データベース・システム 84−22
（１９９１．７．１８）

並列推論マシン上の並列データベース管理システム

河村 元夫
(財) 新世代コンピュータ技術開発機構

並列推論マシン PIM 上で動作する並列データベース管理システム Kappa-P について述べる。このシステムは，データモデルとして非正規関係モデルを採用している．機能的には、拡張関係代数を中心にした言語処理系、および並列処理のための分散データベース機能が拡充されている。処理面としては、KL1 に適したプロセス指向の設計、および言語の各階層に対応した各層での並列処理が特徴である。

A Parallel Database Management System on a Parallel Inference Machine

Moto Kawamura
Institute for New Generation Computer Technology
Mita Kokusai Bldg. 21F, 4-28, Mita 1-chome, Minato-ku, Tokyo 108, JAPAN

The Kappa (Knowledge Application-oriented Advanced Database and Knowledge Base Management System) project is one of the knowledge base management software projects at ICOT. We are now involved in the design and development of a parallel database management system (Kappa-P) to run on a parallel inference machine (PIM) and on its operating system (PIMOS). Kappa-P is based on a nested relational model and the system consists of language processors including extended relational algebra and distributed DBMS for parallel processing, mainly. I describe an outline of Kappa-P system.

1 はじめに

Kappa (Knowledge APPlication-oriented Advanced Database and Knowledge Base Management System) プロジェクト [1] は, ICOT の知識ベース管理ソフトウエア (KBMS) プロジェクトの一つであり, 知識情報処理システム (KIPS) の中核的機能であるデータベース, 知識ベース管理機能を提供することを目的としている. 中期には, 逐次推論マシンとそのオペレーティング システム PSI/SIMPOS の環境で動作する試作版 Kappa-I とその改良 / 機能強化版 Kappa-II を開発し, 国内外の研究機関に広くリリースしている. 後期である現在は, 並列推論マシンとそのオペレーティング システム PIM/PIMOS の環境で動作するデータベース管理システム Kappa-P の試作をおこなっており, 演繹オブジェクト指向データベース システム *Quixote* や種々の並列 KIPS へのデータベース機能の提供をおこなう.

逐次版の Kappa-II は, 自然言語処理のための辞書や, ゲノムや蛋白質などの分子生物学データベースなどの新しい分野で利用され, その有効性が確かめられている. ゲノムや蛋白質などのデータは, 解析技術の進歩により爆発的に増加しており, そのデータ解析や格納には並列マシンのパワーが必要で, Kappa-P の重要な応用と位置付けている.

Kappa-P は, 並列マシン上の DBMS であり応用プログラムと同じマシンで動作することを前提に設計している. このようなシステムは他に MCC の Bubba[2] や PHILIPS の PRISMA などがある.

以下、2節で Kappa プロジェクトの概要, 3節で動作環境, 4節で特徴, 5節で問い合せ処理について述べる.

2 Kappa プロジェクトの概要

Kappa プロジェクトでは, 知識ベース管理はデータベース管理機能を含むべきであり, それをデータベース管理の拡張として位置づけている. そしてそのシステムの実現にあたってはデータベース層, 知識ベース層, ユーザインタフェース層の各層から構成することとしている.

2.1 研究開発状況

Kappa プロジェクトの研究開発経過を以下に示す:

- Kappa-I [1985.9 − 1987.8]
  データベース層の試作実験に主眼を置いたシステムで, PSI 上に ESP で実装した. 特徴は, 非正規関係モデルの採用, データ型としての項やネットワーク型構造データのサポートなどである. そして電子化辞書などのデータを試験データとして格納し効率的に動作することを確認した.

- Kappa-II [1987.4 − 1989.3]
  Kappa-I の経験をもとに, システムの三つの層全体にわたる研究開発が本格化した. データベース層では, システムの効率化と実用性を重視した拡張をおこなった. すなわち, CPU 資源と主記憶資源の一層の効率的使用, PSI の大規模主記憶を活用した主記憶データベース機能の付加と, 知識ベース層での非正規関係に対する意味論の反映, 拡張関係代数インタフェースの付加, ユーザ定義コマンドの提供などである. また知識ベース層では, 非正規関係のための論理型言語 CRL の設計と, 確定節と CRL に基づく演繹データベースの検討と試作をおこなった. ユーザインタフェース層では, 非正規関係に対する端末インタフェースや構造エディタを利用したメタデータ保守ツールの開発をおこった. Kappa-II のデータベース層とユーザ インタフェース層は, PSI 上のツールとしてリリースされており, 国内外の研究機関で利用されている.

- Kappa-P [1989.2 − 1992.3]
  PIM/PIMOS の並列推論マシンの環境で動作するデータベース管理システムを研究試作し、PIM 上で動作する並列 KIPS に対しデータベース管理機能を提供する. Kappa-I,II でその有効性が確認された非正規関係モデルをデータモデルとして採用している. 機能的には、拡張関係代数を中心にした言語処理系、および並列処理のための分散データベース機能が拡充されている。処理面としては、KL1 に適したプロセス指向の設計、および言語の各階層に対応した各層での並列処理が特徴である。

3 動作環境

並列データベース管理システム Kappa-P は, 並列推論マシン PIM とそのオペレーティング システム PIMOS の環境で動作する。また、実装言語は, 並列言語 KL1 である. この環境では, DBMS と応用プログラムは混在して実行される.

- 並列推論マシン PIM
  PIM は MIMD 型の並列マシンあり, いくつかの種類のものが作られるが, 基本的には図1のような構成である. クラスタ間はネットワークによる疎結合であり, クラスタは、10 台程度の要素プロセッサ (PE) が、共有バス / 共有メモリにより結合されている。共有メモリは, 数百メガ バイトで, ディスクは各クラスタに取り付けることができる.

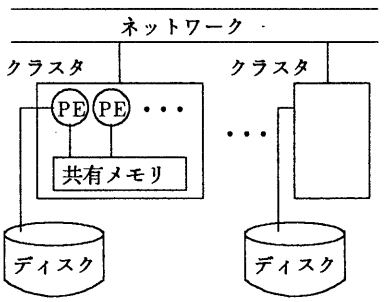

図 1: PIM の構成

- オペレーティング システム PIMOS
  独立したひととおりの機能をそろえたオペレーティング システムで, 並列ハードウエアを一つに見せる集中型の単一オペレーティング システムである.

- 並列言語 KL1
  並行論理型言語 GHC をもとに, オペレーティング システム記述能力を持たせたものである. 並列処理の粒度は小さく, 述語単位の AND 並列である. 並列言語としてみた場合, 単純でかつ高機能である. 同期はデータの依存関係によってのみ取られるため, それ以外の要因を考える必要がない. データが存在する位置によってプログラムが変わることもない. 多様なプログラミング スタイルを使って書くことができるが、一般にプロセス指向プログラミングをおこなう. また, 論理変数は単一代入性を持つため破壊的代入が記述できない (たとえば, 配列データの更新は新たな配列が作られる) が, 簡素化した効率的な参照カウント方式により, 効率を落とさずに破壊的代入に相当することが記述できる.

4 特徴

並列データベース管理システムとしての特徴を述べる.

- 非正規関係モデルの採用

  基盤となるデータモデルとして採用した非正規関係モデルの意味論として, 非正規関係に固有の操作である行ネスト操作に対し独立した意味を与えている. つまり, 非正規関係 $\{[a/c_1, b/\{c_2, c_3\}], [a/c_1, b/\{c_3, c_4\}]\}$ と $\{[a/c_1, b/\{c_2, c_3, c_4\}]\}$ は同じ意味を持っている. これにより, ネスト構造を意識しない問合せが可能となる. 先の非正規関係に対し問合せ $? - [a/X, b/\{c_2, c_4\}]$ を出すと, どちらも $X = c_1$ が結果として得られる.

これは，意味論としては基本的に関係に基づきながら表現(および蓄積構造)の効率化をめざしているという点で，関係モデルの自然な拡張であり，多値従属性をもつデータの表現と処理の効率化が可能である．この意味を反映して，関係代数を，行，列のネスト，アンネスト操作を含む拡張関係代数として定義しなおした．

- データベース管理システムとしての構成

  Kappa-P は、疎結合を考慮した並列処理と密結合を考慮した並列処理の組み合わせになっている．

  疎結合を考慮した並列処理とは、独立したデータベース管理システム(ローカル DBMS)が複数集まって一つのデータベース管理システムを構成し、問い合わせはそれらのローカル DBMS が協調することでおこなわれる、ということである．密結合を考慮した並列処理とは，構成要素であるローカル DBMS それぞれを別々のクラスタに割り当て，そのの内部処理を密結合向きの並列処理でおこなうということである。

- 主記憶データベースの提供

  PIM は各クラスタは，数百メガ バイトの大容量主記憶を持つ．この大容量主記憶を使って，主記憶データベースをサポートする．この主記憶データベースでは二次記憶へ反映はおこなわない．しかし，多数の中間結果テーブルを生成するシステム，たとえば演繹データベースなどでは，これで十分である．また，主記憶データベースと二次記憶データベースを組み合わせて，ログ付き主記憶テーブルも提供する．主記憶データベースと二次記憶データベースの両方に同じ内容のテーブルを作っておき，読み系問い合わせは主記憶データベースに出し，更新系問い合わせは主記憶データベースと二次記憶データベースの両方に同じものを出す．これにより読み系問い合わせは主記憶データベースの性能でて，更新系問い合わせは並列処理できるので二次記憶データベースの性能ぐらいでおこなうことができる．

- データの配置

  非正規関係のローカル DBMS への配置は、データベース生成時にユーザが決めることにした．巨大なデータや複数のローカル DBMS に分けることにより高い並列性が得られるようなデータのために、複数のテーブルを仮想的に一つに見せるような機構をもつ水平分割テーブルなどもサポートする．

- 大域情報の管理

  非正規関係の名前はそれが格納されているローカル DBMS とはまったく無関係とすることにしたため，非正規関係の名前を次のように管理することにした．

  非正規関係の名前とそれが格納されているローカル DBMS の対応が必要になるのは、問い合わせ変換時の初期だけで、重い処理ではないので、集中管理することにした。すなわち、サーバ DBMS と呼ぶ DBMS でシステム全体の非正規関係の名前を保持し名前の一意性検査や名前とそれが存在するローカル DBMS の対応づけをおこなう．たとえこの情報が壊れたとしても，ローカル DBMS が局所的に管理しているメタデータを集めることにより復元可能であるが，ただ一つのサーバ DBMS が存在するだけだと可用性が高いとはいえない．ある程度可用性を高める目的で，複数のサーバ DBMS により大域名前情報の複製をおこなう。複製には、Voting に基づく手法を利用する。

- 問い合わせ処理

  問い合わせは、利用するテーブルの所在とそれに対する操作、分割した時に発生する通信量やローカル DBMS の負荷を考慮して分割される．詳しくは，5節で述べる．

- 疎結合向き並列処理

  疎結合向き並列処理では、複数のローカル DBMS が協調することで、問い合わせ処理を行うことである。そのために次のような処理を行う。

- 分散トランザクション

  複数のローカルDBMSによる分散処理により、複数の対象(主に二次記憶)に更新が発生するため、分散トランザクションをサポートする(プロトコルは二相コミット).

- ローカルDBMS間の通信

  ローカルDBMS間の通信では、非正規関係の複雑な構造のデータを流す必要がある. KL1ではデータの存在場所を意識することなくデータをアクセスすることができるが、複雑な構造のデータを大量にアクセスすることには不向きである. そのため、ローカルDBMS間の通信データを、バッファリングし一括転送向きにコード化しデータの内容ごと転送することにした. 転送コマンドは、問い合わせの分割(変換)時に、問い合わせに埋め込まれる. 問い合わせを解析することによりデータのうち不要な部分がわかるので、それを除くためのコマンドも同時に埋め込まれる。

- 密結合向き並列処理

  ローカルDBMSの内部では、データの所在を考慮しない処理が中心であり、一つのローカルDBMSは、主に一つのクラスタ内でのみ実行されると仮定している。

  - レコード ストリームによる並列処理

    問い合わせを、演算をノード、その間の依存関係をアークとしたグラフとしてみて、演算ノードをプロセス、アークを演算対象であるレコードを流すストリームとすることで、並列処理をおこなう。中間結果を少なくするために、レコードは要求駆動でストリームに流される。

  - 基本的な演算の並列処理

    集合演算、インデックス操作などで共有メモリーを仮定した演算を行う。

5 問い合せ処理

5.1 言語の階層

次の三階層の言語により問い合わせを処理する.

- ユーザ記述言語

  基本的には、非正規関係のために拡張した拡張関係代数からなる式の集まりである. 演繹データベースで必要な推移閉包を効率的に処理するためにループも記述できるようにしてある.

- 中間言語

  拡張関係代数の処理のための中間言語であり, 拡張関係代数の処理をさらに細分化したものである.

- 原始コマンド

  PSI上のDBMS Kappa-IIのPrimitive Commandに対応するもので, 一つのテーブルに対するレコード ポインターに基づく処理が中心である. Kappa-IIのコマンドとの違いは, 非正規関係の操作は機能的にサブセットであること, レコード ストリームを導入したこと, 分散トランザクションをサポートしていることなどである.

ユーザ記述言語で書かれた問い合わせは、一つのトランザクション単位でまとめて受けとられる. そして, 最適化, 中間言語への変換, ローカルDBMS間データ転送コマンドの埋め込み, 演算順序制御コマンドの埋め込み, 分割などをおこない, 最終的に中間言語処理プロセスに中間言語コマンドをメッセージとして送るKL1プログラムに変換され, 該当するローカルDBMSに送られる. KL1プログラムを受けとったローカルDBMSは, それに対応するトランザクションの下で実行する. すると, 中間言語処理プロセスに対し中間言語コマンドが実行すべき順序で送られてくる. 中間言語処理プロセスが中間言語コマンドを処理する過程で, 原始コマンドが出される.

5.2 問い合せ処理

問い合せ処理は, 大きく分けると次の二つに分けられる。

- 問い合わせ変換

  ユーザ記述言語で書かれた問い合わせを, 中間言語処理プロセスに中間言語コマンドをメッセージとして送る KL1 プログラムに変換する部分で, インタフェース プロセスがこれをおこなう. 変換の際に, テーブルの位置情報を取り出すためにサーバ DBMS をアクセスする. そして、テーブルが存在するローカル DBMS からテーブルのスキーマと格納データに対する補助情報を取り出し, その情報を基に, 最適化, 中間言語への変換, ローカル DBMS 間データ転送コマンドの埋め込み, 演算順序制御コマンドの埋め込み, 分割などがおこなわれる.

- 問い合わせ実行

  インタフェース プロセスが変換した中間言語処理プロセスに中間言語コマンドをメッセージとして送る KL1 プログラムを受けとり、それを分散トランザクションをサポートするトランザクション プロセスの管理下で実行する. 中間言語の実行では, レコード ストリームによる並列処理がなされる. テーブルの生成 / 削除時にはサーバ DBMS をアクセスする. 他ローカル DBMS 上のテーブルへのアクセスは, 変換時に埋め込まれた中間言語の転送コマンドによってなされる. すべての問い合わせ処理が終了すると分散トランザクションのプロトコル (二相コミットを採用) に従ってトランザクションを終了させるが、このときに他ローカル DBMS のトランザクション プロセスと通信を行う.

5.3 問い合わせ変換

インタフェース プロセスが, ユーザ記述言語で書かれた問い合わせを, 中間言語処理プロセスに中間言語コマンドをメッセージとして送る KL1 プログラムに変換する. 図 2が問い合わせ変換でおこなうことである。

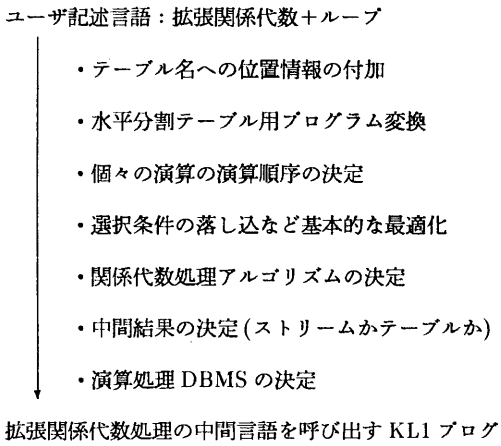

図 2: 問い合せの変換

1. サーバ DBMS をアクセスしテーブルの位置情報を取り出す
2. テーブルが存在するローカル DBMS からテーブルのスキーマと格納データに対する補助情報を取り出す。この補助情報には、次のようなものがある.

- 属性に対する索引の有無: 索引の有無で, 演算に使用できるアルゴリズムが決まる.
- 属性値の唯一性: 値がユニークであるかどうかで、演算時の属性の選択に利用.
- 属性値の種類, 属性値の数: 演算結果の量の推定に利用.
- レコードの数とその大きさ: 演算結果の量の推定に利用.

3. 水平分割テーブルをテーブルの実体に置き換える.

4. 個々の演算の演算順序を決定し、演算間に半順序関係をつける. 更新ではない演算は順序を付けずに動かすことができるが、更新操作が入ってくると、更新前 / 後で結果が異なるので、更新操作はそれより前に終わるべき全ての演算が終了したら開始でき、更新操作後に開始すべき演算は更新処理が終了したら開始できる. ここで、決定された順序は, 演算順序制御コマンドとして埋め込まれる.

5. テーブルの情報を基に, 選択条件の落し込などの基本的な最適化をおこない, 中間結果をレコード ストリームにするかテーブルにするか決定し, 索引の有無などから関係代数処理アルゴリズム (マージ法かハッシュ法かなど) を決め中間言語コマンドへの変換する. ここでは, 複数の候補が得られる。

6. 上で求まった複数の候補に対し、テーブルの位置情報を基に分割 (演算を実行するローカル DBMS を決める) する。そして、データ転送量を予測し最も小さいものを選択し, 通信路にデータ転送コマンドの埋め込む。
   最終的には、演算の重さを考慮し大雑把な時間軸を入れ処理時間の最小値をとるようにしたい.

7. 演算順序制御コマンドの埋め込みなどをおこない KL1 プログラムとしての体裁を整える.

図 3はユーザ記述の例で, 図 4はそれを変換した KL1 プログラムである. 記述が繁雑になるので, 一つのローカル DBMS で実行される場合のみに限定し, 中間言語への変換はせず拡張関係代数のまま載せてある。実際には、これにさらに他ローカル DBMS への通信路が引数として引き回される.

```
go(Result::result1, Temp::result2) :-
      selection(table2, ``(from = "icot"), Temp),
      difference(table1, table1, EmptyTable),
      transitive_closure(table1, EmptyTable, table1, Result),
      replace(table2, ``(a = a + 1 where a > 10)).

transitive_closure(Delta, In, R, Out) :- empty(Delta) |
      In = Out.

transitive_closure(Delta, In, R, Out) :- true |
      join(In, In, `(to = from), In1),
      projection(In1, {'1.from', '2.to'}, In2::(from, to)),
      union(In2, R, NextIn),
      difference(NextIn, In, Delta),
      transitive_closure(Delta, NextIn, R, Out).
```

図 3: 問い合わせ (ユーザ記述)

6 まとめ

並列データベース管理システム Kappa-P の特徴と問い合わせ処理の並列処理について述べた. 現在, 原始コマンドが動きつつあり, 今年度中に分子生物学データベースの格納実験や評価をおこなう予定である.

参考文献

[1] K. Yokota, M. Kawamura and A. Kanaegami, "Overview of the Knowledge Base Management System (Kappa)", *FGCS*, 1988

[2] H. Boral, W. Alexander, L. Clay, G. Copeland, S. Danforth, M. Franklin, B. Hart, M. Smith and P. Valduriez, "Prototyping Bubba, A Highly Parallel Database System", IEEE Transactions on Knowledge and Data Engineering, Vol. 2, No. 1,March 1990

```
go(ILP, {Table2, Table1, Param_Table_for_tc}, Status) :-
      true |
      ILP = {ILP1, ILP2, ILP3, ILP4},
      create(ILC1, UDC),
      Table2 = {Table2_1, Table2_2},
      Table1 = {Table1_1, Table1_2, Table1_3, Table1_4},
      ILC1 = [selection(Table2_1, ``('1' = ``icot''), Temp, Status1),
              difference(Table1_1, Table1_2, EmptyTable, Status2)],
      UDC = [transitive_closure(Table1_3, EmptyTable, Table1_4, Result,
                                Param_Table_for_tc, Status3),
      ilp:wait([Status1], ILC2,
              [replace(Table2_2,``('2' = '2' + 1 where '2' > 10), Status4)]),
      ilp:wait([Status3], ILC3, [modify_schema(Result, result1, Status5)]),
      ilp:wait([Status1], ILC4, [modify_schema(Temp, result2, Status6)]),
      ilp:wait([Status5], Result, []),
      ilp:wait([Status6], Temp, []),
      lib:error_check([Status1,Status2,Status3,Status4,Status5,Status6],
                      go, Status).

create(ILP, UDCin1) :- true |
      merge({UDCin1,UDCin2}, UDC),
      top_loop(UDC, UDCin2, ILP).

top_loop([transitive_closure(Delta,In,R,Out,Param_Table_for_tc,Status)
          | UDC], UDCin, ILP) :-
      true |
      ILP = {ILP1, ILP2},
      UDCin = {UDCin1, UDCin2},
      transitive_closure(Delta,In,R,Out,Param_Table_for_tc,ILP1,UDCin1,Status),
      top_loop(UDC, UDCin2, ILP2).

top_loop([], UDCin, ILP) :- true | UDCin = [], ILP = [].

transitive_closure(Delta, In, R, Out, Param_Table_for_tc,
                   ILP, UDC, Status) :- true |
      ILP = {ILP1, ILP2},
      Delta = {Delta1, Delta2},
      ILP1 = [get_cardinality(Delta1, Cardinarity, Status1)],
      transitive_closure_01(Cardinarity, Delta2, In, R, Out, Param_Table_for_tc,
                            ILP2, UDC, Status2),
      lib:error_check([Status1,Status2], transitive_closure, Status).

transitive_closure_01(0, Delta, In, R, Out, Param_Table_for_tc,
                   ILP, UDC, Status) :- true |

      In = Out, Delta = [], R = [], ILP = [], UDC = [], Status = normal.

otherwise.
transitive_closure_01(_, Delta, In, R, Out, {Iformat1},
                   ILP, UDC, Status) :- true |
      ILP = {ILP1, ILP2, ILP3, ILP4, ILP5},
      In = {In_1, In_2, In_3},
      R = {R_1, R_2},
      ILP1 = [join(In_1, In_2, `(`2` = `1`), In1, Status1),
              projection(In1, Iformat1, In2, Status2),
              union(In2, R_1, NextIn, Status3),
              difference(NextIn, In_3, Delta, Status4)],
      UDC = [transitive_closure(Delta, NextIn, R_2, Out, {Iformat1}, Status5)],
      lib:error_check([Status1, Status2, Status3, Status4, Status5],
                      transitive_closure_0102, Status).
```

図 4: 変換後のプログラム