチやソート単体では20から50倍の速度向上を達成しているが，10%検索のようなフェッチ行数の多い問合せでは，その効果が3倍程度まで低下してしまうのが現状である．したがって，検索対象となる表のサイズが大きく検索結果が少ない場合に，RINDAの効果が大きいといえる．また，データベースを更新するトランザクションはソフトウェアで処理することから，更新処理中の検索などの並行処理性能を高めることも今後の課題である．

5.6 まとめ

関係データベースの非定型問合せを高速化することを目的とするデータベースプロセッサRINDAの基本アーキテクチャ，および，RINDAのハードウェア構成要素である関係演算高速化プロセッサと内容検索プロセッサの構成法を述べた．

関係演算高速化プロセッサROPは，ソート，ジョイン，グループ化を高速化する専用ハードウェアであり，一次元アレイ構造のソートアレーにおける並列比較，ジョインにおける行単位のパイプライン処理等の細粒度並列処理を専用ハードウェアで実現している．ROPではソートやジョインで使用する内部キーを行から抽出するキー抽出処理も専用ハードウェア化しているため，キー抽出→ふるい落とし→ソート，および，ソート→出力編集を各々行単位でパイプライン処理している．乗算重ね合わせ法を適用したふるい落とし回路の内部は，行よりも更に細かい単位でパイプライン処

理しており，行を単位とするROPの入出力処理に完全に追従している．

一方，内容検索プロセッサCSPは，IO負荷の大きいサーチをディスク制御装置の位置で直接実行する専用ハードウェアである．ハードウェア量を削減して実現の容易性を高めるために，DBMSは与えられた検索条件の論理式を乗法標準形に変換し，CSPが実行可能な部分はCSPが高速に処理し，CSPが実行できない部分はDBMSが引き継いで処理する方式とした．上記論理式の変換過程において，DBMSは論理式をNOTを含まない乗法標準形に変換する．これにより，SQL仕様に基づく3値論理の検索条件判定を等価的に2値論理に縮退可能となり，CSPの論理式判定の回路規模を削減した．

CSPはシリンダ単位のマルチトラックリードでページの連続読み出しを行い，ページ単位でデータ読み出しと内容検索をパイプライン処理する．さらに，一つの表を複数の磁気ディスク装置に水平分割して格納し，複数(n台)のCSPで並列サーチすることでサーチ時間を$1/n$に短縮する．

以上示したように，RINDAでは，比較的粒度の小さい細粒度並列処理を中心に，サーチ，ソート，ジョインで必要な種々の演算を専用ハードウェア化している．これらの専用ハードウェア化した処理は，行あるいは行より細かい単位でのデータ並列，および，パイプライン処理によって，十分高速に実行できており，従来ソフトウェアで実行した場合と比較し，数倍～100倍以上の高速化を図ることができた．

第 6 章

マルチマイクロによるデータベース並列処理法

6.1 まえがき

本章では，資源共有型マルチプロセッサシステムでの実現を前提とする粗粒度並列処理法 [64, 66] を論じる．

非定型問合せの応答時間は，データベース規模の拡大と問合せの複雑化によって増大する傾向にあり，問合せ応答時間を短縮する高速処理が必須となってきている．一方，OLTP (Online Transaction Processing) に代表される定型的な問合せや更新処理は，応答時間の短縮より同時走行性の向上要求が強い．さらに，データベース利用が高度化するに伴ない，販売在庫管理等を行なう応用情報システムでは，定型的な日常業務と並行して非定型問合せを実行することで，より鮮度の高い情報を得る必要が出てきた．このため，定型的な問合せや更新処理と非定型問合せを混在して実行する要求が高まってきた．

このような要求を背景に，関係データベース処理を高速化するデータベースプロセッサが実現されてきている．ICL 社の CAFS[3] や前章に示した RINDA は，非定型問合せの高速化を狙いとした専用プロセッサであり，専用ハードウェア化した細粒度並列によって高速化を達成している．しかし，これらの専用プロセッサでは，定型的な問合せや更新処理と非定型問合せを混在して効率よく実行することができていない．

一方，複数のプロセッサによる並列処理によって，非定型問合せの高速化と定型・非定型の混在実行を両立させる研究も盛んである [7, 15, 9, 71]．その理由は，

- 非定型問合せは，処理対象となるデータが大量で膨大な処理時間を要する．

- 表をスキャンするサーチは，データ並列によって複数プロセッサで並列処理することができる．また，ジョインを含む比較的複雑な問い合わせでも，複数のフェーズに分割すればフェーズ内はデータ毎に独立に処理できる [69].
- 定型的な問合せや更新は，それを単位とするトランザクション間並列処理が可能であり，複数プロセッサを用いてスループットを向上できる．

ためである．このようなマルチプロセッサによる並列処理を実現しているシステムとして Bubba[7]， Gamma[15]， Tandem[78, 18] などがある．これらのシステムは資源非共有型のマルチプロセッサシステムを前提としており，データ格納時にデータのプロセッサへの配分を静的に決定している．このため，各プロセッサに格納されるデータ数や選択されるデータ数が異なることに起因して各プロセッサの処理時間にスキューが生じる．この場合，最も処理時間の長いプロセッサによって全体の応答時間が決まるため，プロセッサ全体の処理能力を有効に利用できないという問題がある．また，資源非共有型のマルチプロセッサシステムは，プロセッサ間の通信と同期のためのオーバヘッドが大きく，トランザクション内並列による応答時間の短縮とシステムのスループット向上を阻害する要因となっている [16].

これに対して，資源共有型マルチプロセッサシステムは，資源非共有型システムより少ないコストで全てのプロセッサに負荷を配分できる．このため，静的に負荷を配分する場合よりも性能向上を期待できる [70]．プロセッサ間通信のオーバヘッドも一般的に小さい．ここでの課題は，並列処理できるタスクあるいはプロセスを均一なサイズ (処理量) で作成することである．例えば， B-tree インデックスを用いた範囲検索を複数のタスクに，かつ，各々のタスクが独立に処理できるように分割することは非常に難しい．作成したタスクを各プロセッサに負荷が均等になるように配分する負荷配分法も重要である．

本章では，資源共有型マルチプロセッサ上で非定型問合せを並列処理するための粗粒度並列処理法 [64] を論じる．特に，サーチとジョインを高速に処理するための負荷配分法 [22]，インデックス操作を伴う範囲検索と更新のための並列処理法 [24] を述べる．従来の負荷配分法 [86] では， 1 回に配分するデータ数，即ち配分単位が固定で

あり，配分単位の大きさによって負荷配分のオーバヘッドとプロセッサのアイドル時間が変化していた．提案する負荷配分法は，処理の進行にともなって配分ページ数を動的に変更することで性能の変動を小さくし，しかも負荷配分のオーバヘッドを削減する．また，従来の木構造インデックスは，並列処理するためのタスク分割が困難であったことから，ハッシングと木構造インデックスを融合させた新たなインデックス構造を提案する．

以下，6.2では，並列処理するバッチトランザクションとプロセッサのモデルを示す．6.3では資源共有型マルチプロセッサシステム上で負荷配分オーバヘッドを削減することを狙いとした適応型動的負荷配分法を示す．6.4ではハッシングと木構造インデックスを融合させたハイブリッド・インデックスの構成法とそこでの処理法を示す．6.5では，提案する負荷配分法とインデックス構成法の有効性を試作したプロトタイプDBMSを用いて評価し，最後に6.6でまとめる．

6.2 並列処理モデル

6.2.1 バッチトランザクション

以下の4種類のトランザクションを想定する．いずれのトランザクションもCPU負荷が高く，応答時間の短縮要求が強いトランザクションである．

(a) **単純問合せ:** 検索対象となる表に格納された全ての行に対して検索条件を適用する問合せである．検索条件を任意の列に指定できるアドホックな問合せは，あらかじめインデックスを作成できない典型的な単純問合せの例である．このような，ソフトウェア的な高速化手法であるインデックスを有効に使用できない，条件判定の繰り返し操作を必要とする問合せを単純問合せという．

(b) **結合を含む問合せ:** 結合するする2表に上記の単純問合せを行ない，得られた一時表を結合条件で連結する処理で，極めてCPU負荷の高い問合せである．ハッシュ結合法[69]は，問合せの応答時間を短縮するのに適した並列処理法である．この方法は，スプリット，ビルド，プローブの3つのフェーズからなり，それぞれのフェーズでデータ並列による並列処理が行える．

(c) **インデックスによる範囲検索:** インデックスが付与された行に範囲条件を指定した検索である．ユニークなインデックスによる 1 件検索は，従来でも極めて高速に処理することができるが，範囲条件による検索では，たとえインデックスを使用したとしても，検索条件に合致する複数の行を取り出して返却するための繰り返し処理が必要であり，多くの CPU 処理を必要とする．

(d) **インデックス更新を伴う表の統合:** 並列処理を実現・評価する上で最も興味深いトランザクションである．それぞれインデックスが付与された 2 つの表を，インデックスの統合も含めて 1 つの表にする．この処理は，インデックスを追加・更新しながら行を挿入する操作の繰り返しを必要とする CPU 負荷の高い処理である．

6.2.2 共有メモリ型マルチプロセッサ

上述した 4 種類のバッチトランザクションに共通する特徴は，処理対象とする表のほとんど全ての行を読み出すことである．これらの行データは，初期状態でディスク装置に格納されているが，大容量の主記憶装置を使ってバッファリングし，複数のプロセスで共有して使用できる．

大量の問合せ結果を出力するトランザクション (a), (b), (c) は，問合せ結果を格納するための一時表が必要である．また，トランザクション (b) は，大容量の中間結果を一時的に保持しておかなければならない．トランザクション (d) は，2 つの表を統合した結果が出力される．これらの一時表や中間結果は，全て大容量のバッファ上に保持することができる．

並列処理する個々のプロセッサは，主記憶装置へのアクセス頻度を下げるためにローカルなキャッシュを持っている [68]．キャッシュの一貫性を保つための特別なハードウェア (スヌープキャッシュ) を備えることで，高速なシステムバスに複数のプロセッサを接続可能としている．キャッシュ操作のオーバヘッドを低減することで，プロセッサ台数に対してスケーラブルなシステムを構築できる．

6.2.3 並列処理アーキテクチャ

並列処理を実現する際に重要な課題は，プロセス間通信に伴うオーバヘッドの低減と，プロセス間のスキューの削減である．一般に，これらの間にはトレードオフがあり，タスクの粒度を小さくするとスキューは小さくなるが通信のオーバヘッドは増大し，逆にタスクの粒度を大きくするとスキューが大きくなる傾向がある．プロセス間通信のオーバヘッドは，データ転送と実行制御のための通信回数に比例する．要求駆動型の並列実行制御は，実行開始要求の到着を契機として処理を開始するので，処理単位毎に実行開始要求を転送する通信回数は多いが，実行制御を厳密に行える．このため，知識ベース処理のような再帰的な処理の呼び出しとバックトラックによる実行中断が頻発する処理の実行制御に適している [41].

バッチトランザクションを対象とした関係データベースの並列処理は，データベースの表に格納された行に同一の処理，例えば検索条件との一致判定処理を繰り返し適用するのが中心であるから，データ並列によって並列処理を比較的容易に実現できる．このような大量データに対する同一処理の繰り返しを要求駆動型で並列に処理すると，データ転送要求を出す制御のためにプロセス間通信が多発しオーバヘッドが増大する．そこで，1回のデータ転送要求で，大量のデータを連続的に転送できるデータ駆動型の並列実行制御を適用することにした．この場合に，パイプライン処理している2つのプロセス間で先行プロセスのオーバランの抑止と，データ並列処理しているプロセス間のスキューの低減が課題となる．オーバランは，プロセス間通信に使用するバッファの割り当て量を制御することで抑止する．スキューの低減法は，**6.3**で詳細に述べる．

システム全体のプロセス構成を図 6.1に示す．利用者に近いサイトにクライアントプロセスを生成し，複数のクライアントプロセスからサーバプロセスである DBMS を同時にアクセスする．DBMS 管理プロセスは，トランザクション管理プロセスに対して資源の割当と回収を行う．割当対象となる代表的な資源には，プロセッサ数あるいは並行実行するプロセス数，バッファメモリの容量がある．

DBMS 管理プロセスは，最初にクライアントから CONNECT コマンドを受け取

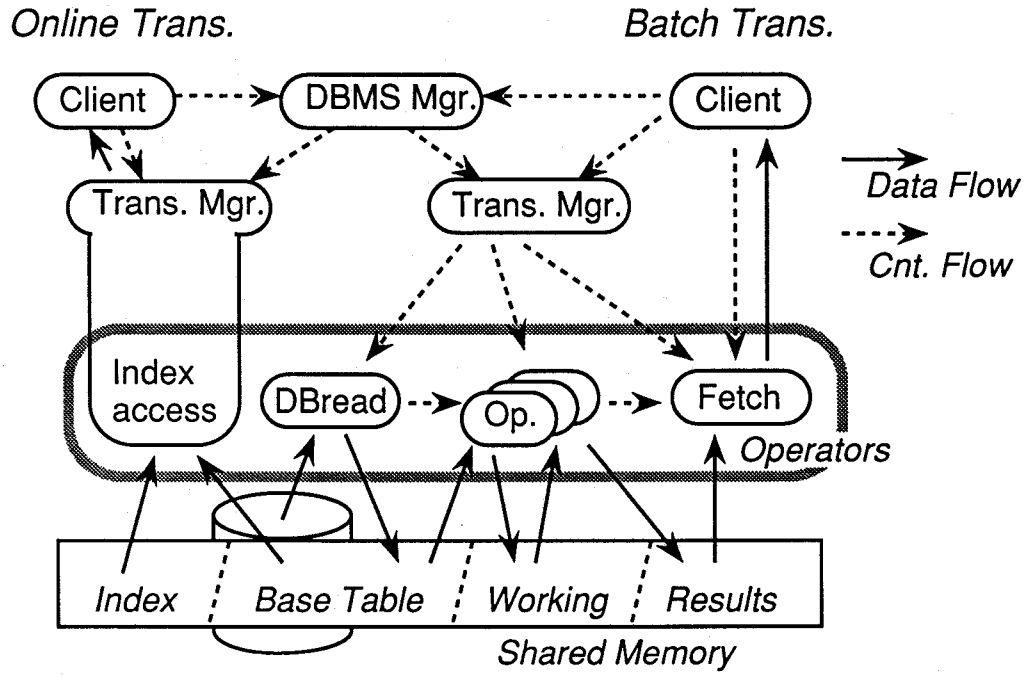


図 6.1: 並列処理のプロセス構成

りトランザクション管理プロセスを生成する．トランザクション管理プロセスは，トランザクションを解析して実行プランを作成し，必要があれば複数の実行プロセス (オペレータ) を起動して並列処理をスケジュールし，その実行を制御する．1 つのバッチトランザクションは，トランザクション管理プロセスによって制御される複数のオペレータプロセスで並列処理を行う．

トップダウンな実行制御は，並列実行の進捗を厳密に制御でき，実行途中で計算機資源 w を再割り当てすることが容易である．このようなトップダウンな実行制御を単一トランザクションの並列処理に適用した例が文献 [53] に示されている．ここでは，同時に実行している個々のトランザクション毎にトランザクション管理プロセスを設け，このプロセスが配下のオペレータプロセスの並列処理を制御する方式に拡張する．トランザクション管理プロセスが作成する初期実行プランは，トランザクションタイプ，実行の優先度から，並列処理できるオペレータプロセスの個数等を決定する．実行開始後は，実行の同期ポイントで，並行処理している他のトランザクションの処理状況等の実行環境に基づいて，資源の再割り当てを行うとした．

6.3 適応型動的負荷配分法

6.3.1 負荷配分モデル

6.2.1に示したバッチトランザクションは，データ並列による並列処理が行なえる．トランザクション (a) やトランザクション (b) のスプリット，ビルド，プローブの各フェーズは，問合せ条件に基づく条件判定を個々の行に対して繰り返し適用する操作である．そこで，トランザクション (a) に代表される表のスキャンを例に，負荷配分のモデルを仮定する．

- データベース中の表は，固定サイズの複数ページに分割して格納されている．ページの総数 n ($n >>$ プロセッサ数) は既知である．
- 各ページはどのプロセッサに対しても同じ時間で配分できる．1回の配分時間は，配分するページ数に依存しない．
- プロセッサの処理はページ毎に独立に実行でき，結果は実行順序に依存しない．ページ当たりの処理時間の最大値 T_{max} と最小値 T_{min} は，あらかじめ予測できるとする．

以上の条件のもとでページのプロセッサへの配分を決定する負荷配分法を検討する．負荷配分の目的は，単一の問い合わせの処理が開始されてから終了するまでの経過時間（以後応答時間という）をできるだけ短くすることである．並列処理を行なう場合でも，各プロセッサが本来の問い合わせ処理自体をしている時間の総和は逐次処理の場合に要する時間と同一であるから，応答時間を短縮するという負荷配分の目的は，並列処理によって生じるオーバヘッドをできるだけ少なくすることと同値である．

6.3.2 従来の負荷配分法と問題点

独立に実行可能なタスクが複数のタスク源で生成され，それらのタスクを複数のプロセッサで実行する場合の負荷配分法が文献で分類，評価されている [86]．負荷配分法は，タスク源がプロセッサにタスクを送るタスク源主導型と，タスク実行可能に

なったプロセッサがタスクを取りに行くプロセッサ主導型に分類できる．両者を比較すると，負荷配分に使用する情報のレベルが同じであればプロセッサ主導型が高い性能が達成できるとされている．そこで，**6.3.1**に示した負荷配分モデルにプロセッサ主導型の負荷配分を適用する．

(1) 各プロセッサは，一定数のページを取り出して処理する．このページ数を配分単位という．

(2) 処理を終えたプロセッサは，次の配分単位を取りに行く．全てのページが処理されるまで，配分単位の取り出しとその処理を繰り返す．

配分単位を3ページとした従来の動的負荷配分の例を図6.2に示す．プロセッサが取り出した配分単位(ページ)の処理時間はページ毎に異なるから，個々のプロセッサの負荷配分回数は実行時まで確定しない．このように配分単位を固定した動的負荷配分法では，負荷配分に要するオーバヘッド時間は以下の2種に分類できる(図6.2参照)．

(1) **負荷配分時間:** 取り出す配分単位(ページ数)を決定する時間，その間の排他制御および待ち時間，通信時間など．一般に，負荷配分時間は負荷配分回数に比例し，負荷配分回数は配分単位に反比例する．

(2) **アイドル時間:** 全てのページの配分が終わった後に，処理すべき負荷がないプロセッサが最後まで処理しているプロセッサの実行終了を待つ時間．1プロセッサ当たりのアイドル時間は配分単位の処理時間以下である．

負荷配分時間は配分単位を大きくすると減少し，アイドル時間は配分単位を大きくすると増大する．従って，負荷配分時間とアイドル時間はトレードオフの関係にあり，オーバヘッド時間を小さく抑えるには適当な配分単位を決定することが重要になる．しかし，データベースの並列処理における配分単位の最適値は以下の要因に依存して変動すると考えられる．

(a) **ページ数:** 負荷配分の回数はページ数に比例するので，負荷配分時間はページ数(表のサイズ)に比例する．一方，アイドル時間はページ数には無関係である．従って，ページ数が多くなると配分単位の最適値は大きくなる．

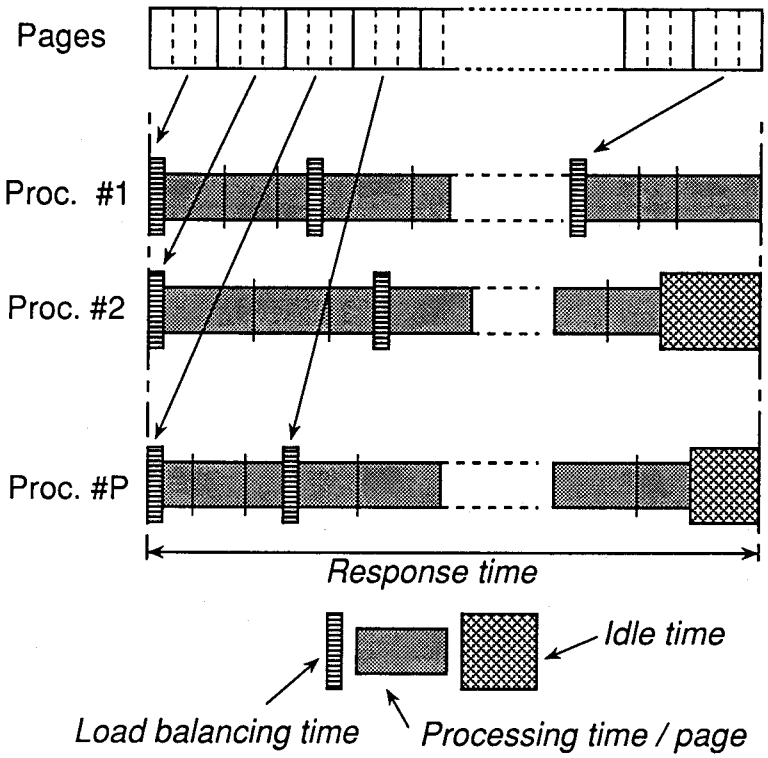


図 6.2: 従来の動的負荷配分法 (配分単位: 3)

(b) プロセッサ数: プロセッサ数が増えると，アイドル状態になるプロセッサが増えるのでアイドル時間が大きくなる．従って，プロセッサが多くなると配分単位の最適値は小さくなる．

(c) データ分布: ページ毎の処理時間は，ページ内のデータ数や選択されるデータ数によって変化する．アイドル時間は終了間際に配分されたページの処理時間に依存する．従って，データ分布によって配分単位の最適値が変化する．

一般に，データ分布は事前には判らないため，配分単位の最適値を決定することができない．他の要因に関しても，実行途中でプロセッサ数が変化する場合もあるため，配分単位を固定とする従来法では必ずしも良好な性能が得られなかった．

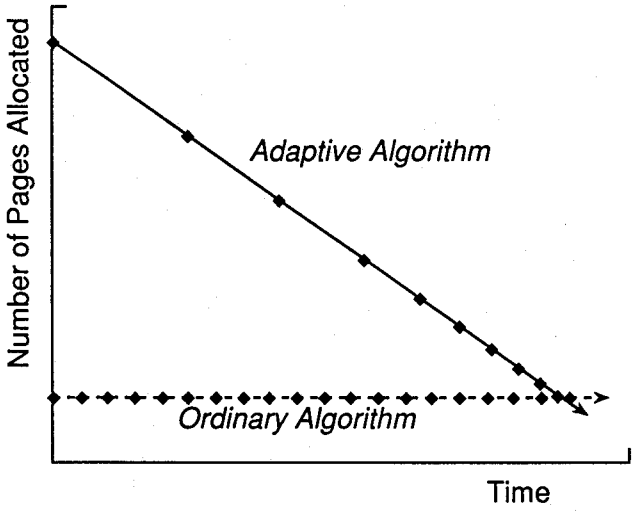


図 6.3: 配分単位の動的決定の概念図

6.3.3 適応型動的負荷配分法の提案

従来の負荷配分法の問題は，負荷配分時間とアイドル時間のトレードオフがあり，しかも，配分単位の最適値を事前に決定できないことに起因している．この問題を解決するために，アイドル時間を小さく抑えたままで負荷配分時間を大幅に削減することを考える．即ち，アイドル時間が小さい (配分単位が小さい) 領域で負荷配分時間を，その領域でのアイドル時間程度に小さくできれば，オーバヘッド時間をその領域内でほぼ一定にでき，配分単位の大きさに関係なく最適に近い性能が得られる．

6.3.2で述べたように，負荷配分時間は主に配分回数に依存し，アイドル時間は主に処理終了間際の配分単位に依存する．そこで，図 6.3のように配分単位を可変とすることで，アイドル時間を小さく抑えたままで配分回数を削減できると考えられる．

- 最初は多くのページを配分することによって，負荷配分回数を減らす．
- 処理の進行に従って配分ページ数を徐々に少なくすることによって，アイドル時間を小さくする．

次に，配分ページ数とアイドル時間の関係を定式化し，アイドル時間を許容限度以下にできる配分ページ数の決定法を求める．以下の場合を考える．

- n ページを P 個のプロセッサで処理する.
- プロセッサ i $(1 \leq i \leq P)$ に b_i ページを配分する.
- プロセッサ i が b_i ページを処理し終える前に，他の $(P-1)$ 個のプロセッサが，残りの $(n-b_i)$ ページの処理を終える.

この場合にアイドル時間が最大になる条件は，ページ当たりの処理時間の最大値を T_{max}，最小値を T_{min} とすると，

- プロセッサ i に配分されたページは全て処理時間が T_{max} であり，かつ，
- 他の $(n-b_i)$ ページは全て処理時間が T_{min} である

場合である．この時，プロセッサ i の処理時間は $b_i \times T_{max}$ であり，残りのページの処理に要する延べ時間は $(n-b_i) \times T_{min}$ である．したがって，アイドル時間の総和は次式となる.

$$b_i \times T_{max} \times (P-1) - (n-b_i) \times T_{min} \tag{6.1}$$

アイドル時間を小さく抑えたまま多くのページを配分するために，式 (6.1) のアイドル時間を与えられた許容限度 T_{idl} 以下にできる最大の b_i を求める.

$$b_i = \frac{n + T_{idl}/T_{min}}{(T_{max}/T_{min}) \times (P-1) + 1} \tag{6.2}$$

式 (6.2) を，配分の度に繰り返し用いるように書き換えると次式が得られる.

$$\begin{aligned} b_i(t) &= \{n(t) + \sum_{j=1}^{P} m_j(t) + \alpha\}/\beta - m_i(t) \\ \alpha &= T_{idl}/T_{min} \\ \beta &= (T_{max}/T_{min}) \times (P-1) + 1 \end{aligned} \tag{6.3}$$

但し，

$b_i(t)$: 時刻 t にプロセッサ i に配分するページ数 $(i = 1, 2, ...P)$
$n(t)$: 時刻 t における未配分ページ数
$m_j(t)$: 時刻 t におけるプロセッサ j の未処理ページ数 $(j = 1, 2, ...P)$
T_{idl}: アイドル時間の総和の許容限度 (但し，$T_{idl} \geq T_{max} \times (P-1)$)
T_{max}: 1 ページ当たりの処理時間の上限
T_{min}: 1 ページ当たりの処理時間の下限
P: プロセッサ数

である．なお，式 6.3で，

$$n(t) + \sum_{j=1}^{P} m_j(t)$$

が時刻 t においてまだ処理されていないページ数（時刻 $t = 0$ に相当する式 (6.2) では n）である．以上により，配分されたページの処理を終えた ($m_i(t) = 0$ となった) プロセッサが，他のプロセッサの未処理ページ数 $m_j(t)$ を収集して式 (6.3) を計算して割り当てページ数を算出し，処理を行う．

6.3.4 適応型負荷配分法の実現

以上示した負荷配分法では，配分 1 回につき $(P-1)$ 個のプロセッサから $m_j(t)$ の値を収集する必要がある．しかし，$m_j(t)$ を収集してからページを配分するまでの間に，他のプロセッサでは処理が進行するため，$m_j(t)$ の正確な値を利用することはできない．そこで，負荷配分以前に収集した値を用いて配分ページ数を決定することで，各プロセッサの残りページ数を収集する通信回数を削減する方法を示す．

未処理ページ数 $m_j(t)$ は，以下のように分解できる．

$$m_j(t) = alloc_j(t) - report_j(t) - proc_j(t) \tag{6.4}$$

$alloc_j(t)$：時刻 t までにプロセッサ j に配分したページ数

$report_j(t)$: 時刻 t までにプロセッサ j が処理し，既に報告されたページ数

$proc_j(t)$：時刻 t までにプロセッサ j が処理したが，まだ報告されていないページ数

負荷配分を行うプロセッサが式 (6.4) の計算で利用可能なのは $alloc_j(t)$ と $report_j(t)$ であるから，$m_j(t)$ の近似式として次の 2 種を利用できる．

$$m_j'(t) = alloc_j(t) - report_j(t) \geq m_j(t) \tag{6.5}$$

$$m_j''(t) = 0 \leq m_j(t) \tag{6.6}$$

そこで，アイドル時間を許容限度 T_{idl} 以下にするために，配分ページ数が式 (6.3) の $b_i(t)$ より小さくなる方向に近似する．

$$+\sum_{j=1}^{P} m_j(t) \rightarrow +0 \tag{6.7}$$

$$-m_i(t) \rightarrow -\{alloc_i(t) - report_i(t)\} \tag{6.8}$$

ここで，式 (6.7) は (式 6.6) を，式 (6.8) は式 (6.5) を近似に用いた．以上により，

$$b_i'(t) = \{n(t) + \alpha\}/\beta - \{alloc_i(t) - report_i(t)\} \tag{6.9}$$

が得られる．式 (6.3) の代わりに式 (6.9) を用いると，配分されたページの処理を終えたプロセッサ（プロセッサ i とする）では，

$$alloc_i(t) - report_i(t) = 0$$

であるから，以下の負荷配分方法が得られる．この方法は，並列処理の進行状況に応じて残りページ数から配分するページ数を動的に決定していることから，適応型動的負荷配分法と呼ぶ．

適応型負荷配分法： 時刻 t において，配分されたページの処理を終えたプロセッサ i は，次式で計算した数のページを取り出して処理する．この操作を，全てのページがなくなるまで繰り返す．

$$b_i''(t) = \{n(t) + \alpha\}/\beta \tag{6.10}$$

この方法では，近似により 1 回の配分ページ数が理論式 (6.3) より少なくなるため，配分回数は下限値より僅かに増える．しかし，配分毎に他の $(P-1)$ 個のプロセッサからデータを収集しなくて済むので，負荷配分のオーバヘッドが小さく，容易に実装できる．

6.4 ハイブリッド・インデックス構成法

6.4.1 従来法と問題点

インデックスは，データベース処理をソフトウェア的に高速化する手段であり，基本的なデータ構造は木構造かハッシュのいずれかに分類される．B-tree に代表され

る木構造のインデックスは，格納効率が高く検索時間も短い優れた特性を持っている．しかし，ノードの追加やキー値の変更を必要とするインデックスの更新は，動的な木の再構成を必要とする場合があり処理に時間がかかる．しかも，木の再構成を同時に2つのプロセスで実行することは極めて難しく，並列処理する上で直列化されるクリティカルな処理である．一方，データベースのインデックスとして使用できるハッシュは，チェインバケットハッシュ (Chained bucket hashing)，拡張ハッシュ (Extensible hashing)，線形ハッシュ (Linear hashing) 等がある．これらの方法では，検索するキーを持つ行の格納位置をハッシュ関数によって計算するため，1件検索の問合せが極めて短時間で行える．しかし，範囲検索は，ハッシュ表がキー値の順序を保つように構成されていないので実現が困難である．しかも，一般にハッシュでは格納効率を高くできず，キーの挿入や削除を繰り返すとその値が低下する．

これまで，単一の B-tree を同時にアクセスするための方法が提案されている [33, 47, 59]．Jonge らは，木を再構成する際にロックを必要としないプロトコルを提案している [31]．これらの方法を用いることで，単一プロセッサによる処理での並行処理性を大幅に向上できる．しかし，インデックスをアクセスする際のオーバヘッドの増大や，複数のプロセスが並列にアクセスを行うマルチプロセッサシステムでは，十分な並行処理性を達成できていない．

6.4.2 ハイブリッド・インデックスの提案

マルチプロセッサシステムでの並列処理に適したインデックス構造は，検索と更新トランザクション，あるいは，更新トランザクション間でアクセスが衝突するようなホットスポットがない構成である．キーの挿入や削除を行う際のロックは，その波及範囲を最小限にすることも重要である．また，データベースのインデックスでは，範囲検索に対しても有効に機能することが望まれる．

提案するハイブリッド・インデックスは，上述の優れた特性を持つ B-tree を基本とする．6.2.1に示したバッチトランザクションの (c)，(d) は，インデックスの参照や更新が頻繁に行われるから，一般的な B-tree では並列処理する上で以下の問題がある．

(1) ルートに近いノードほどアクセス頻度が高くなり，ルートノードは必ずアクセスされる．このため，ノードの位置するレベル (深さ) によって，アクセス頻度に偏りが生じる．

(2) キーの追加や削除を繰り返すとノードのオーバフローやアンダーフローが生じる．ノードの分割や統合を行なって木をバランス化させる再構成をする必要があるが，この操作は，ルートノードに向かって分割や統合操作が波及する場合がある．

(3) 一般に，トランザクションは，インデックスを参照する際はノードを保護モードでロックし，更新する際は排他モードでロックする必要がある．ロックするノードがルートに近いほど，ロックの影響を受けるノードの数が多くなり，他のトランザクションの操作を妨害する．

これらの問題は，従来1つであった木を複数のサブ木に分割することで解決できる．分割したサブ木の選択をハッシュ関数を用いてキー値から決定するインデックスの構成法を提案し，ハイブリッド・インデックス (HbX) と呼ぶ．HbX は，図 6.4に示すように複数のサブ木とハッシュ・ディレクトリで構成される．キーとポインタの組からなるインデックスのエントリは，ハッシュ関数によっていずれかのサブ木に分配され格納される．各サブ木は，B-tree と同じ構造を持つ．ハッシュ・ディレクトリは，それぞれのサブ木のルートノードへのポインタ (アドレス) を持つ．このような構成からなる HbX は，並列処理する上で以下の利点がある．

- 単一の B-tree でアクセスが集中するルートノードは複数のノードに分割されるので，アクセス競合が低下する．
- 並列処理するプロセッサの個数やアクセス競合の度合いから，必要なサブ木の個数を設定できる．
- 単一の B-tree で構成する場合と比較して個々のサブ木の高さは低くなるから，一致検索の性能が向上する可能性がある．

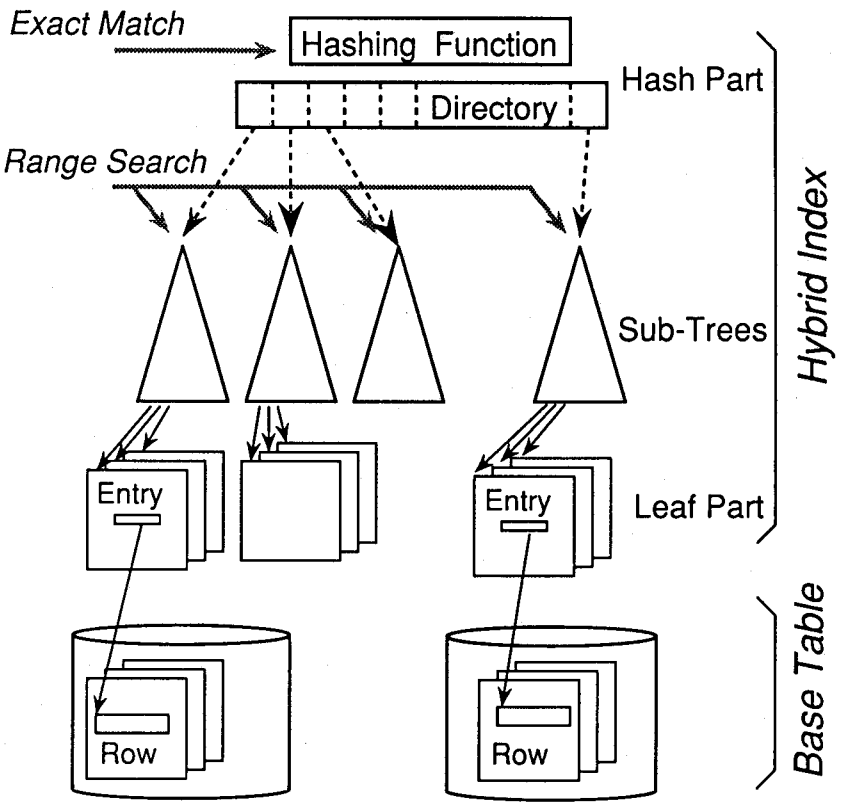


図 6.4: ハイブリッド・インデックスの構成

6.4.3 操作とアクセス制御

データベースの検索・更新操作に伴うインデックスの操作は，以下の (a) 参照，(b) 挿入，(c) 削除の組み合わせで行える．キー値の更新は，新しいキー値を持つエントリの挿入と，古いキー値を持つエントリの削除に分解する．図 6.4に示した HbX での操作を述べる．

(a) **参照:** トランザクションのタイプが一致検索か範囲検索かによって異なる参照方法を実現する．一致検索の場合は，ハッシュ関数を用いて 1 つのサブ木を決定し，そのサブ木を保護モードでロックして B-tree と同じ手法で目的とするキー値を持つエントリを探す．範囲検索の場合は，全てのサブ木を選択し，それらを保護モードでロックして探索する．この操作は，サブ木ごとに独立して複数のプロセッサで並列に処理できる．いずれの場合も処理が終了したらロックを解除する．

(b) **挿入:** 挿入するエントリのキー値からハッシュ関数を用いて 1 つのサブ木を選択し，排他モードでロックする．B-tree における挿入手順で新しいエントリをサブ木に追加しロックを解除する．

(c) **削除:** 削除するキーの値からハッシュ関数を用いて 1 つのサブ木を選択し，排他モードでロックする．B-tree と同様にサブ木に格納されているエントリを探して削除し，最後にロックを解除する．

以上の操作で重要なことは，サブ木単位でロックを確保 / 開放することである．従来，ノード単位で行っていたロック操作と比較して，サブ木単位のロック操作は実現が容易であり，しかも，実行時のオーバヘッドを削減できる．HbX では，サブ木単位のロック操作でも十分な並列実行が可能なようにサブ木の数を増やすことができる．

6.5 トランザクションの実装と評価

6.5.1 評価モデル

バッチトランザクションの性能向上を評価するために，共有メモリ型マルチプロセッサ上にプロトタイプシステムを実現した．評価モデルとマルチプロセッサの仕様を表 6.1に示す．

評価に使用した表は，ウィスコンシン・ベンチマーク [14] を基本に，初期状態で 50,000 個，10,000 個，0 個の行を格納した T1，T2，T3 である．評価するトランザクションは **6.2.1**に分類した 4 種類である．

Tx1: SELECT * FROM T1 WHERE UNIQUE2 < xxxx;

Tx2: SELECT * FROM T1, T2
WHERE T1.UNIQUE1=T2.UNIQUE1 AND T1.UNIQUE2 < 5000;

Tx3: SELECT * FROM T1 WHERE UNIQUE1 < 5000;

Tx4: INSERT INTO T3 SELECT * FROM T2;

トランザクション Tx1 と Tx2 の評価では，表 T1 と T2 にインデックスはない．Tx3 と Tx4 の評価では，関連する 3 個の表のそれぞれに，カラム “UNIQUE1” にユニーク・インデックスを設定する．Tx1 では，選択率 0% と 10% の選択率を変えた評価を行った．選択率 0% では，検索結果が出力されないことから，並列処理する検索のみの性能を測定できる．最後のトランザクションは，表 T3 に 10,000 の行を一括挿入する更新型である．インデックスは行を挿入する毎に更新され，徐々にサイズが大きくなっていく．

6.5.2 プロトタイプシステム

プロトタイプシステムは，表 6.1の仕様を持つシーケント社製 S27 マルチプロセッサ [68] 上に実現した．プロトタイプシステムは，**6.2.3**に示した並列処理アーキテクチャ (図 6.1) に基づいて，バッチトランザクションの処理系を実現した．バッチトランザクションの実行手順を以下に示す．

表 6.1: 評価モデル

データベース	Wisconsin DB[14]
行長	208B
ページサイズ	2KB
行数 / ページ	9
格納場所	主記憶
並列計算機	Sequent S27
プロセッサ	i80386(16MHz)×8
キャッシュ容量	64KB
主記憶容量	40MB
O S	DYNIX Ver. 3.0.17.9 (4.2BSD 準拠)

(1)PREPARE フェーズ: トランザクション管理プロセスは，クライアントから受け取ったコマンドのパラメータを解析して，必要な個数のオペレータプロセスを起動する．検索結果等を格納するための一時表を作成する．DB 読み出しプロセス (DBreader) は，ディスク装置に格納されたデータベースから処理に必要な表をバッファに読み込む．

(2)OPEN フェーズ: オペレータプロセスは，バッチトランザクションの処理内容に基づいて処理を開始する．処理内容は，トランザクション管理プロセスからプロセス間通信によって渡される．必要な処理が終了したら，トランザクション管理プロセスに処理終了を通知する．

(3)CLOSE フェーズ: トランザクション管理プロセスは，並列処理している複数のオペレータプロセスの進捗状況を監視して，必要に応じてオペレータプロセスに終了 / 中断を指示する．いずれかのオペレータプロセスが異常終了した，あるいは，クライアントから処理を中断するコマンドが与えられた時は，オペレータプロセスに中断を指示する．それ以外の配下のオペレータプロセスが全て正常に処理を終了した場合は終了を指示する．

(4)COMMIT フェーズ: トランザクション管理プロセスは，関係する全てのオペレータ・プロセスを削除する．更に，PREPARE で作成した一時表を消去する．

プロトタイプによる評価の目的は，提案した適応型動的負荷配分法とハイブリッド・インデックス構成法の有効性を検証し，トランザクションの応答時間の短縮を確認することである．このため，データベースは主記憶上にあるとし，主記憶上の一時表への検索結果の書き込みが完了するまでの時間を測定した．すなわち，上記の処理手順の OPEN フェーズに要する時間を測定し，問い合わせの解析やプロセス起動の時間は測定範囲から除外した．検索結果は，一時表として主記憶上のバッファに保持し，ディスクには書き戻さない．

6.5.3 単純問合せ

トランザクション管理プロセスは，DB 読み出しプロセス，選択プロセス (Selector)，フェッチプロセス (Fetcher) を生成し，その実行を管理する．3 種類のオペレータプロセス間で，パイプライン処理を実現し，さらに，複数の選択プロセス間でデータ並列を実現する．OPEN フェーズで以下の操作を行う．

(1) 選択プロセスは，DB 読み出しプロセスに対して負荷 (ページ) の配分を要求する．DB 読み出しプロセスは，適応型あるいは従来の負荷配分法に基づいて算出した数のページを配分する．プロセス間のメッセージとしてはページ番号だけを渡し，ページの実体は共有メモリを使うことで高速な通信を実現した．

(2) 選択プロセスは，トランザクション管理プロセスから渡された検索条件にしたがって行を選択し，選択した行を一時表に書き込む．一時表は複数の選択プロセスに共有されているため，排他モードのロックを確保してからページの払い出しを行う．

(3) フェッチプロセスは，ページ単位で選択プロセスが作成した一時表から，選択結果を 1 行づつクライアントプロセスに送る．

以上示した OPEN フェーズの処理時間を計測し，プロセッサ数を増やすことによるスピードアップ率を求めた．その際，クライアントに検索結果を返却するフェッチプロセスは，SQL の規定により処理がシリアライズされるため，その動作を止めた．

単純問合せの応答時間の実測値とスケーラビリティを図6.6に示す．選択率 (Selectivity) が0% と 10%，負荷配分法が適応型 (Adaptive-LB) と従来 (Ordinary-LB) の固定16ページ割り当ての場合について測定した．適応型動的負荷配分法では，最小の割り当てページ数を16とした．プロセッサ数が1の場合の応答時間は，適応型負荷配分法を用いることで選択率と関係なく1.2秒短縮できている．これは，従来の固定配分法が348回の負荷配分を行うのに対して適応型は1回の負荷配分しか行わないからである．従って，平均的な1回の負荷配分時間は3.5ミリ秒であることが分かる．プロセッサ数が8の場合では，適応型負荷配分法を用いることでスケーラビリティを10% 程度向上できている．

6.5.4 結合を含む問合せ

ハイブリッド・ハッシュ結合法 [69] は，第1表T1をバケットに分割するスプリットフェーズ，個々のバケットに対するハッシュ表を作成するビルドフェーズ，第2表T2を読み込んでハッシュ表と突き合わせ処理するプローブフェーズからなる．以上の3フェーズをそれぞれ並列化したハッシュ結合の流れを図6.5に示す．スプリットフェーズやプローブフェーズの並列処理において，表T1, T2を複数のプロセッサに負荷分散するオーバヘッド，中間結果であるバケットや最終結果の一時表に書き込む際の競合によってスケーラビリティが低下する．

スプリットフェーズとプローブフェーズにおける負荷配分法を，適応型動的負荷配分法と従来法として，トランザクションTx2の応答時間を測定した．従来法では，割り当て単位を16ページに固定した．応答時間とスケーラビリティを図6.7に示す．プロセッサ数が1の場合，いずれの負荷配分法でも1回の配分で全てのページをプロセッサに渡している．プロセッサ数が2以上の場合には，負荷配分アルゴリズムに基づいて割り当てページ数を決定し負荷配分を行う．図6.7の結果から，適応型動的負荷配分法を用いることによって，プロセッサ数が2以上の場合にスケーラビリティを15% 程度向上できることがわかった．

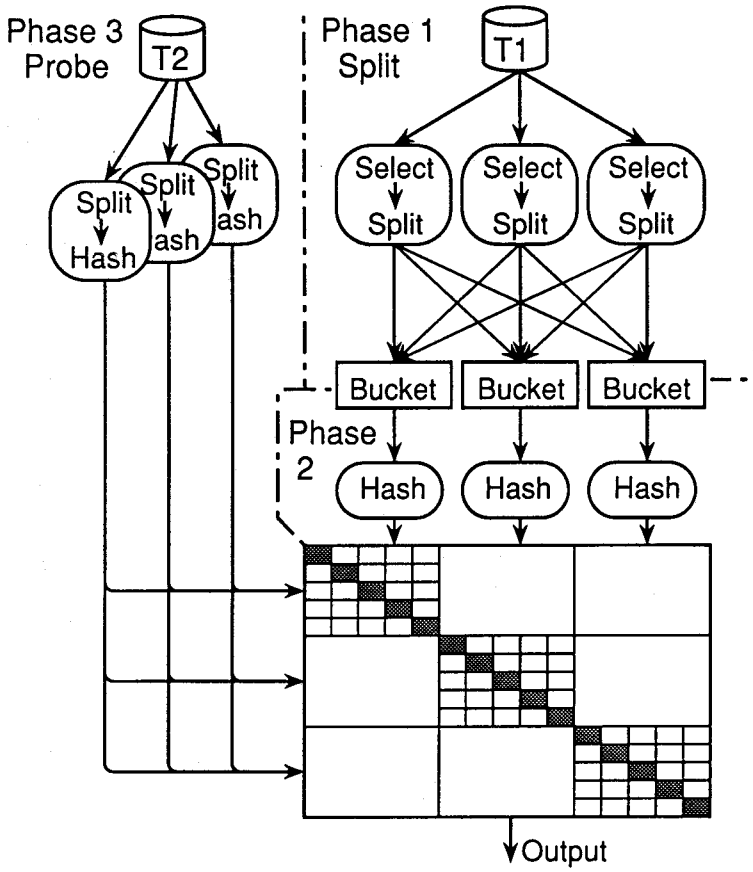


図 6.5: 並列化したハッシュ結合の流れ

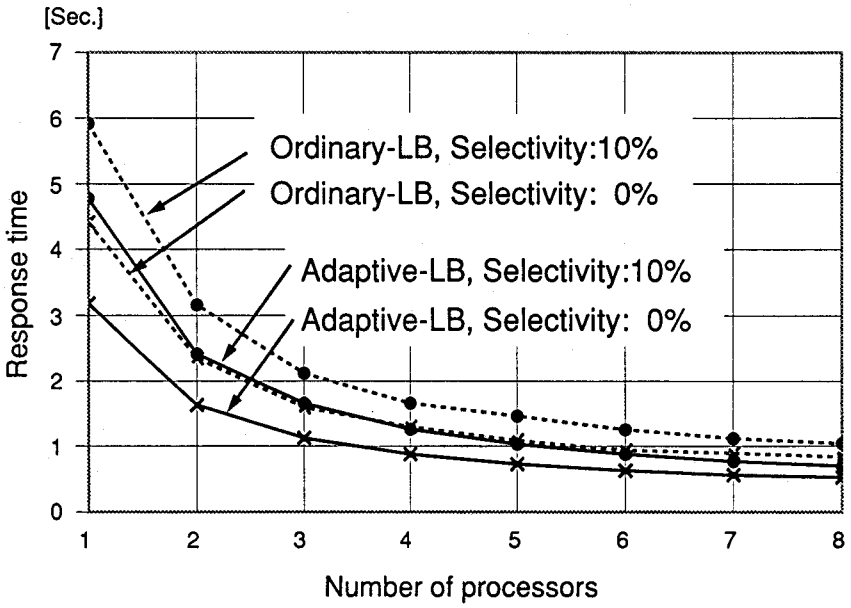
6.5.5 インデックスによる範囲検索

範囲検索の並列処理にハイブリッド・インデックスが有効性を検証するために，サブ木の数 (Nsub) とプロセッサ数 (Ncpu) を変えて，トランザクション Tx3 の応答時間を測定した．サブ木の数を 1 としたハイブリッド・インデックスは，従来の B 木に相当する．ただし，ノード単位ではなくサブ木単位にロックを管理しているので，インデックスの更新ではロック待ちの頻度が高くなるが，範囲検索では，保護モードでのロックだけであるからロックの影響は小さいといえる．

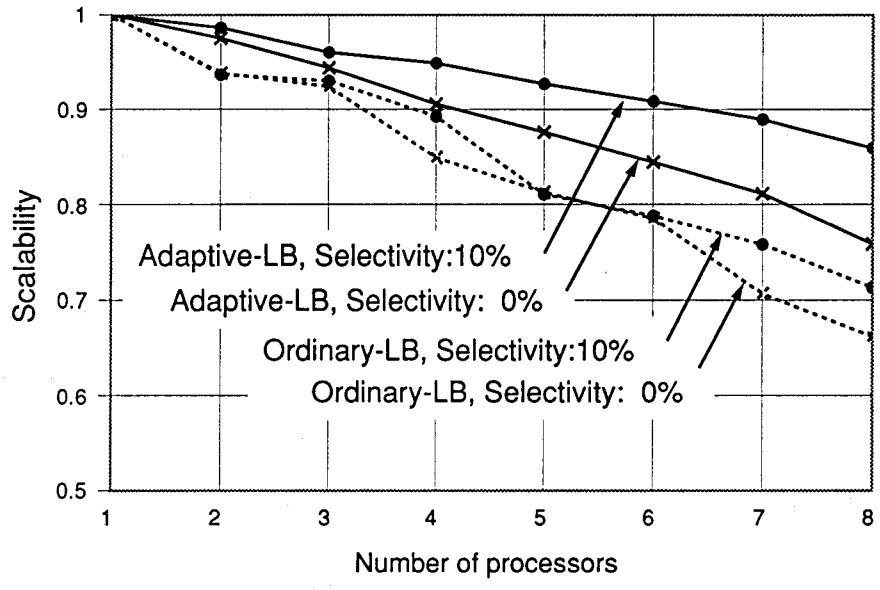
図 6.8は，サブ木の数をパラメータとして測定した選択率 10% での範囲検索の応答時間とスピードアップ率である．スピードアップ率は，単一プロセッサで単一サブ木の場合の応答時間で正規化した値である．プロセッサ数とサブ木数を増やすことでスピードアップが図れている．図から，スピードアップ率が最大となる条件は，プロセッサ数とサブ木数が等しい場合であるといえる．これは，個々のサブ木である B 木での範囲検索が，検索条件の下限値 (この場合は 5000) で木をルートから探索し，リーフに到達したらリーフ間のリンクで条件に合致するエントリのリストを作成するため，サブ木が多くなるとルートから探索する回数が増えるのがオーバヘッドになっていると考えられる．サブ木にエントリを分配するハッシュ関数としてサブ木数を底とする剰余を使用したので，検索条件 5000 以下を満足するエントリが，全てのサブ木に均等に配置されているために，スキューが小さかったことも要因である．

6.5.6 インデックス更新を伴う表の統合

図 6.9は，ハイブリッド・インデックスを用いてトランザクション Tx4 を実行した結果である．スピードアップ率は，単一プロセッサかつ単一サブ木の場合の応答時間で正規化した値である．サブ木が 1 の場合は，プロセッサ数を 3 以上に増やすと，ロック競合のために応答速度が悪化してくる．サブ木の数を増やしていくことによって，プロセッサ数に比例したスピードアップが図れている．サブ木の数がプロセッサ数よりも多い場合にも，サブ木を増やしていくことで僅かではあるが性能が向上している．

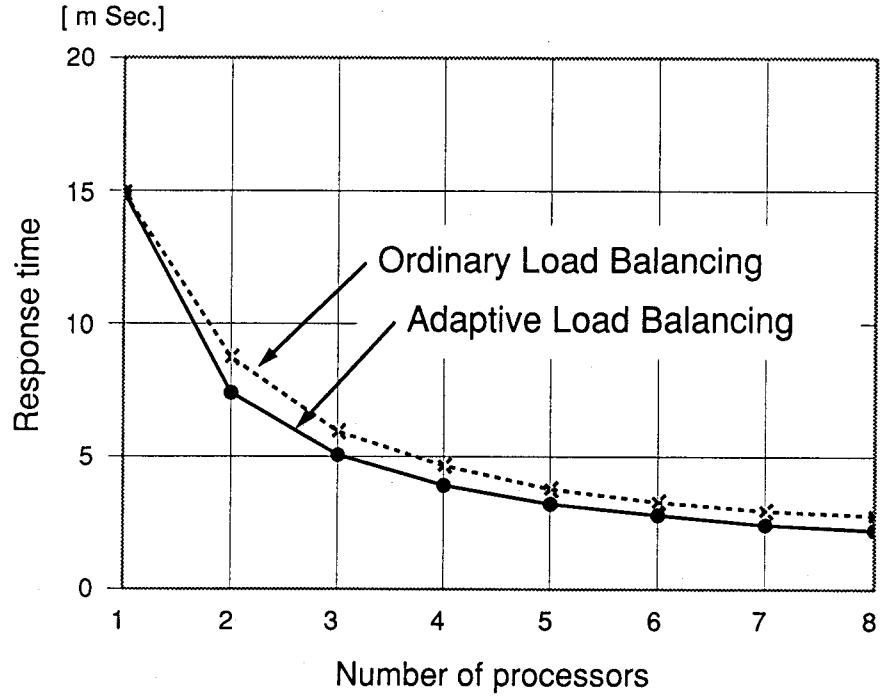


(a) 応答時間

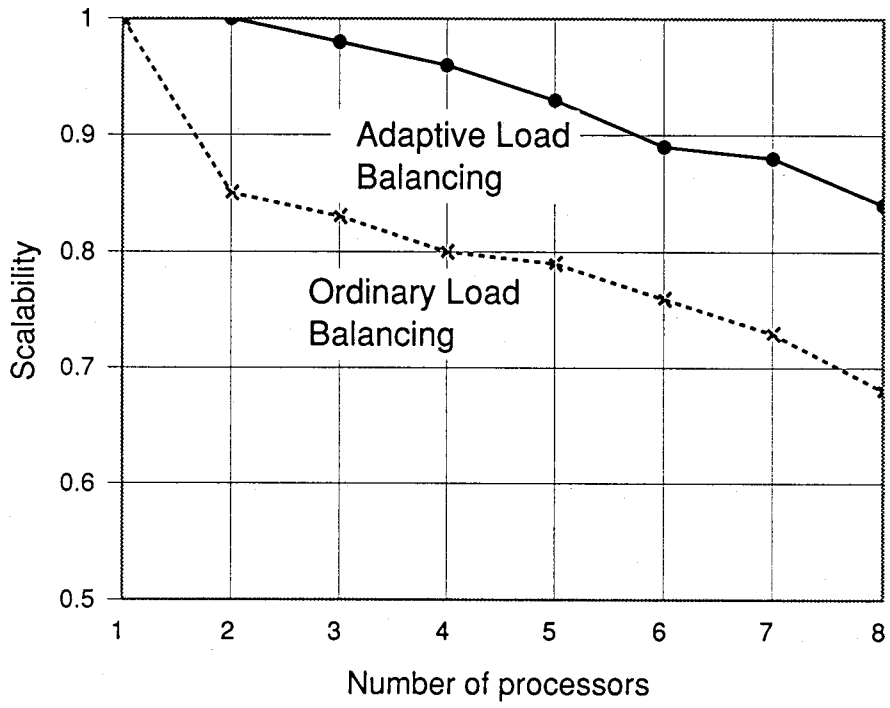


(b) スケーラビリティ

図 6.6: 単純問合せの応答時間とスケーラビリティ

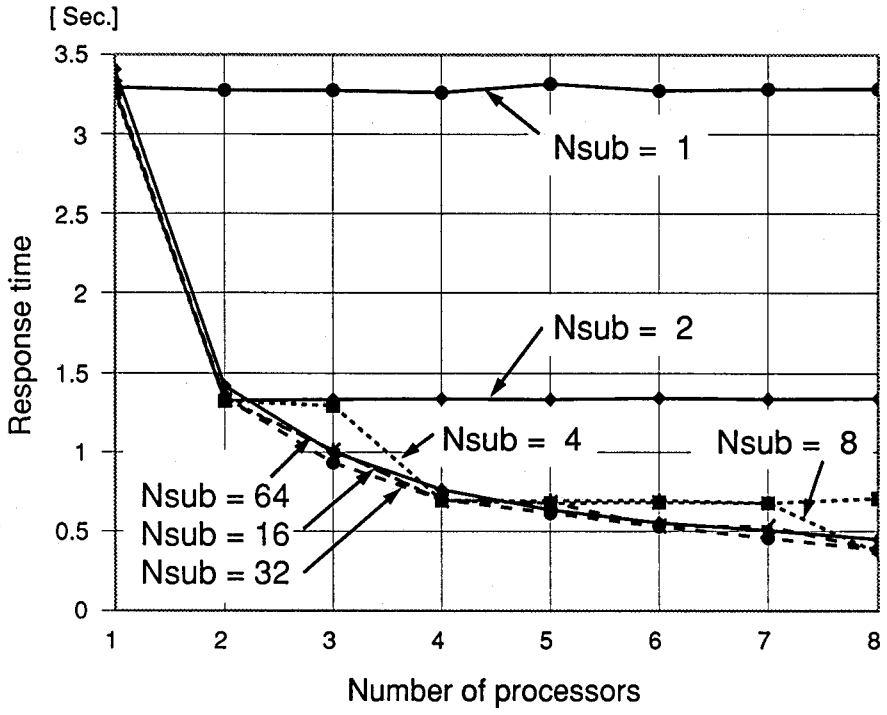


(a) 応答時間

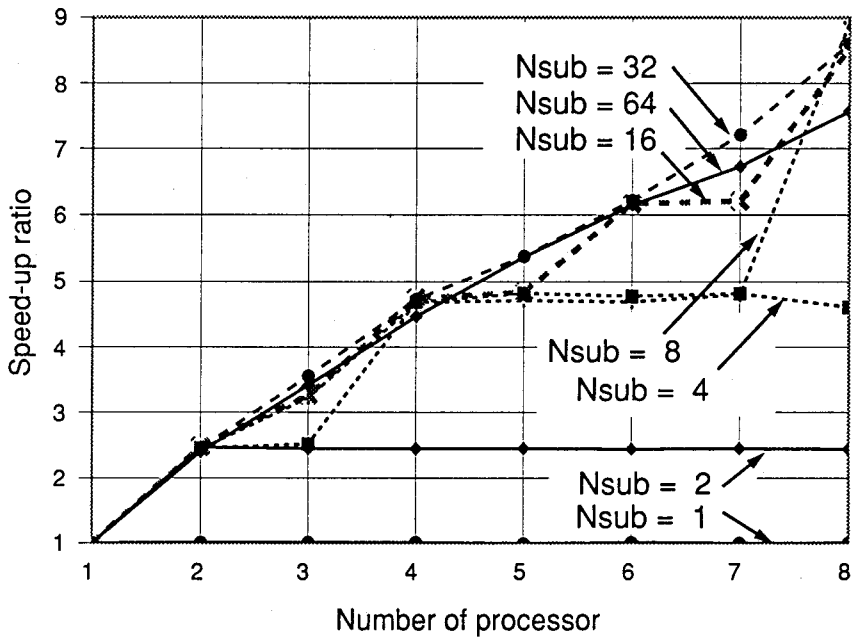


(b) スケーラビリティ

図 6.7: 結合を含む問合せの応答時間とスケーラビリティ

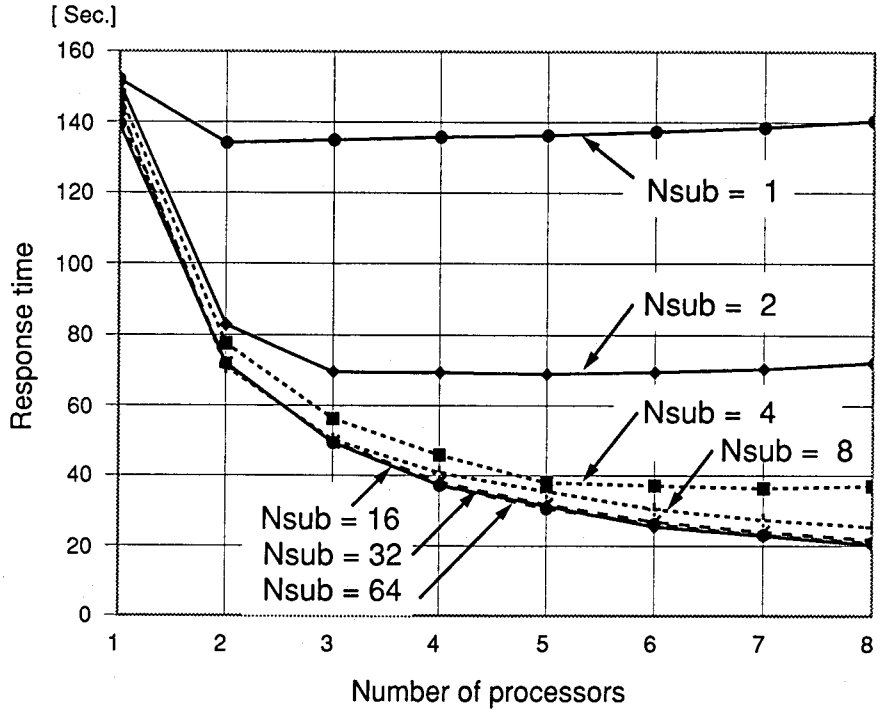


(a) 応答時間

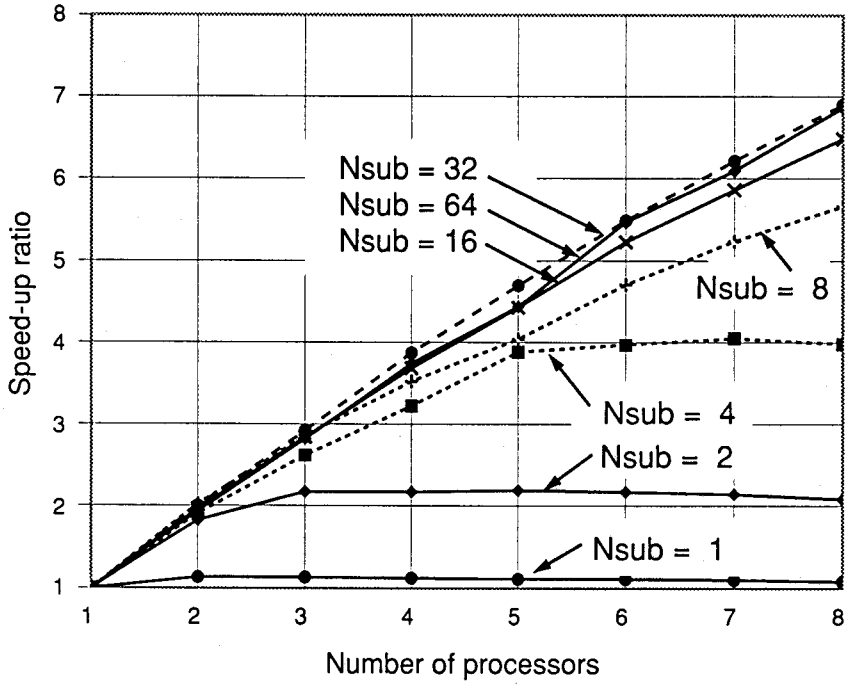


(b) スピードアップ率

図 6.8: インデックスによる範囲検索の応答時間とスピードアップ率



(a) 応答時間



(b) スピードアップ率

図 6.9: インデックス更新を伴う表の統合の応答時間とスピードアップ率

以上の結果から，ハイブリッド・インデックスは，インデックスを参照・更新のいずれに対しても，並列処理のスケーラビリティを向上するうえで有効であることが確認できた．ハイブリッド・インデックスの重要なパラメータであるサブ木の数は，並列処理する最大プロセッサ数と同等かわずかに多い程度が，インデックスの参照および更新のいずれの処理に対しても高いスケーラビリティを達成できる．

6.6 まとめ

本章では，関係データベース処理の高速化を実現する第 2 のアプローチである粗粒度並列処理を述べた．

共有メモリ型マルチプロセッサにおける並列処理オーバヘッドを削減できる適応型動的負荷配分法を提案した．1 回に割り当てるタスクのサイズは，残りタスク数とプロセッサ数，タスクあたりの最大 / 最小処理時間から計算する．処理の開始時点では負荷配分単位が大きく処理進行に伴って小さくなるように，並列処理の単位 (粒度) を可変とすることで，大量の負荷 (ページ) を少ない配分回数で，各プロセッサの処理量が均等になるように配分することができた．

インデックスの参照 / 更新を並列処理するのに適したハイブリッド・インデックス構成法を提案した．従来の B-tree を複数のサブ木としてそれらをハッシュディレクトリによって統合した構造を持ち，サブ木に対する操作を他のサブ木と独立に実行できる．サブ木に対する検索あるいは更新操作が並列処理の単位である．

上記，提案した負荷配分方法とインデックス構成法の有効性を検証するために，メモリ共有型マルチプロセッサ上にプロトタイプ DBMS を作成した．長大な応答時間を要する 4 種類のトランザクションについて応答時間を測定し，適応型動的負荷配分法は従来法と比較して 10% から 15% 程度スケーラビリティを向上できることを明らかにした．また，インデックスを用いた範囲検索と行の挿入では，プロセッサ数と同数以上のサブ木からなるハイブリッド・インデックスを用いることで，プロセッサ数にほぼ比例した性能向上が達成できることを明らかにした．

第 7 章

結論

7.1 本研究のまとめ

本論文では，大規模な関係データベース処理の高速化を狙いとする並列処理法に関する研究成果を述べた．

この研究の背景には，関係データベースの言語 SQL が標準化され，その本来の使い勝手と柔軟性の高さを発揮できる業務への適用が広がるにつれて，大規模なデータベースに対する非定型の問合せ処理の高速化が必要とされてきたことがある．特に，経営戦略を作成支援する応用情報システムでは，表から統計情報を作成したり表間の相関を取る処理に長大な処理時間を要するため，処理の高速化が大きな技術的課題となっていた．この課題を解決するために，これまで多くの方式が研究・開発されてきていたが，大規模データベースに十分適用できるものはほとんどなかった．

本研究では，関係データベースの問合せ応答時間を長大化させる要因が，データベースの表を条件検索するサーチ，問合せ結果の並べ換え等で多用されるソート，複数の表を関連づけるジョインであることを示し，これらの基本演算を並列処理によって高速化する手法を提案した．一般に並列度を上げるには，与えられた問題 (問合せ) を細分化して同時に実行できるタスク数を増やす必要がある．しかし，問合せの内容によっては問題分割が困難な場合もある．簡単に問題が分割できる場合でも，タスク数を増やすと，タスク間の同期や通信のための実行オーバヘッドが増大することが避けられない．そこで，本研究では，粒度の小さい細粒度並列は専用ハードウェアで実現することで実行オーバヘッドを削減することを目指した．問合せ条件やデータ分布

の偏りによって均一な問題分割が困難な場合は，負荷分散をソフトウェア的に制御できるマルチプロセッサシステムを用いて粗粒度並列を実現し，分割オーバヘッドを増やすことなく高速化することを目指した．

本論文では，最初に，典型的な細粒度並列であるソートの並列処理法を提案した．これまで報告されているハードウェアソータが高速化に重点を置いたアーキテクチャであるのに対して，提案するマルチウェイマージソータは，データベース処理で重要なレコード長とレコード数に対する高い拡張性を有しているところに特徴がある．比較ユニットを1次元アレー配置したソートアレーでは，レコードの入出力と同期してシストリックに並列比較を行なう．ソートアレーの並列比較機能を用いてマージウェイ数を拡大しても速度が低下しないマルチウェイマージ回路を構成し，逐次的なマージ操作の繰り返しでワークメモリに格納された任意個数のレコードを高速にソートする．また，比較ユニット間の接続を可変として，レコード長に応じた個数の比較ユニットを連動させることで，レコード長が短い場合でも長い場合でもハードウェアを有効に利用できる．

次に，上記のハードウェアソータにハッシュ化ビットアレーを用いたふるい落とし手法を組み合わた3フェーズジョイン法を提案した．この方法は，扱うデータの属性や長さ，個数が広範に変化するデータベース処理の形態を考慮して，ハードウェアとソフトウェアを協調させ柔軟性を犠牲にすることなく高速化するところに特徴がある．ハードウェア化の容易性とシステム構成の柔軟性から，フィルタフェーズとソートフェーズを専用ハードウェアで実現し，問合せ条件に依存する行の連結操作が必要になるマージ結合フェーズはソフトウェアで処理することにした．扱う行数が多く演算量も多いフィルタとソートを専用ハードウェアで実現することによって，フィルタフェーズとソートフェーズのレコード入力，ソートフェーズのレコード出力とマージ結合フェーズの処理を行単位でパイプライン化し，高速なジョインを実現する．

これらソートとジョインを高速化する専用ハードウェアと，サーチを高速化するハードウェアとでデータベースプロセッサRINDAを構成し，非定型な問合せの応答時間を測定・評価し，広範な問合せ処理に対して顕著な性能向上を達成できることを明らかにした．

一方，粗粒度並列は，資源共有型マルチプロセッサシステム上に問合せ処理を実装し，応答時間の測定結果から，関係データベースの並列処理を高いスケーラビリティで実現できることを確認した．そこでは，プロセッサに配分する負荷のサイズを処理の進行に伴って変更する適応型動的負荷配分法を提案し，スケーラビリティ向上に有効であることを明らかにした．また，複数プロセッサから同時にアクセスされるインデックスをハッシュと木構造を組み合わせたハイブリッドな構成とすることで，アクセス競合を大幅に低減できることを示した．

本論文では，以上の研究成果を並列処理の観点から以下の5章に分けて述べた．

第1章では，研究の背景と問題設定，従来の研究，本研究の方針について述べた．問題設定の中では，研究の背景で抽出したサーチ，ソート，ジョインの処理概要を述べ，処理時間の長大化要因を分析した．本研究では，関係データベース処理が行単位で独立に処理できる場合が多いことに着目し，行単位のデータ並列およびパイプライン処理 (細粒度並列) によって高速化することを検討した．更に，データベースがページと呼ばれる固定サイズのブロックを単位として格納されていることに着目し，1ページあるいは複数ページを単位とした並列処理 (粗粒度並列) を検討した．

第2章では，細粒度並列処理の典型例である並列比較回路 (ソートアレー) の柔軟な構成法，および，階層化冗長構成を適用した大規模一括集積の設計法を述べた．1次元アレー配置した比較ユニットからなるソートアレーでは，レコードの入出力と同期してシストリックに並列比較を行なう．比較ユニット間の接続を可変として複数ユニットを連動させることで，ソートできる最大レコード長の拡張とソート速度の向上を実現した．規則的な繰り返し構造からなるソートアレーは，回路の階層構造に着目して階層的に予備を設ける階層化冗長構成法を適用することで，複数ユニットを大規模一括集積した巨大チップを実現できる．予備を増やして冗長度を上げるとWSIの良品率を向上できることから，良品率と冗長度から算出したウェハの有効利用率を用いてWSI規模を最適化できることを明らかにした．3階層の冗長構成を適用した冗長度0.88のソートアレーチップは，基本ブロックとして40個の比較ユニットを良好な良品率で大規模一括集積できた．

第 3 章は，第 2 章に示したソートアレーを基本構成要素とし，ソートできる最大レコード数とソート処理速度に柔軟に対応できるマルチウェイマージソータの構成法を提案した．本方法を適用したソータは，レコードを格納するワークメモリ，レコードを比較 (並列比較) するソートアレー，逐次的なマージ操作を制御するマージ制御回路で構成される．データ駆動形マージ制御と並列比較を行なうソートアレーを採用し，ウェイ数を拡大してもマージ速度が低下しないマルチウェイマージ回路を構成する．本マージ回路を用いて逐次的なマージ操作を繰り返し，ワークメモリに格納された任意個数のレコードをソートする．上記ソータのワークメモリ管理法として，メモリ利用率を十分高くできるエリア格納法を提案した．従来のメモリ管理法では，マージウェイ数を大きくするとメモリ利用率が低下するのに対し，本方法ではマージウェイ数を大きくするに伴ってメモリ利用率を向上できることを明らかにした．提案したマルチウェイマージソータは，レコードの比較回路 (ソートアレー) とレコードの格納回路 (ワークメモリ) とが分離されているため，それぞれを高密度に実装できる．また，マージ段数に依存するソート処理速度と，ワークメモリの容量で決まるソート可能な最大レコード数を独立に設定できるため，利用者の要求条件に応じて最適な構成のソータを柔軟に実現できる．

第 4 章では，第3章に示したマルチウェイマージソータを用い，さらに，ソートの前処理としてハッシュ化ビットアレーを用いたふるい落とし手法を適用した 3 フェーズジョイン法を提案した．本方法は，結合する表から結合可能性のない行をふるい落とすフィルタフェーズ，残った行を並べ替えるソートフェーズ，ソートされた行をマージしながら連結するマージ結合フェーズからなる．本結合法は，従来のソートマージ結合法の前処理としてふるい落とし処理を効果的に組み合わせた手法であり，ふるい落としの対象となる表が，1 個 (片方) か 2 個 (両方) かによって片ハッシュ結合法と両ハッシュ結合法の 2 種類が実現できることを示した．ハードウェア化の容易性とシステム構成の柔軟性から，フィルタフェーズとソートフェーズを専用ハードウェア化する．フィルタフェーズ，ソートフェーズのレコード入力，ソートフェーズのレコード出力とマージ結合フェーズの処理を行単位でパイプライン化して重畳することができ，専用ハードウェア化することで極めて高速なジョインを実現できた．ハッシュ化ビッ

トアレーを用いるフィルタフェーズに適用するハッシュ関数として，キーの長さ，属性が変わっても衝突が起こりにくい乗算重ね合わせ法を提案した．固定長化したキー切片に定数を掛け合せる乗算は精度の要求が小さいことから，乗算結果をメモリに展開した乗算表を用いることとし，乗算結果の重ね合わせはシフト回路と排他的論理和回路で実現することとした．このように乗算重ね合わせ法を用いたハッシュ回路は簡単な回路で実現でき，キー切片を単位とするパイプライン処理によってキーの入力速度に完全に追従した動作を実現できた．

第5章では，これまで述べてきた細粒度並列によるソートやジョインの高速化手法を有機的に統合したデータベースプロセッサRINDAのアーキテクチャと性能評価結果を述べた． さらに，第1章で提起したサーチの高速化手法についても述べた．RINDAは，ホスト計算機上のデータベース管理システムの配下で動作する専用ハードウェアであり，比較的粒度の小さい細粒度並列処理を中心に，さまざまな粒度での並列処理を実現している．関係演算高速化プロセッサROPは，ソート，ジョイン，グループ化を高速化する専用ハードウェアであり，一次元アレー構造のソートアレーにおける並列比較やジョインにおける行単位のパイプライン処理等の細粒度並列処理を専用ハードウェアで実現している． ROPではソートやジョインで使用する内部キーを行から抽出するキー抽出処理も専用ハードウェア化しているため，キー抽出→ふるい落とし→ソート，および，ソート→出力編集を各々行単位でパイプライン処理している．乗算重ね合わせ法を適用したふるい落とし回路の内部は，行よりも更に細かい単位でパイプライン処理しており，行を単位とするROPの入出力処理に完全に追従した処理速度を実現できた．一方，内容検索プロセッサCSPは，IO負荷の大きいサーチをディスク制御装置の位置で直接実行する専用ハードウェアである．ハードウェア量を削減して実現の容易性を高めるために，DBMSは与えられた検索条件の論理式を乗法標準形に変換し，CSPが実行可能な部分はCSPが高速に処理し，CSPが実行できない部分はDBMSが引き継いで処理する方式とした．上記論理式の変換過程において，DBMSは論理式をNOTを含まない乗法標準形に変換する．これにより，SQL仕様に基づく3値論理の検索条件判定は等価な2値論理で実行でき，CSPの論理式判定の回路規模を削減できた． CSPはシリンダ単位のマルチトラックリードで

ページの連続読み出しを行い，ページ単位でのデータ読み出しと内容検索をパイプライン処理する．さらに，一つの表を複数の磁気ディスク装置に水平分割して格納し，複数台の CSP で並列サーチすることで更に高速化することができた．

第 6 章では，関係データベース処理の高速化を実現する第 2 のアプローチである粗粒度並列処理を述べた．資源共有型マルチプロセッサにおける並列処理オーバヘッドを削減する適応型動的負荷配分法を提案した．1 回に割り当てるタスクのサイズは，残りタスク数とプロセッサ数，タスクあたりの最大 / 最小処理時間から計算する．処理の開始時点では負荷配分単位が大きく処理進行に伴って小さくなるように，並列処理の単位 (粒度) を可変とすることで，大量の負荷 (ページ) を少ない配分回数で，各プロセッサの処理量が均等になるように配分できることを明らかにした．次いで，インデックスの参照 / 更新を並列処理するのに適したハイブリッド・インデックス構成法を提案した．従来の B-tree を複数のサブ木としてそれらをハッシュディレクトリによって統合した構造を持ち，サブ木に対する操作を他のサブ木と独立に実行できる．サブ木に対する検索あるいは更新操作が並列処理の単位である．上記，提案した負荷配分方法とインデックス構成法の有効性を検証するために，資源共有型マルチプロセッサ上にプロトタイプ DBMS を作成した．長大な応答時間を要する典型的な 4 種類のトランザクションについて応答時間を評価し，適応型動的負荷配分法は従来法と比較して 10% から 15% 程度スケーラビリティを向上できることを明らかにした．また，インデックスを用いた範囲検索と行の挿入では，プロセッサ数と同数以上のサブ木からなるハイブリッド・インデックスを用いることで，プロセッサ数にほぼ比例した性能向上が達成できることを明らかにした．

7.2 今後の研究課題

本論文の第 2 章から第 5 章で述べた細粒度並列を中心とする処理方法は，データベースプロセッサ RINDA として既に数十の実用システムで稼働している．典型的な利用形態は，複数の利用者がデータを追加・更新しているデータベースに対して，オンラインで統計情報を求める業務である．従来，これらの業務は，統計処理に多大な処理時間を要するので，複数利用者がオンラインで使用している現用システムからデー

タベースを他のマシンに移行して，統計処理をバッチで実行していた．このため，データベースの最新状態で統計をとることができなかった．RINDA を用いることで，これまで求めることが不可能であった最新の統計情報を得ることができるようになった．

現在は，求めた統計情報に多少の誤差が含まれることが許容されているため，データベースの追加・更新処理と検索処理は並行処理制御なしで実行させているが，より精度の高い新鮮な統計情報を得るために，更新処理中でも検索処理が行なえる多版実行制御を実現することが今後の課題である．

データベース言語 SQL におけるフェッチ操作は並列処理との親和性が低い．このため，検索結果の行数が増大するに伴って専用ハードウェアで達成した高速化効果が低減している．この問題を解決するためには，アプリケーションプログラマには現状の逐次的なモデルを提供したままで，実装上は検索結果の一括返却を実現できる言語面からのアプローチも今後重要になると考えられる．

近年，マイクロプロセッサの性能向上と低価格化が加速度的に進んでおり，本論文の後半で述べたマルチプロセッサシステムを前提とした粗粒度並列がデータベース処理を高速化する主流になると思われる．本論文で述べた粗粒度並列は，単に並列処理する単位が大きい (粗い) ことを意味するのではなく，様々な粒度での並列処理を同時に実現することであり，適応型動的負荷配分法で述べたように処理の過程で負荷分散の粒度を調整することも含んでいる．従って，将来的には，本論文の前半で述べた細粒度並列と，粗粒度並列を融合した処理形態，すなわち，専用ハードウェアを備えたプロセッサでノードを構成し，複数個のノードをネットワークで結合した処理形態が有望である．この場合，プロセッサ台数やデータベースの規模に比例したスケーラブルなシステム構成とすることが重要な課題となる．また，データベース自身も複数のノードに分散格納されると考えられるが，従来の分散データベースの考え方に基づいてデータベースの一貫性を強く保証するのでは，ノード数が増えた場合に一貫性を保証するために処理が待たされる，あるいは，すでに実行した処理を無効にする等の問題が生じ，スケーラビリティを低下させる大きな要因となってくると考えられる．このため，分散配置されたデータベースを，必要かつ最小限の一貫性で管理し，全体を有機的に連合させることも重要な課題である．

謝辞

本研究をまとめるにあたり，懇切なるご指導と格別のご配慮を賜った大阪大学工学部情報システム工学科 西尾 章治郎教授に心から感謝申し上げます．

また，本研究をまとめるにあたり，貴重な時間を度々割いていただき，懇切なるご指導とご助言を賜った大阪大学工学部 鷹田 憲久 教授，鈴木 胖 教授，寺田 浩詔 教授，ならびに，白川 功 教授に深く感謝申し上げます．

本研究を遂行するにあたり，ソータの構成法に関して熱心な討論とご助言を頂いた東京大学 喜連川 優 助教授に深く感謝いたします．粗粒度並列全般，特に動的負荷配分法に関して熱心に討論頂いた筑波大学 清木 康 助教授に深く感謝いたします．

本研究は，筆者が日本電信電話公社 (NTT) 電気通信研究所において行なった研究成果をまとめたものであり，本研究をまとめる動機づけとその機会を与えて頂き，さらに常に励まして頂いたNTT 情報通信網研究所 石野福彌 前所長，伊土誠一 研究企画部長，石垣昭一郎 データベース研究部長，寺中勝美 プロジェクトリーダに厚くお礼申し上げます．

本研究は長期にわたり遂行してきた成果をまとめたものであり．本研究を開始するきっかけを与えて下さり，論文の共著者でもある境界領域研究所 川田忠通 主席研究員，情報通信網研究所 津田伸生 主幹研究員に深く感謝いたします．本研究の成果をデータベースプロセッサ RINDA として実現する機会を与えて下さった歴代の研究部長 松永俊雄 博士，拝原正人 氏，松田晃一 氏，著者の元上司である 篠岡信 氏，鈴木健司 氏，福岡秀樹 氏に心からお礼申し上げます．著者の現在の上司であり論文の共著者でもある 井上 潮 博士には本論文作成のための貴重なご指導を頂きました．

著者の論文の共著者である 武田英昭 氏，速水治夫 氏，中村敏夫 氏，黒岩淳一 氏，平野泰宏 氏，芳西 崇 氏には多大なるご協力とご援助を頂きました．協力メーカの源代裕治 氏，清水尚彦 氏には関係演算高速化プロセッサの実現に関し深夜まで議論して頂いた．個々の御名前は省かせて頂くが，NTT 情報通信網研究所の研究員の方々にも数多くの討論を頂いた．深く感謝いたします．

最後に，家族の励ましと協力に感謝します．

参考文献

[1] Akl, S. G. : "Parallel Sorting Algorithm", Academic Press, Inc. (1985).

[2] 安藤隆朗, 小宮冨士夫, 中込 宏, 伏見信也, 喜連川 優: "リレーショナルデータベースプロセッサ GREO の構成", 電子情報通信学会技術研究報告 (データ工学研究会), DE89-37, pp. 9-15 (1989).

[3] Babb, E. : "Implementing a Relational Database by Means of Specialized Hardware", ACM Trans. on Database Syst., Vol. 4, No. 1, pp. 1-29 (March 1979).

[4] Bitton, D., DeWitt, D. J., and Turbyfill, C. : "Benchmarking Database Systems - A Systematic Approach", CTSR #526, Univ. of Wisconsin-Madison (1983).

[5] Blasgen, M. W., and Eswaran, K. P. : "Storage and Access in Relational Data Bases", IBM Syst. J., No. 4, pp. 363-377 (1977).

[6] Boral, H., and Redfild, S. : "Database Machine Morphology", Proc. 11th Int. Conf. Vary Large Data Bases, pp. 59-71 (1985).

[7] Boral, H., Alexander, W., Cray, L., Copeland, G., Danforth, S., Franklin, M., Hart, B., Smith, M., and Valduriez, P. : "Prototyping Bubba, A Highly Parallel Database System", IEEE Trans. Knowledge and Data Engeneering, Vol. 2, No. 1, pp. 4-24 (Mar. 1990).

[8] Britton Lee, Inc. : "Server/8000 for use with ShareBase II Software" (1988).

[9] Bülzingsloewen, G. V., Iochpe, C., Liedtke, R.-P., Kramer, R., Schryro, M., Dittrich, K. R., and Lockemann, P. C. : "Design and Implementation of KAR-

DAMON – A Set-oriented Data Flow Database Machine", Lecture Notes in Computer Science Vol. 368, Proc. 6th Int. Workshop, IWDM'89, Boral, H and Faudemay, P. eds., pp. 18–33, Springer-Verlag (Jun. 1989).

[10] Chen, T. C., Lum, V. Y., and Tung, C. : "The Rebound Sorter: An Efficient Sort Engine for Large Files", Proc. 4th Int. Conf. Very Large Data Bases, pp. 312–318 (1978).

[11] Codd, E. F. : "A Relational Model of Data for Large Shared Data Banks", Communications of the ACM, Vol. 13, No. 6, pp. 377–387 (June 1970).

[12] DeWitt, D. J. : "DIRECT – A Multiprocessor Organization for Supporting Relational Database Management", IEEE Trans. Comput., Vol. C–28, No. 6, pp. 395–406 (June 1979).

[13] DeWitt, D. J., Gerber, R. H., Graefe, G., Heytens, M. L., Kumar, K. B., and Muralik-rishna, M. : "GAMMA – A High Performance Dataflow Database Machine", Proc. 12th Int. Conf. Very Large Data Bases, pp.228–237 (1986).

[14] DeWitt, D. J., Ghandeharizadeh, S., Schneider, D., Jauhari, R., Muralikrishna, M., and Sharma, A. : "A Single User Evaluation of the GAMMA Database Machine", Database Machines and Knowledge Base Machines, Kitsuregawa, M. and Tanaka, H. eds., pp.370–386, Kluwer Academic (1988).

[15] DeWitt, D. J., Ghandeharizadeh, S., Schneider, D. A., Bricker, A., Hsiao, H.-I., and Rasmussen, R. : "The Gamma Database Machine Project", IEEE Trans. Knowledge and Data Engineering, Vol. 2, No. 1, pp. 44–62 (Mar. 1990).

[16] DeWitt, D. J. and Gray, J. : "Parallel Database Systems: The Future of Database Processing or a Passing Fad?", ACM SIGMOD RECORD, Vol. 19, No. 4, pp. 104–112 (Dec. 1990).

[17] 土肥康孝：“大容量ファイルを整列するシストリック・ソータ”, 電子情報通信学会 論文誌, Vol. J67–D, No. 3, pp. 281–288 (Mar. 1984).

[18] Englert, S., Gray, J., Kocher, T., and Shah, P. : “A Benchmark of NonStop SQL Release 2, Demonstrating Near-Linear Speedup and Scaleup on Large Databases”, Tandem Technical Report 89.4, Part No. 27469 (May 1989).

[19] 伏見信也, 科野順蔵, 鈴木 孝, 笠原康則, 太刀掛伸一, 鍋田芳則, 木村廣隆, 沢井善彦, 喜連川 優, 楊 維康：“LSI ソートプロセッサ”, 電子情報通信学会技術研究報告 (データ工学研究会), DE88–2, pp. 9–17 (1988).

[20] 速水治夫：“SQL 探索条件判定における NULL 処理の簡単化”, 情報処理学会 論文誌, Vol. 32, No. 1, pp. 71–76 (1991).

[21] 速水治夫, 佐藤哲司, 中村敏夫, 黒岩淳一, 武田英昭：“リレーショナルデータベースマシンにおけるサーチ処理方式”, 電子情報通信学会 論文誌, Vol. J75–D–I, No. 4, pp. 232–240 (Apr. 1992).

[22] 平野泰宏, 佐藤哲司, 井上 潮, 寺中勝美：“資源共有型マルチプロセッサにおけるデータベース処理の動的負荷配分法”, 電子情報通信学会 論文誌, Vol. J75–D–I, No. 3, pp. 152–159 (Mar. 1992).

[23] 芳西 崇, 板倉一朗, 中村敏夫, 井上 潮：“データベースプロセッサ RINDA における問合せ処理のアクセスパス決定方式”, 情報処理学会 論文誌, Vol. 32, No. 11, pp. 1412–1422 (1991).

[24] Honishi,T., Satoh, T., and Inoue, U. : “An Index Structure for Parallel Database Processing”, Proc. Int. Workshop Research Issues on Data Engineering: Transaction and Query Processing, pp. 224–225 (Feb. 1992).

[25] 井上 潮, 北村 正, 速水治夫, 中村敏夫：“情報提供サービスに適用可能な超大規模リレーショナルデータベースマシン”, 情報処理学会研究会報告, Vol. 85, No. 6, 85–DB–47–5 (1985).

[26] Inoue, U., Satoh, T., Hayami, H., Takeda, H., Nakamura, T., and Fukuoka, H. : "Rinda: A Relational Database Processor with Hardware Specialized for Searching and Sorting", IEEE Micro, Vol. 11, No. 6, pp. 61–70 (Dec. 1991).

[27] 井上 潮, 佐藤哲司, 速水治夫 : "データベースプロセッサ RINDA", 情報処理, Vol. 33, No. 12, pp. 1403–1408 (Dec. 1992).

[28] ISO : "Information Processing System – Database Language SQL", International Standared ISO 9075 (June 1987).

[29] 伊藤文英, 島川和典 他 : "可変長レコード用関係データベース処理エンジンの試作とソート処理性能の評価", 情報処理学会 論文誌, Vol. 30, No. 8, pp. 1033–1045 (1989).

[30] 岩田和秀, 神谷茂雄, 酒井 浩, 柴山茂樹, 伊藤英則, 村上国男 : "関係データベース処理エンジンのソータの試作と評価", 情報処理学会 論文誌, Vol. 28, No. 7, pp. 748–757 (1987).

[31] Jonge, W. and Schijf, A. : "Concurrent Access to B-trees", Proc. Int. Conf. Databases, Parallel Architectures, and Their Applications (PARBASE-90), Rishe, N., Navathe, S., and Tal, D. eds., pp. 312–320 (March 1990).

[32] 上林 弥彦 : "データベース研究 —21 世紀に向けての挑戦 —", 情報処理学会 データベース・システム研報 (DBS84–5), pp. 39–46 (1991).

[33] Kim, P.-C., Choi, H.-I., and Lee, Y.-J. : "Design and Implementation of the Multiuse Index-based Data Access System", Proc. 2nd Int. Symp. Database Systems for Advances Applications, pp. 156–164 (April 1991).

[34] Kitsuregawa, M., Tanaka, H., and Moto-oka, T. : "Application of Hash to Data Base Machine and Its Architecture", New Generation Computing, Vol. 1, pp. 63–74, Springer-Verlag (1983).

[35] 喜連川 優, 伏見信也, 桑原和宏, 田中英彦, 元岡 達：“パイプラインマージソータの構成”, 電子情報通信学会 論文誌, Vol. J66–D, No. 3, pp.332–339 (Mar. 1983).

[36] Kitsuregawa, M., Fushimi, S., Tanaka, H., and Moto-oka, T. : “Memory Management Algorithms in Pipeline Merge Sorter”, Proc. 4th Int. Workshop Database Machine, pp. 208–232 (1985).

[37] 喜連川 優, 伏見信也：“データベースマシン”, 情報処理, Vol. 28, No. 1, pp. 56–67 (1987).

[38] 喜連川 優, 楊 維康, 鈴木慎司：“VLSI ソートプロセッサ”, 情報処理, Vol. 31, No. 4, pp. 457–465 (1990).

[39] 喜連川 優：“最近のデータベースプロセッサの商用化ならびに研究開発の動向”, 情報処理, Vol. 33, No. 12, pp. 1388–1402 (1992).

[40] 清木 康：“データベースマシンの動向”, アドバンスト・データベース・システムシンポジューム論文集, pp.31–40 (Dec. 1987).

[41] Kiyoki, Y., Kurosawa, T., Kato, K., and Masuda, T. : “The Software Architecture of a Parallel Processing System for Advanced Database Applications”, Proc. 7th Int. Conf. Data Engineering, pp. 220–229 (Apr. 1991).

[42] Knott, G. D. : “Hashing functions”, Computer J., Vol. 18, No.3, pp.265–287 (1975).

[43] 小島啓二, 鳥居俊一, 吉住誠一：“ベクトル型データベースプロセッサ IDP”, 情報処理学会 論文誌, Vol. 31, No. 1, pp. 163–173 (1990).

[44] Kumar, M. and Hirschberg, D. S. : “An Efficient Implementation of Batcher's Odd-Even Merge Algorithm and Its Application in Parallel Sorting Schemes”, IEEE Trans. Comput., Vol. C–32, No. 3, pp. 254–264 (1983).

[45] Kung, H. T. : "Why Systoric Architectures?", IEEE Comput., Vol. 15, No. 1, pp. 37–46 (1982).

[46] Lee, D. T., Chang, H., and Wong, K. : "An Onchip Compare/Steer Bubble Sorter", IEEE Trans. Comput., Vol. C–30, No. 6, pp. 396–404 (June 1981).

[47] Lehman, P. and Yao, S. : "Efficient Locking for Concurrent Operations on B-Trees", ACM Trans. Database Syst., Vol. 6, pp. 650–670 (1981).

[48] Lum, V. Y., Yuen, P. S. T., and Dodd, M. : "Key-to-Address Transform Techniques: A Fundamental Performance Study on Large Existing Formatted Files", Communications of the ACM, Vol. 14, No. 4, pp. 228–239 (1971).

[49] 松田 進, 東郷一生, 島川和典, 岩崎孝夫 : " データベース演算処理装置のアーキテクチャ ", 電子情報通信学会技術研究報告 (データ工学研究会), DE 91–259, pp. 33–38 (1991).

[50] McGregor, D. R., Thomson, R. G., and Dawson, W. N. : "High Performance Hardware for Database Systems", Systems for Large Data Bases, pp. 103–116, North-Holland (1976).

[51] Miller, L. L. and Hurson, A. R. : "MAYBE ALGEBRA Operations IN DATABASE MACHINE ARCHITECTURE", Fall Joint Computer Conf., pp. 1210–1218 (1986).

[52] Miranker, G., Tang, L., and Wong, C. K. : "A 'Zero-Time' VLSI Sorter", IBM J. Res. Dev., Vol. 27, No. 2, pp. 140–148 (1983).

[53] Murphy, M. C. and Shan, M.-C. : "Execution Plan Balancing", Proc. 7th Int. Conf. Data Engineering, pp. 698–706 (Apr. 1991).

[54] Neches, P. M. : "The Ynet: An Interconnect Structure for a Highly Concurrent Data Base Computer System", Proc. 2nd Symp. Frontiers of Massively Parallel Computation, pp. 429–435 (1988).

[55] 日本工業規格：“データベース言語 SQL”, JIS X 3005 (Nov. 1987).

[56] 小柳津育郎, 塩川鎮雄, 木ノ内康夫, 安保 進：“DIPS-11/5E シリーズの実用化”, NTT 研究実用化報告, Vol. 36, No. 1, pp. 49–56 (1986).

[57] Ozkarahan, E. A., Schuster, S. A., and Sevcik, K. C. : “Performance Evaluation of a Relational Associative Processor”, ACM Trans. Database Syst., Vol. 2, No. 2, pp. 175–195 (1977).

[58] Ozkarahan, E.A. and Penaloza, M. A. : “On-the-Fly and Background Data Filtering System for Database Architectures”, New Generation Conputing, Vol. 5, pp. 281–314, OHMSHA and Springer-Verlag (1987).

[59] Sagiv, Y. : “Concurrent Operations on B-trees with Overtaking”, Proc. 4th ACM SIGACT/SIGMOD Symp. Principals of Database Systems, pp. 28–37 (1985).

[60] 佐藤哲司, 津田伸生：“階層化冗長構成を用いたソート回路の欠陥検出切替法”, 昭和 59 年度電子通信学会総合全国大会講演論文集, 398 (1984).

[61] 佐藤哲司, 津田伸生：“階層化冗長構成による 1 次元アレイ論理ＬＳＩの欠陥救済”, 電子情報通信学会 (フォールトトレラントシステム研究会), FTS87–4, pp. 27–33 (May 1987).

[62] 佐藤哲司, 武田英昭, 津田伸生：“大容量データベース処理に適した ソータ構成法”, 情報処理学会 論文誌, Vol. 31, No. 11, pp. 1653–1660 (Nov. 1990).

[63] 佐藤哲司, 武田英昭, 井上 潮, 福岡秀樹：“データベースプロセッサ RINDA の結合演算処理機構の構成と評価”, 情報処理学会 論文誌, Vol. 32, No. 8, pp. 1006–1013 (Aug. 1991).

[64] Satoh, T., Hirano, Y., Honishi, T., and Inoue, U. : “Design and Implementation of Parallel Database Processing on a Shared Memory Multiprocessor System”,

Proc. 2nd Far-East Workshop on Future Database Systems, pp. 337–346 (Apr. 1992).

[65] Satoh, T., Takeda, H., and Tsuda, N. : "A Multiway Merge Sorter for Sorting of Large Databases", Journal of Information Processing, Vol. 15, No. 3, pp. 434–440 (Mar. 1993).

[66] 佐藤哲司, 井上 潮 : "関係データベースシステムの高度化技術", コンピュートロール, Vol. 43, 特集 / 高度応用のためのデータベース, 上林弥彦 責任編集, pp. 54–61, コロナ社 (July 1993).

[67] Satoh, T. and Inoue, U. : "Rinda: A Relational Database Processor for Large Databases", EMERGING TRENDS IN DATABASE AND KNOWLEDGE-BASE MACHINES: the application of parallel architectures to smart information systems, Abdelguerfi, M. and Lavington, S. H. eds, IEEE-CS Press (to appear).

[68] Sequent Computer Systems, Inc. : "Symmetry Technical Summary", P/N: 100344447 Rev.A (1987).

[69] Schneider, D. A. and DeWitt, D. J. : "A Performance Evaluation of Four Parallel Join Algorithms in a Shared-Nothing Multiprocessor Environment", Proc. ACM SIGMOD Int. Conf., pp. 110–121 (June 1989).

[70] 新城 靖, 清木 康, 劉 澎, 益田隆司 : "データベースおよび知識ベースを対象としたストリーム指向型並列処理系の共有メモリマシン上への実現", アドバンスト・データベース・システムシンポジウム予稿集, pp. 117–126 (Dec. 1988).

[71] 新城 靖, 清木 康 : "データベースの並列処理を支援するオペレーティング・システムの基本機能", 情報処理学会 研究会報告 (データベースシステム研究会), DBS–73–2 (1989).

[72] Simon, E. : "Update to December 1983 'Dewitt' Benchmark", Britton Lee Inc. (1985).

[73] Slotnick, D. L. : "Logic-per-Track Devices", Advances in Computers, Frautz, Alt. Ed., pp. 291–296, Academic Press (1970).

[74] Su, S. Y. W., Nguyen, L. H., Eman, A., and Lipovski, G. J. : "The Architectural Reatures and Implementation Techniques of the Multicell CASSM", IEEE Trans. Comput. Vol. C–28, No. 6, pp. 430–445 (1979).

[75] 高橋恒介 : " 文字列照合処理への応用 ", 情報処理, Vol. 32, No. 12, pp. 1268–1275 (Dec. 1991).

[76] 武田英昭, 佐藤哲司, 中村敏夫, 速水治夫 : " 関係演算高速化プロセッサ ", 情報処理学会 論文誌, Vol. 31, No. 8, pp. 1230–1241 (Aug. 1990).

[77] Tanaka, Y., Nozaka, Y., and Masuyama, A. : "Pipeline Searching and Sorting Modules as Components of a Data Flow Database Computer", Information Processing 80, Lavington, S. H. ed., pp. 427–432, North Holland (1980).

[78] The Tandem Performance Group : "A Benchmark of NonStop SQL on the Debit Credit Transaction", Proc. ACM SIGMOD Int. Conf., pp. 337–341 (1988).

[79] Todd, S. : "Algorithm and Hardware for a Merge Sort Using Multiple Processors", IBM J. Res. Dev., Vol. 22, No. 5, pp. 509–517 (1978).

[80] Tsuda, N., Satoh, T., and Kawada, T. : "A Pipeline Sorting Chip", Proc. 1987 IEEE Int. Solid-State Circuits Conf., pp. 270 (Feb. 1987).

[81] Tsuda, N. and Satoh, T. : "Hierarchical Redundancy for A Linear-Array Sorting Chip", IFIP WG 10.5 2nd Workshop on Wafer Scale Integration, Wafer Scale Integration II, Lea, M. ed., pp. 63–74, North-Holland (1988).

[82] 津田伸生：“アレー構造 WSI の階層化冗長構成法”, 電子情報通信学会 論文誌, Vol. J75–D–I, No. 1, pp.41–52 (Jan. 1992).

[83] 植村俊亮, 弓場敏嗣, 他：“磁気バブルデータベースマシン”, パターン情報処理システム調査・研究報告, PIPS–R–No. 28, 29, 電総研 (Mar. 1981).

[84] 安浦寛人, 高木直史：“並列計数法による高速ソーティング回路”, 電子情報通信学会 論文誌, Vol. J65–D, No. 2, pp. 179–186 (1982).

[85] 矢沢良一, 平野正則, 山口利和, 岡田靖史：“DIPS-V30E のハードウェア構成”, NTT 研究実用化報告, Vol. 37, No. 9, pp. 523–532 (1986).

[86] Wang, Y.-T. and Morris, R. J. T. : “Load Sharing in Distributed Systems”, IEEE Trans. Comput., Vol. C–34, No. 3, pp. 204–217 (Mar. 1985).