計算機アーキテクチャ 35－4
（1979. 9. 12）

リレーショナルデータベース演算のハードウェア化アルゴリズムとそれに基づくデータベースマシンアーキテクチャ

上林憲行 小沢裕之 清木康 加藤洋
瀬尾和男 田中浩一 相磯秀夫（慶應義塾大学工学部）

1. はじめに

最近、IE³ transactions on Computes[4,20]やIE³ society Computer[23~27]でデータベースマシン(DBM)に関する特集号があいついで組まれ、DBMの研究・関心が一時的なブームではなく、80年代以降の情報処理全体の質的変化――数値計算処理からデータベース処理(非数値計算処理)へ――を踏まえた重要な研究課題であることがあらためて印象づけられた。さらに従来のたぶんに概念・観念的な研究姿勢から一歩脱却した現実的なアプローチが主流となってきており、性能評価を踏まえた方法論が焦点となってきた。

DBMの研究の動機及び必要性としては重要性の高まってきたDB処理が現状のノイマン型計算機アーキテクチャが不得意とする非数値処理・連想処理・データ構造処理及び大量データ処理を要求しており、データベース管理システム(DBMS)を今後強力に発展させる引き金としてDB処理指向の計算機アーキテクチャの研究・開発は重要であることが挙げられる。[10]

こうした背景とは別にDBMの研究が一般的な計算機アーキテクチャの研究と比べて有利な条件として次のことがいえる。すなわち、DBMS自身が過渡的な状況であって、必ずしも過去のソフトウェア遺産等の拘束を受けず比較的新しいアプローチをとることが現実的に可能であり、従来のアーキテクチャと共存可能でスムーズにシステムに組み込める可能性が強い。しかし、研究の難しい点はDBMS自身、まだ、第1世代という観をまぬがれず、システムを構築する際、各基本データモデルによって要求される処理機能が微妙に異なっていることである。そこでDBMを研究するにあたっての最初の選択肢はモデルから独立したアプローチをとるかモデルを指向したアプローチをとるかということである。

(1) データモデルから独立なアプローチ：このアプローチは代表的なモデルに共通する機能や処理に着目して、記憶媒体及びチャネルのDB処理に関した機能の強化を図るものであり、次の4つに分類される。① DASD型アプローチ：基本アーキテクチャを本質的に変えずにチャネルやディスクの強化を図る。② アクセスメソッド及びインテリジェント・ディスク型アプローチ[28]：現状のデータベースファイルシステムの基底部分を形成しているアクセスメソッドの機能をハードウェア・ファームウェアで実現して処理向上を図るアプローチである。③ データベース・ニュークリアス アプローチ：②のアプローチにDBMSで必要な機能をさらに追加して、そのレベルをDBMの基本アーキテクチャとするアプローチである。④ 統合データベースマシン型アプローチ[9]：DBCに代表されるようにDB処理に必要でかつ処理上のオーバーヘッドとなる機能を1つのコンセプトのもとに結集したシステムを目指すアプローチである。これらのアプローチはDBMの一般化の点では有利であるがデータモデルを効率よくサポートするという観点からはやや不満な点があると思われる。

(2) データモデルを指向したアプローチ：この種のアプローチは特定のデータモデルを想定してDBMを設計するもので、各データモデルの特徴を考慮した基本アーキテクチャが設定される。このアプローチは内部データ構造の違い、すなわち、データ間の関係を① 構造的に表現するか ② 非構造的に表現するかによって大別される。前者は階層型モデルに代表され、基本的にはデータ構造に沿った手続き的な操作によって処理の効率向上がはかられるがデータ構造に沿わない操作の効率はよくない。後者はリレーショナルモデルに代表され、非構造的なフラットファイル表現に対して集合論的な演算体系によって処理を行なう。リレーショナルモデルは理論的裏付けが確固たるものであり、その単純なデータ構造と演算体系はシステム的アプローチとしては他のモデルに比べて圧倒的な魅力があると思われる。

筆者らは以上の論点からリレーショナルモデルを指向したDBMアーキテクチャの研究を進めている。本論文では現在、研究・設計を行なっている統合リレーショナルデータベースマシンSPIRITの基本アーキテクチャについて述べる。

2. リレーショナル・データベースマシン(RDBM)に関する基本的考察とSPIRITの設計思想

リレーショナルデータベース(RDB)を主に対象としたDBMの代表例としてRAP(original RAP[3], RAP-2,3[4,5,6]), RARES[24], CASSM[7], DIRECT[20], CAFS[21]等があるが それらのアプローチの得失を論じながらSPIRITのアーキテクチャの基本概念と特長を述べる。

基本アーキテクチャの設定基準(DBストア依存型 VS 独立型)

初期のRDBMではディスクなどの回転型記憶媒体に直接連想処理機構を付加した方式が主流であった。しかし、最近ではこうしたアーキテクチャをもってしても リレーショナル演算、特にprojection, joinなどの演算に対して、かえって大幅なオーバーヘッドを招き、現実的な方法論としての価値を失いつつある。この方式の代表格であったRAPプロジェクトの研究の推移を見ても、現在のRAP2ではlogic per track(LPT)方式を断念しており、DBストアと演算機能を直接結びつけずにマシンを構成することを考えている。演算はCCDメモリと連想処理機能を結びつけたセルを複数配置して遂行される。さらに出力の効率を上げるためにデータの局所性を高めるように改良されている。現実的なアプローチとして、RAPのようなLPT方式よりむしろトラック並列読み出し機構をDBストア上に充実した方式の方が現在の技術の延長上からも望ましく、確実な性能向上が得られる。また、一歩譲って、LPT方式の最大能力を発揮できる状況を考えた場合でも 各トラック単位にトラック容量分のバッファを用意しないと実質的なデータ転送時間の削減が保証されず、実質的な処理向上が期待できない。そればかりか、ディスクに直接論理機能をつけた意味が帳消しになってしまう。

以上の論点から、SPIRITではDBストアからは独立したアーキテクチャを基本として、高速なリレーショナル演算方式とDBストアからのデータステージング能力の向上技法、さらにマルチトランザクション環境下での各種のスケジューリングや最適化が有機的に結びついたシステムとなっている。また、DBストアとしては その容量と現実的な技術の完成度から浮動ヘッドディスクを想定し、その出力問題を解決するためにトラック並列読み出し機構を追加機能と考えている。

2.1 リレーショナルデータベース演算の方式

[1] リレーション生成方式, 固定マークビット方式 VS 動的マークビット(TMB)方式

リレーショナル演算体系については Coddの提示した関係代数, 関係論理に基づいた操作をいかにハードウェア化するかという方法論が重要である。まず、DIRECTのたえず新しいリレーションを生成していく方法であるが このアルゴリズムではリレーションの再構成を各演算ごとに行なう必要があり、そのコストは甚大となる。さらにこの方法は手続き的でかつソフトウェアアルゴリズムの域を一歩も脱していない。

一方、RAPの演算方式では リレーション内の演算は タップルに固定して付加されているマークビットを使って遂行され、原則的にリレーションの再構成は必要ない。しかし、リレーション間の演算では再構成の必要があるばかりでなく、マークビットがタップルに固定的に付加されているので、① マークビット用の格納エリアが常に固定的に必要である、② マークビットの数が複雑な演算の実行及びタップルの共有(マルチプログラミングの多重度)を制限する、という点で好ましくないとされている。

SPIRITでは上記の点を考慮して、リレーショナル演算体系の基本を次のように設定している。① 属性単位の演算方式: 基本操作対象データの単位を属性と規定し、属性独立な演算体系を構築している。そのため必要な属性だけに対して演算が遂行される。データ出力が属性単位の入出力の自由度が保障された方式としており、出力の効率向上を図っている。② 動的マークビット方式: 動的マークビット方式とは リレーション生成方式(DIRECT方式)と固定マークビット方式(RAP方式)の長所を融合した方式で 属性に対する条件検索の結果を有効タップルを示すビットマップという別のデータ構造として 演算系中に組み込まれている。さらにマークビットのデータ構造を ① VTF: 有効タップル識別子(TID)のビットマップ ② VTF_g: ある属性に対するgroup by の結果を示し、{VTF}で定義される。③ AIR: リレーション間の演算の結果が保持され、joinなどの場合はリンクテーブルとしての役割を果たし、{VTF_g for R.A., VTF_g for S.B.}で定義される。以上のように体系化し、各データ構造間の演算を定義することによって、リレーションの再構成なしに演算が遂行可能である。この方式によって ① 原則としてリレーションの再構成を行なう必要がない, ② 属性を処理するうえでの共有に制限がない, ③ ビットマップは動的にアロケーションされるのでDBストアの負担にならない, などのメリットが得られる。さらに 手続

き的あるいはソフトウェア的アルゴリズムではなく、ハードウェア化処理に適した方式となっている。

[2] リレーショナル演算の効率を考慮したデータ構造空間（traditional方式 vs SPIRIT方式）

SPIRITではリレーションの定義式から、その想定するデータ構造空間を次のように提案する。① ベースドメイン（BD）：重複のない、データから構成され、リレーションの定義における定義域（ドメイン）を忠実に表現したものである。② コード化リレーション（CR）：BDから写像されて、各実データに対応するコードで構成される属性の集合である。

コード化の利点として ① スキーマ空間全体のデータ容量の削減，② 演算対象データの圧縮によって転送時間・演算時間の減少，③ 集合演算の方法論からも大きなレンジをもって広がる空間を連続アドレス空間に圧縮する効果 が挙げられる。さらに③により限定された競合のないアドレス（コード値）空間を使って projection，equal join などの演算の高速化が可能となる。実データ空間とコード化空間の間の相互変換のオーバーヘッドはパイプライン処理により、吸収可能であり、コード化空間での演算コストの低減を考えれば許容できる。（図1参照）

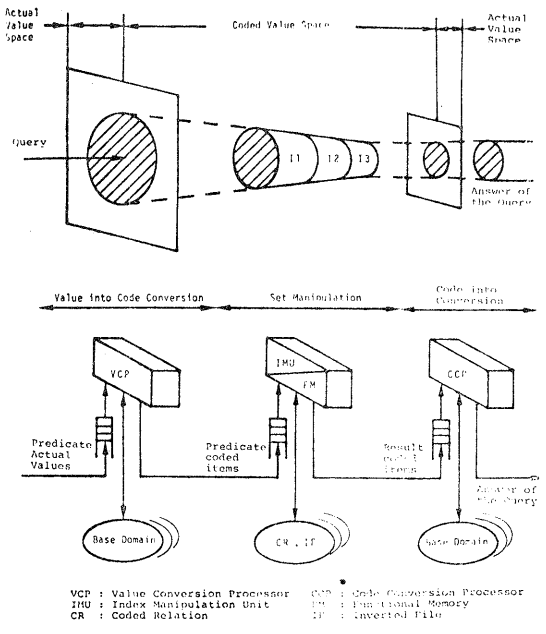


図1 SPIRITにおけるリレーショナル演算の基本概念

[3] 並列・非同期処理方式

前述したようにRDBにおいては条件検索は並列処理によって、1/(セルの数) の処理時間となる。しかし、projection や join のような演算では、本質的にセル間の交信が必要となり、各セル間の非同期処理はこの点が難点があった。SPIRITでは詳細は後述するが projection はセル間の交信なしに完全な非同期並列処理で遂行でき、join などのリレーション間の演算も高々2回の交信が必要となるだけで演算の大部分は非同期並列処理で行なわれる。これによって従来、並列検索のジレンマ ― 並列検索の効率のみを重視してセル数（並列性）を増すと、リレーション内・間の演算においてセル間交信の増加をまたらし、その同期のため、せっかくの並列性が生かされない ― に1つの解が与えられたといえる。

[4] 機能分散処理方式

SPIRITでは集合演算の論理は ① コード化リレーションに対する条件検索の結果はすべて動的マークビット（TMB）に変換される，② TMB間の3種のデータ構造もその原始要素はビットマップ表現であり、TMB間の演算によってコード化リレーションなしで最終的な有効TID情報を得ることができる。

前者はFMと呼ばれる機能メモリによって遂行され、後者はIMUと呼ばれるビットマップ間の演算を専門に行なうユニットで遂行される。

以上のようにSPIRITではリレーションに対する基本操作を機能分散方式によって行なうとともにFM/IMUのカップル化されたセルを複数配置したシステムをPure SPIRIT（PS）と呼び、PSでは各セルは大部分のリレーション演算において並列・非同期に遂行される。

[5] データ駆動型によるSPIRIT命令のスケジューリング

SPIRITではデータステージングモジュールからのデータ供給にタイミングを合わせて、集合演算モジュールにおけるSPIRIT命令は実行に必要なデータがそろったものから順にスケジューリングされる。つまり、非同期に動作しているモジュール間での同期の問題がデータ駆動方式のスケジューリングの中に吸収されている。また、集合演算モジュール内の機能プロセッサ間のパイプライン処理制御もこの概念を使用すると同期をとりやすい。すなわち、このスケジューリングはSPIRITの非同期・並列処理のアーキテクチャに適している。

2.2 データステージング機能と階層記憶方式

一般にデータ格納媒体と演算処理系が独立している場合、I/Oネック(データ供給能力の限界)が問題となってくる。すなわち、データアクセス時間、データ転送時間及び演算プロセッサの許容メモリ容量の制限に起因するデータリプレイスメント、ページフォールトなどのオーバーヘッドが問題となる。そこで、SPIRITでは2.1で示した高速集合演算アーキテクチャの能力を最大限に引き出すため次のようなデータステージング機構を用意し、統合化されたシステムを考えている。

[1] データ圧縮技法：2.1で述べたコード化を基礎としたデータ構造を採用することによって①演算対象データの容量圧縮による実質的な転送時間の短縮、②格納媒体上での局所性向上によるアクセス時間の短縮、及び③集合演算時の演算コストの削減を達成している。

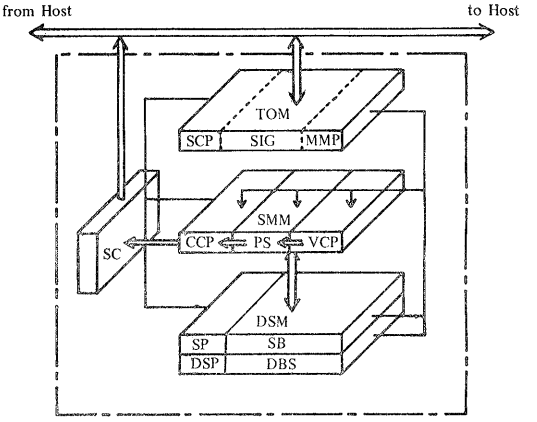
[2] 階層記憶方式：DBストアと演算モジュールの間にステージングバッファ(SB)と呼ばれる緩衝メモリシステムを用意する。これにより、DBストアと演算モジュールの非同期操作が保障され、データ局所性及び先回りステージングの効果が生かされるとともに、見かけ上のアクセス時間の短縮が達成される。

[3] データ格納単位としての属性の採用：データの入出力及び演算の単位をSPIRITでは演算の最小論理単位である属性として、システム全体を構築している。そのためデータ格納単位もリレーション(タップル集合)ではなく、属性である。これにより処理時に不要な属性が転送されず、リレーション単位の格納方式に比べて、細かい入出力制御が可能であり、データ転送時間が短縮される。

[4] トラック並列データ配置と並列読み出し機構：現状の浮動ヘッドディスクをDBストアと仮定して、トラック並列読み出し機構を付加する。これによりトラック数に比例したデータ転送能力の向上が期待できる。各トラックには属性の部分集合が均等に格納されており、並列読み出しが可能となる。

[5] データステージングの最適制御：SBと演算プロセッサの間のデータステージングを最適にスケジューリングするため、ステージングプロセッサ(SP)が導入されている。また、浮動ヘッドディスク群の最適アクセス制御を行なうためにディスクスケジューリングプロセッサ(DSP)が用意されている。これらにより記憶階層間のデータ移動を極力減らす。

さて、データステージングモジュール(DSM)は独立のストラテジで動作し、演算モジュールの命令実行のスケジューリングもSBに到着したデータに関連する命令から自動的に発火するというデータ駆動型の方法が導入されており、そのためこのシステムにおいてはページフォールト実行時のデータリプレースメントという状況を本質的に回避している。SPIRITのシステムアーキテクチャを図2に示す。



TOM : Transaction Optimization Module
SCP : Security Check Processor
SIG : SPIRIT Instruction Generator
MMP : Model Mapping Processor
SC : SPIRIT Controller
SMM : Set Manipulation Module
VCP : Value Conversion Processor
PS : Pure SPIRIT
CCP : Code Conversion Processor
DSM : Data Staging Module
SP : Staging Processor
SB : Staging Buffer
DSP : Disk Scheduling Processor
DBS : Database Store

図2 SPIRITのシステムアーキテクチャ

3. SPIRITのシステムアーキテクチャ

SPIRITは図2に示されるように3つの機能モジュールから構成される。

3.1 システムコンポーネント

[1] トランザクション最適化モジュール(TOM)：これはユーザからの問い合わせを最適化されたSPIRIT命令におとすモジュールで、①ユーザ・ビューに対する問い合わせをシステムレベルのモデルにマッピングするモデルマッピングプロセッサ(MMP) ②セキュリティチェックを行なうセキュリティチェックプロセッサ(SCP) ③SPIRIT命令を生成するSPIRIT命令生成プロセッサ(SIG)から成る。

[2] 集合演算モジュール(SMM)：これは集合演算を行なうモジュールで、①問い合わせ中のオペランドアイテム値をコードに変換する実値変換プロセッサ(VCP)

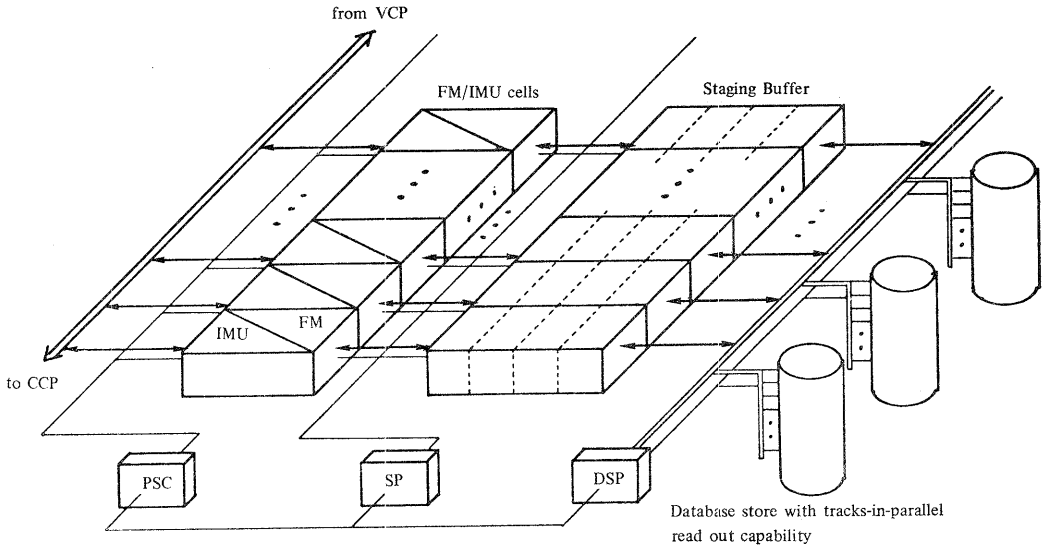


図3 Pure SPIRIT及びそのデータステージング系の構成図

② コード化リレーション(CR)に対する連想処理を行なう機能メモリ(FM)と動的マークビット(TMB)の処理を行なうインデックス操作ユニット(IMU)から成るPure SPIRIT(PS)(図3を参照) ③ 演算結果の出力コード群を実アイテム値に変換するコード変換プロセッサ(CCP)から成る。

[3] データステージングモジュール(DSM): これはDB本体を格納するとともにSMMへのデータ供給を行なうモジュールである。データの供給力を上げるためにディスクとステージングバッファ(ディスクキャッシュ)の階層構造をとり、それぞれに専用のスケジューリングプロセッサ(DSP, SP)を付加することによってデータの先読みを行なう。

[4] SPIRITコントローラ(SC): これは上記の3つの機能モジュール間の同期をとるとともにそれらを管理し、スループット及び応答時間の向上、さらにマルチトランザクション環境下の一貫性の維持などを行なう。

3.2 基本データ構造

リレーショナルモデルの実用化の際の最大の問題点は、関係の維持を属性値の重複表現にゆだねることに起因するデータ量の増大である。SPIRITではこれに対処すべく、コード化を基礎としたデータ構造、すなわち、ベースドメインテーブル(BDT)、コード化リレーション(CR)転置ファイル(IF)などを採用している。(図4を参照)

[1] ベースドメインテーブル(BDT): これは、各スキーマ空間における重複のない実アイテム値の集合(定義域)を表現した、実アイテム値と対応コードの変換テーブルである。初期的なコードの割り当て(BDTの生成)は実アイテム値に対して自然数を1から順次割り当てていくもので実アイテム値の順序はハッシングによって決まる。問い合わせ処理時のコード化も実アイテムのハッシングによる対応コードのアクセスによって行なわれ、コード化されたアイテムはそれ以後、そのコードをアドレスとしてIFなどをアクセスすることができる。同一スキーマ内ではコードは一意的にふられるためjoinなどの演算をコードのまま行なうことができる。

[2] コード化リレーション(CR): SPIRITにおける集合演算の対象はBDTを用いて変換されたコード化アイテムから成るCRである。CRは転送時間のオーバーヘッドを緩和するように属性単位で格納される。このため物理的に分離された1つのタップルに属するアイテムは各属性内でのそのアイテムの先頭アイテムからの変位を示すタップル識別子(TID)によって論理的に結合される。また、集合演算実行時のタップル並列性を考え、属性はセクションという単位に分割される。1つのCRの各属性の1セクションのアイテム数は定である。

[3] 転置ファイル(IF): SPIRITではIFはその維持コストを考えてオプションとして持たれる。IFの表現形式としてはビットマップ形式が

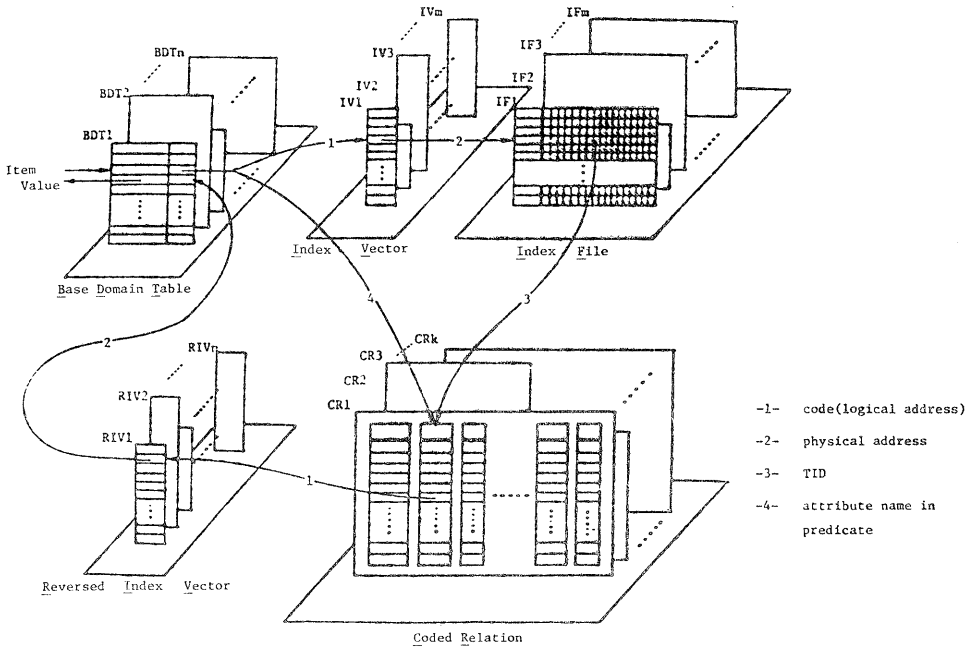


図4 基本データ構造空間

とられている。これはFMでの検索結果と同形式であり、これによってIFを集合演算体系の中に組み込んでいる。

4. SPIRIT命令及びその生成方法と評価基準

SPIRITはDSL-ALPHA[22]を構文的に制限したKeio-ALPHA(K-ALPHA)を外部インタフェース言語としている。SPIRIT命令生成プロセッサ(SIG)はK-ALPHAを解釈し、SPIRITアーキテクチャに最適なSPIRIT命令を生成している。本章ではSPIRIT命令及びその生成方法と評価基準について述べる。

4.1 K-ALPHA ダイアグラム

SIGはSPIRIT命令を容易に生成するためにK-ALPHAを一端、K-ALPHAダイアグラム(KAD)という中間形に変換している。

KADはノードと枝から構成され、原則としてリレーション間の関係を表現する。ノードはタップル変数を示し、枝はリレーション間の関係を表わす。タップル変数は制限子によって束縛されているものがあり、これを表わすために3種類のノードが用意されている。(図5参照) また、KADの例を図6に示す。

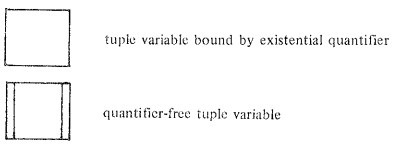


図5 KADの3種類のノード

```
[Sample database]
     S  (S#,SNAME,SAL,CITY)
     P  (P#,PNAME,COLOR,WEIGHT)
     J  (J#,JNAME,CITY)
   SPJ  (S#,P#,J#,QTY)

[DSL-ALPHA]
   RANGE  SPJ  SPJX  SOME
   GET  W  (S.CITY,J.CITY) : (S.S# = SPJ.S# ∧ J.J# = SPJ.J)

[KAD  representation]
```

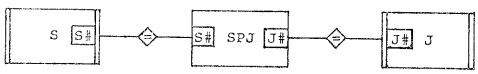


図6 問い合わせと対応するKAD表現

KADはノード・枝・ノードの形をした基本的な処理エレメント（これをベースタイプと呼ぶ）の結合したものと見ることができる。ベースタイプはちょうど述語論理の項（term）に相当する。そして、この結合方式にはリンクタイプ、リングタイプなど数種類ある。一般に1つのKAD表現はいくつかの結合方式を含んでいる。図6のKAD表現は2つのベースタイプが結合したリンクタイプだけから構成されている。

ベースタイプはどんな順で評価されてもよい。たとえば、すべてを並列に評価することができる。しかし、最適な評価順序は対応する結合方式により決定される。個々の結合方式はそれぞれいくつかの処理パターンを持っており、状況に合わせてそれを選択することができるのである。図6ではリンクタイプなので、2つのベースタイプを並列処理してもいっぺんに逐次処理してもよいが、この選択は4.3で述べる評価基準に基づいて決定される。

4.2 SPIRIT命令

SPIRIT命令は属性や動的マークビット（TMB）をオペランドとし、演算結果としてTMBを出力する。SPIRIT命令の実行はFM及びIMUが行なうが、その制御はデータフローに基づいてPure SPIRITコントローラ（PSC）が行なう。つまり、命令の実行に必要なオペランド（属性、TMB）がすべてセットされたとき、その命令は実行可能となる。これを表現したものが図7である。そして、PSCは実行可能状態にある命令の中からPure SPIRIT内の状況に最も適した命令を選び実行させる。SPIRITではFM, IMUなどの各プロセッサが非同期に動作するのでデータフローで命令を制御することは非常に効果的である。

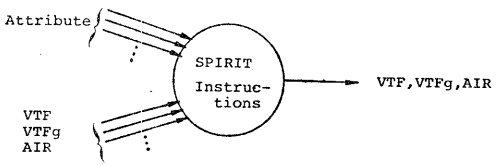


図7 SPIRIT命令のデータフロー表現

SPIRIT命令は基本的には1オペランド演算である。これは命令がデータフローで制御されるため、データの待ち合わせ時間を最小にすることがシステムパフォーマンスの向上につながると考えられるからである。しかし、待ち合わせ時間を犠牲にしてもオペランドをそろえて実行した方が効率の良い処理（projection, joinなど）に対しては複数オペランドを持つ命令（DE, LKなど）を用意している。SPIRIT命令を表1に示し、以下にその説明を行なう。

オペランドの1つであるTMBには次のようなものがある。VTF：有効なタップルを示すフラッグ、VTFg：属性の各値に対するVTFの集合、AIR：リレーション内のタップルの対応関係を表わすビットマップ表現のリンクテーブル

Pure SPIRITに対する命令は次のようである。SL：selectionなどのリレーション内の演算命令、LK：joinなどのリレーション間の演算命令、GR：指定された属性に対するグループ化命令、SR：グループ化されたリレーションと他のリレーションの間に包含関係が成立しているかどうか調べるための命令（divisionを行なう際に使用される）、DE：指定された属性に対して重複をはずす、OP：指定された属性を出力する命令（その属性に対して重複があればたとえリレーション間にまたがっていてもはずす）、DL：VTF

表1 SPIRIT命令

Module		function	numonic	output type	source type	*1	*2	*3
VCP		encode	EC	code	value,AT	X	X	0
CCP		decode	DC	value	AT	X	X	0
DSP,SP		stage up	SG	VTF	AT	X	X	0
PS	FM	intra-relation operation	SL	VTF	AT,[AT]	0	0	1
		partition a relation into groups	GR	VTFg	AT	X	X	2
		inter-relation operation	LK	VTF,VTFg,AIR	AT,AT	0	X	6
		set restriction	SR	VTFg	AT,AT,VTFg	0	X	2
		duplicate elimination	DE	VTF	{ATi}	X	X	1
		output and reconfigration	OP	code	{ATi},VTF,AIR	X	X	3
		update	UP	———	AT,VTF	X	0	0
		insert	IS	———	{ATi}	X	X	0
		delete	DL	———	VTF	X	X	0
	IMU	operation on VTF's	OF	VTF	VTF,[VTF]	X	X	1
		operation on VTFg's	OG	VTFg	VTFg,[VTFg],[VTF]	X	X	1
		operation on AIR's	OA	VTF,AIR	AIR,[AIR],VTF	X	X	1
		define image	DI	VTF,VTFg,AIR	VTF,VTFg,AT	C	X	4

*1 Comparison operator　*2 Operand register　*3 The number of modes
AT: Attribute

で指定されたタップルを削除する命令，UP：VTFで指定されたタップルを変更する命令，IS：タップルを挿入する命令，（以上FMに対する命令）
OE：VTFに対する操作命令（主として論理演算），OG：VTF_gに対する操作命令，OA：AIRに対する操作命令，DI：AIRなどの一時的情報の定義命令。（以上IMUに対する命令）

さらに，VCP，CCP，SP，DSPに対する命令として，EC：実値のコード化命令，DC：コードから実値への変換命令，SG：DBストアからSBへのステージング命令，がある。

4.3 KADの評価方針

SIGは次のような基準でKADを評価し，最適なSPIRIT命令を生成する。

すなわち，中間結果 VS 並列性 を評価基準とする。

まず，リソースのことを考えると，できるだけ中間結果（TMB）を出さない処理を選ぶことが望まれる。しかし，TMBを最小にする処理は逐次処理となり，処理効率を悪化させる。逆に処理効率を向上させようとすると，並列度の高い処理が選ばれ，TMBを増大させる。SIGはこの問題を解決するため，KAD表現の局所性を考慮して評価している。KADが複雑になるとそれをいくつかの部分（sub-KAD）に分割することが考えられる。そして，個々のsub-KAD内については逐次処理を行なうようにし，sub-KAD間では並列処理を行なうようにする。図6の場合，並列処理でも逐次処理でもTMBの量はほぼ同じなので，並列処理が行なわれる。図8が図6から生成されるSPIRIT命令のデータフロー表現である。

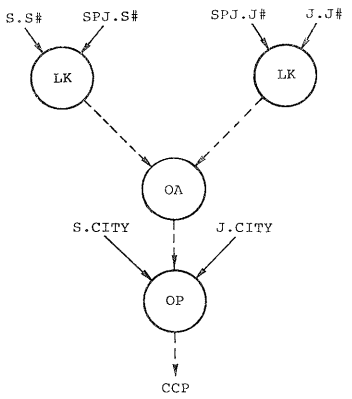


図8　図6に対応するSPIRIT命令のデータフロー表現

5. SPIRITのリレーショナルデータベース演算の方式

この章ではSPIRITのリレーショナルデータベース演算において中心的役割を果たすPure SPIRIT（PS）の集合演算の方式について述べる。PSは機能メモリ（FM），TMB操作ユニット（IMU）及びステージングプロセッサ（SP）によって高レベルの集合演算を実行する。PSはその特徴として，2.1節で述べたように(1)属性単位の演算方式 (2)動的マークビット方式 (3)並列・非同期処理方式 (4)機能分散処理方式 (5)データ駆動型のスケジューリング方式を採用している。

5.1 Pure SPIRITの扱うデータ構造

PSにおいて扱われるデータ構造は表2に示されるように2つに大別される。1つはリレーションの属性（のコード値集合）であり，他方はVTF，VTF_g，AIRと呼ばれる動的マークビット（TMB）の集合である。TMBはリレーション内の有効タップルや2つのリレーション間のタップルの対応関係を示すために用いられる。PSの処理対象はこれら2つのデータ構造であり，処理の結果はTMBの形で保持される。

表2　Pure SPIRITの扱うデータ構造

Data Structure	Constituent Element	Correspondence
Attribute	Section	Attribute of a Relation
VTF	Mark Bit	1-dimensional Bit Map
VTF_g	VTF	Inverted File
AIR	VTF_g , VTFg	Link Table

〔1〕VTF：集合演算により選択されたタップルとそうでないタップルを識別するために各タップルに対応させてフラッグが与えられる。これらのフラッグをタップルの配置順に配列したビットマップをVTFと定義する。従って，タップルの識別子（TID）はフラッグのアドレスと等しい。VTFは主として，問い合わせの中間結果を示す一時的なデータとして扱われ，また，最終的な結果もVTFで保持できる。VTFは実際には2階層構造である。1つは上述したように1つのフラッグに1つのタップルが対応するものであり，他方は1つのフラッグに1つのセクションが対応するものである。前者は密なVTF，後者は疎なVTFと

呼ばれ、密なVTFがすべて"0"であったり、すべて"1"であったりする場合に起こる無意味なデータの保持やその処理が回避できる。

[2] VTFg： VTFgはVTFの集合であり、group byの結果を示すために使われる。各VTFはグループ化する属性に存在する各値に対応して作られる。従って、VTFgは図9に示されるように転置ファイルの構造に等しい。

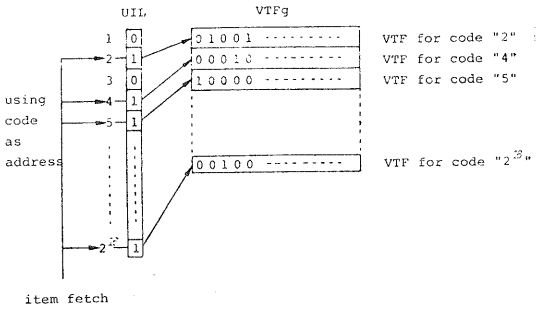


図9 UILとVTFgのデータ構造

[3] AIR： AIRはjoin及びimplicit joinの結果を新たなリレーションを生成することなしに示す2次元ビットマップであり、VTFの集合である。ビットマップの各行はjoinされる一方のTIDに、各列は他方のリレーションのTIDに各々対応する。joinの結果として両リレーションの結合されるべきタップル同志の各TIDに対応する位置のビットに"1"がセットされる。このビットマップは非常に大きくなる可能性があるので、実際にはAIRの物理構造は一方のリレーションに対応するVTFgの各VTFと他方のリレーションに対応するVTFgの各VTFを共通の値のもの同志で結合させることにより2次元のビットマップと同様の情報が保持される。

5.2 Pure SPIRITの基本操作

SPIRIT命令はPure SPIRITコントローラ(PSC)により以下に示すような基本操作に展開される。基本操作はFMで実行されるものとIMUで実行されるものに大別される。FMでは属性を対象として処理を行ない、その結果をVTF, VTFg, AIRで示す。IMUではFMで生成されたこれらのTMBに対して単純な論理演算などの処理が行なわれ、その結果もVTF, VTFg, AIRのいずれかで表現される。このようなFMとIMUの機能分散により、各々単純な操作を行なうだけで高度な集合演算を実現できる。

[1] FMの基本命令：

① RF-Set UIL [(Rn:Dm);(VTF#n);(UIL#n)]

(RF: レコードフェッチ)

FMセル内に格納されたRnリレーションの属性Dm内の各コード値をVTF#nに示された有効なTIDに従ってフェッチし、UIL#nにフラッグをセットする。ユニークアイテム ロケーション(UIL)は1次元のビットマップであり、図9に示されるようにそのビットマップの各フラッグのアドレスは対応するコード値と等しくなっている。UILは値が"1"であるフラッグのアドレスと等しいコード値がその時点において演算対象となっていることを示す。

② RF-Set VTFg [(Rn:Dm);(UIL#n,VTF#n);(VTFg#n)]

Rnリレーションの属性Dm内の各コード値をVTF#nが示す有効なTIDに従ってフェッチし、VTFg内の対応位置にフラッグをセットする。VTFg内の各VTFは図9に示されるように各コード値と対応しており、VTFの内容であるフラッグ列はそのVTFに対応したコード値をもつタップルのTIDを示す。

これら2つの命令はprojection, join, implicit join, group byの処理において用いられる。

③ Conditional-Seach [(mode x);(qualification);(VTF#n,VTF#m);(VTF#k)]

mode 1： 検索条件で指定された(リレーションの)属性から条件を満たすタップルを比較操作により検出し、その結果をVTFで示す。この命令はrestrictionを処理する際の基本命令である。

mode 2： 検索条件で指定された(リレーションの)属性の持つコード値と同じコード値を他方で指定された(リレーションの)属性に対する比較操作により検出し、その結果をVTFで示す。この命令はFMセル内に指定された2つの属性が同時に存在可能な場合のimplicit join, intersection, differenceの処理過程において用いられる。

[2] IMUの基本命令： IMUは表3に示されるように2つのVTF同, またはVTFg同, またはAIR同で対応するフラッグ同志の論理演算などを行なう。UILがVTFと構造が同じなので2つのUIL同の論理演算も実行できる。IMUは単純な論理演算を行なうだけなのでビット数に対応して並列性の向上が可能である。

表3 IMU基本命令

AND	(VTF-#3)←(VTF-#1).AND.(VTF-#2)
OR	(VTF-#3)←(VTF-#1).OR.(VTF-#2)
XOR	(VTF-#3)←(VTF-#1).XOR.(VTF-#2)
NOT	(VTF-#2)←NOT.(VTF-#1)
SET ALL	(VTF-#1)←SET ALL.(0 or 1)
SET BIT	(VTF-#1)←SET BIT.(TID , 0 or 1)
SELECT First BIT	(TID)←SFB.(VTF-#1 , 0 or 1)
COUNT BIT	(number)←COUNT BIT.(VTF-#1, 0 or 1)
TEST ALL	(TEST flag)←TEST ALL.(VTF-#1)
TEST BIT	(TEST flag)←TEST BIT.(VTF-#1,TID)
COPY	(VTF-#2)←COPY.(VTF-#1)

5.3 SPIRIT命令の展開

集合演算において基本的かつ代表的な演算である selection, group by 及び equi-join についてそのアルゴリズムを 5.2 で述べた基本命令により記述する。

[1] selection (SPIRIT命令の"SL"):

```
Conditional-Search [(mode 1);(R1:D1 = "a");
                    (VTF#1,   );(VTF#2)]
```

[2] group by (SPIRIT命令の"GR"): group by は projection における重複の排除, division さらに転置ファイル生成のための基本的な演算である。

```
RF-Set UIL [(R1:D1);(VTF#1);(UIL#1)]   ——→ 図10の(2)に対応
RF-Set VTFg [(R1:D1);(UIL#1,VTF#1);(VTFg#1)] → 図10の(3)に対応
```

図10にこの処理フローを示す。

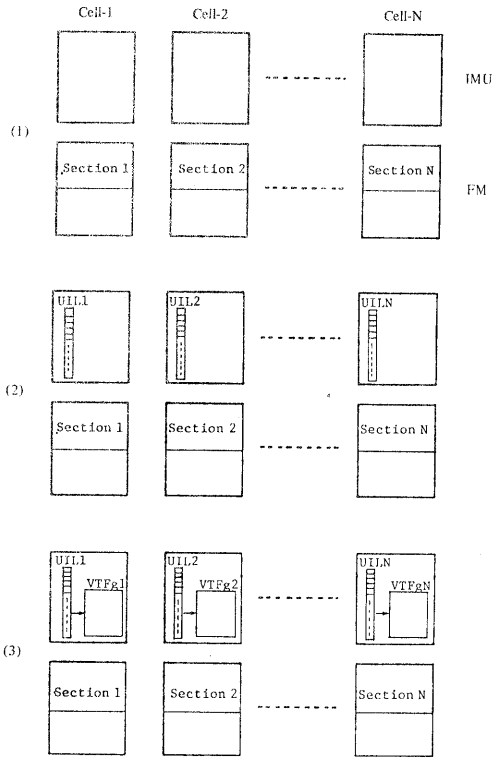


図10 group by演算の処理フロー

[3] equi-join (link; SPIRIT命令の"LK"):

```
RF-Set UIL    [(R1:D1);(VTF#1);(UIL's#1)]   } 図11の(2)に対応
RF-Set UIL    [(R2:D2);(VTF#2);(UIL's#2)]   }
OR            [(UIL's#1);(UIL#3)]           }
OR            [(UIL's#2);(UIL#4)]           } 図11の(3)に対応
AND           [(UIL#3,UIL#4);(UIL#5)]       }
AND           [(UIL's#1,UIL#5);(UIL's#6)]   }
AND           [(UIL's#2,UIL#5);(UIL's#7)]   }
RF-Set VTFg   [(R1:D1);(UIL's#6,VTF#1);(VTFg#1)] } 図11の(4)に対応
RF-Set VTFg   [(R2:D2);(UIL's#7,VTF#2);(VTFg#2)] }
Image-AIR     [(UIL's#6,UIL's#7);(VTFg#1,VTFg#2);(AIR#1)]
```

図11にこの処理フローを示す。

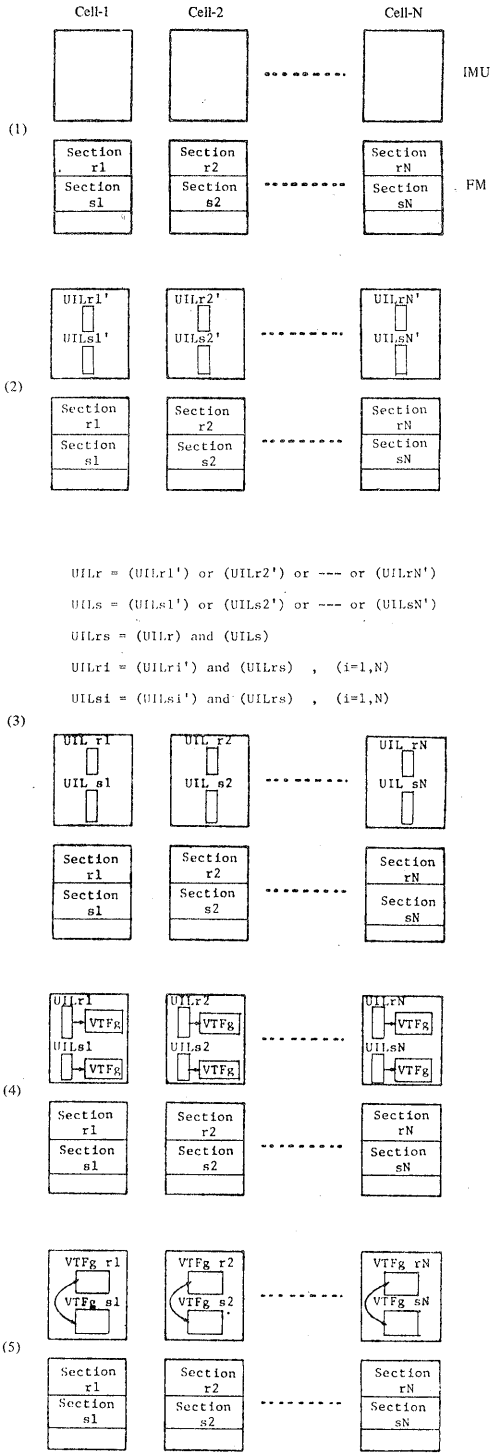


図11 link 演算の処理フロー

linkは、新しいリレーションを生成することなくjoin及びimplicit joinを実現できる演算である。

以上3つの代表的な演算の他、unionや複数の属性を対象として重複を排除するcompound projection, "="以外のオペレータ(>, ≥, <, ≤など)によるθ-joinは次のように処理される。

[4] union : unionは2つのリレーション間で共通なタップルを一方のリレーションから排除し、両リレーションを1つのリレーションと見なすことにより達成される。これには各々のリレーションに対応していたVTFを結合して1つのVTFと見なし、さらに2つのリレーションを1つのリレーションと見なした後、そのVTFを使って重複を排除する。従って、compound projectionと同様の操作で処理される。

[5] compound projection : 各属性ごとにgroup byの操作を行ない、生成された各属性に対応するVTFg間の論理演算によって重複を排除するがこの論理演算の実行回数は問題となるが、最大有効タプル数と属性数の積の回数だけで完遂する。

[6] θ-join : θ-joinではコード値ではなく実アイテム値が必要となる。従って、この場合はjoinの対象となる両方のリレーションをFMセルに配置し、比較操作により、AIRを生成することになる。従って、比較のためのデータフェッチ回数が増すので、従来の方式に比べ効率は向上しないが両リレーションのオペランドの属性内の値の最大値と最小値の指示により意味のない比較を排除している。

*** joinの結果として生成されるAIRは新たなリレーションを生成せずにjoin及びimplicit joinの結果の保持を可能にしたがjoin, implicit join演算ののちに他の集合演算をほどこす場合にはそれらの演算はjoin, implicit joinの演算前の元のリレーションに対してFMで処理を行ない、それによって得られるVTF, VTFg, AIRのいずれかを先に得られたAIRとの間でIMUにおいて論理演算を実行することにより遂行される。たとえば、1つのjoinの結果生成されたリレーションに対してさらに他のリレーションとjoinする場合にはPure SPIRITではそれらのjoinを独立に実行し、それぞれのAIRを生成したのち、それらのAIR間で行列積をとる。この演算の結果得られる新たなAIRがjoin後のjoinの結果を表わすことになる。

以上述べたようにPure SPIRITは多回の比較検索や新しいリレーションの生成などのオーバーヘッドを避け、集合演算をFMとIMUによりシステマティックに実行している。

5.4 Pure SPIRITの演算方式の評価

前節で述べたPure SPIRITの演算方式の評価を表4に示す。この演算方式はprojectionにおける重複排除の基本演算であるgroup byやjoinの基本となるlink演算(equi-joinの場合)において比較操作を必要としていないので処理時間は従来の方式に比べてかなり向上する。

6. おわりに

本稿では現在、筆者らが検討を進めているリレーショナルデータベースマシンSPIRITの基本アーキテクチャを特徴づけるリレーショナルデータベース演算方式を中心に述べたものである。ここで述べた演算方式は原則として対象データがコード化されていることを前提としており、それ以外の主要な特長として次の5つの項目が挙げられる。① 属性単位の演算体系：基本操作対象データの単位を属性と規定することにより必要な属性だけに対して演算が遂行され、

表4 Pure SPIRIT方式の性能評価

architecture / operation	sequential machine		conventional cellular logic		Pure SPIRIT	
	memory reference time	comparison time	memory reference time	comparison time	memory reference time	comparison time
Selection	t	t	t/N	t/N	t/N	t/N
Group By	t(t-1)/2 + t	t(t-1)/2	t(t-1)/2N + t	t(t-1)/2N	2t/N	0
Link (equi-join)	trts + tr + α	trts	trts/N + tr + α	trts/N	2(tr+ts)/N	0

N : the number of cells
t, tr, ts : the number of tuples
α : creation time of a new relation

出力の効率も向上する。② ビットマップで表現された動的マークビット(TMB)による中間結果の保持： TMB間の演算を規定することによってリレーションの再構成をせずに演算が完遂可能である。③ 演算の機能分散化： マクロではコード領空間と実データ空間の相互変換と実際の集合演算の機能分散が行なわれ、また集合演算においては属性に対する処理とTMBに対する処理に機能分散され、各々最大の並列性が発揮でき、かつ個々が単純な操作になるように考えられている。④ 並列・非同期処理： projectionのようなリレーション内演算及びjoinのようなリレーション間演算は本質的には並列処理の性質をもっていないがSPIRITでは前者はすべて完全な並列・非同期処理で、後者もequi-joinではセル間の交信をほとんど必要としない方式を採用している。⑤ データ駆動型のSPIRIT命令のスケジューリング： 非同期に動作している各モジュール、プロセッサ間の同期はデータフロー概念に基づいて行なわれるがこのようなデータの駆動を主体とした制御はデータベース処理に適している。

現在、SPIRITの実際の動作に即したシミュレーションを行ない、SPIRIT全体の詳細な性能評価を進めているがそれをもとにパイロットモデルを設計していきたい。

参考文献

1. Codd, E.F., "A Relational Model of Data for Large Shared Data Banks", Comm. ACM, 13, 6, (June 1970), pp.377 - 397.
2. Codd, E.F., "Relational Completeness of Data-Base Sublanguages", Courant Computer Science Symposia 6, "Data Base Systems", (May 1972), pp.65 - 98.
3. Smith, T.M. and Chang, P.Y, "Optimizing the Performance of a Relational Albegra Database Interface", Comm. ACM, 18, 10, (Oct. 1975), pp.568 - 579.
4. Shuster, E.A. etal, "RAP. 2 - An Associative Processor for Databases and ITS Aplications", (May 1975), pp.379 - 388.
   IEEE Trans. Comput., vol. C - 28, (June 1979), pp.446 - 458.
5. Ozkarahan, E.A.,and Sevik, K.C., "Analysis of Architectural Features for Enhancing the Performance of a Database Machine", ACM Transactions on Database System 2, 4, (Dec. 1977), pp.294 - 316.
6. Sadowski, R.J. and Schuster, S.A., "Exploiting Parallelism in a Relational Associative Processor", Proc. 4th Workshop on Computer Architecture for Non-Numerical Processing, (Aug. 1978), pp.99 - 109.
7. Copeland, G.P., Lisovski, G.J. and Su, S.Y.W., "The Architecture of CASSM: A Cellular System for Non-numeric Processing", Proc. First Annual Symposium on Computer Architecture, (Dec. 1973), pp.121 - 128.
8. Canady, R.H., etal, "A Back-end Computer for Database Management", Comm. ACM, 17, 10, (Oct. 1974), pp.575 - 582.
9. Banerjee,J., Baum, R.I., and Hsiao, D.K., "Concept and Capabilities of a Database Computer", ACM Transactions on Database Systems, 3, 4, (Dec. 1978), pp.575 - 582.
10. Baum, R.I., and Hsiao, D.K., "Database Computers - A Step Towards Data Utilities", IEEE Trans Compute., C-25, (Dec. 1976), pp.1254 - 1259.
11. Banerjee, J. and Hsiao, D.K., "The Use of a Database Machine for Supporting Relational Database", Proc. 4th Workshop on Computer Architecture for Non-Numerical Processing, (Aug. 1978), pp.91 - 98.
12. Copeland, G.P., "String Storage and Searching for Database Application Implementation in the INDY Backend Kernel", ibid., pp.8 - 17.
13. Allen, J.O. and Copeland, G.P., "Editing Requirements for Database Application and Their Implementation on the INDY Backend Kernel", ibid., pp.18 - 29.
14. Chamberlin, D.D., "Relational Data-Base Management Systems", ACM Computing Surveys, 8, 1, (March 1976), pp.43 - 66.
15. Hensen, P.B., "Operating System Principles", Prentice-Hall, (1973).
16. Codd, E.F., "Further Normalization of the Data Base Relational Model", Courant Computer Science Symposia 6, "Data Base Systems", (May 1971), pp.34 - 64.
17. Stonebraker, M., and Wong, E., "Access Control in a Relational Data Base Management Systems by Query Modification", Proc. ACM National Conference, (1974).
18. Stellhorn, W.H., "An Inverted File Processor for Information Retrieval", IEEE Trans. Comput. C-26., (Dec. 1977), pp.1258 - 1267.
19. Severace, D.G., and Carlis, J.V., "A Practical Approach to Selecting Record Access Paths", ACM Computing Surveys, 9, 4, (Dec. 1977), pp.259 - 273.
20. Dewitt, D.J., "DIRECT - A Multiprocessor Organization for Supporting Relational Database Management Systems", IEEE Trans. Comput., vol. C-28, (June 1979), pp.395 - 406.
21. Bobb, E., "Implementing a Relational Databases by Means of Specialized Hardware", ACM Transaction of Database System 4, 1, (March 1979), pp.1 - 29.
22. Codd, E.F., "A data base sublanguage founded on the relational calculus", Proc. 1971 ACM-SIGFIDET Workship on Data Description, Access and Control, (Nov. 1971), pp.35 - 68.
23. Su, S.Y.W., "Cellular-Logic Devices: Concepts and Applications", COMPUTER, vol.12, No.3, (March 1979), pp.11 - 25.
24. Smith, D.C.P. etal., "Relational Data Base Machines", COMPUTER, vol.12, No.3, (March 1979), pp.28 - 38.
25. Hollaar, L.A., "Text Retrieval Computers", COMPUTER, vol.12, No.3, (March 1979), pp.40 - 50.
26. Berra, P.B. etal., "The Role of Associative Arragy Processors in Data Base Machine Architecture", COMPUTER, vol.12, No.3, (March 1979), pp.53 - 61.
27. Kerr, D.S., "Data Base Machines with Large Content-Addressable Blocks and Structual Information Processors", COMPUTER, vol.12, No.3, (March 1979), pp.64 - 79.
28. 南部, 相磯, "データベースマシンのアーキテクチャ水準", 信学会電子計算機研究会資料 EC78-42
29. Kamibayashi, N., etal, "SPIRIT - A New Relational Database Computer Employing Functional-Distributed Multi-microprocessor Configuration", Proc. 1st International Conference on Distributed Computing Systems (to be published)