計算機アーキテクチャ 73－2
（1988. 10. 31, 11. 1）

三次元集積回路を用いた高速並列プロダクションシステム

藤田 聡　相原 玲二　山下 雅史　阿江 忠

広島大学 工学部

本報告では、知識処理における推論メカニズムのひとつであるプロダクションシステムに注目し、その高速並列実行方式の提案を行なう。ここでは対象をルールの並列実行が可能なクラスのプロダクションシステムに限定し、文法的に依存関係のある複数のルールを並列に実行させるためのいくつかの競合解消方式について検討する。つぎにこれら各競合解消方式が、三次元集積回路技術を用いて効率的に実現されることを示す。これは、ルールを並列に実行する際に必要となるプロセッサ間の密な通信が、三次元集積回路を用いることによって少ないオーバヘッドで実現可能なためである。三次元集積回路は現在までのところ試作段階の域を出ないが、近い将来実用化が期待されている実装技術の一つであり、本学集積化システム研究センターにおいても三次元共有メモリの設計,製造が進められている。また、提案する並列プロダクションシステムにおける高速化の効果は、シミュレーションにより評価される。

A HIGH SPEED EXECUTION OF PARALLEL PRODUCTION SYSTEMS USING THREE-DIMENSIONAL INTEGRATED CIRCUITS

Satoshi FUJITA, Reiji AIBARA, Masafumi YAMASHITA and Tadashi AE

Faculty of Engineering, Hiroshima University
Saijo-cho, Higashi-Hiroshima-shi, 724 Japan

This paper proposes mechanisms for high speed execution of parallel production systems, in which several rule instances can be fired in parallel. Firstly, we concentrate on conflict resolution models to support parallel firing of rule instances which might be dependent to each other. The models can be efficiently implemented by using three-dimensional integrated circuits technology, since the tight communication among processors required for the parallel firing can be realized without much communication overhead by the technology. The effect of the proposed mechanism is estimated by simulation.

1. はじめに

プロダクションシステム(以下PS)は、断片的知識を自然に表現するという点で注目を集めている推論メカニズムの一つである。さらに実行方式の簡明さなどから、プラント制御や各種診断システムなど多くの分野への適用が進められている。しかしながら、ルール記述そのものが平板的であることやその実行が基本的にデータベースサーチに基づくものであることなどから、ルールベースやデータベースの大規模化に伴う実行時間の増大が、実用上の大きな問題となってきている。

この問題を逐次的な処理によって解決するには限界があることから、並列処理によってこの問題を解決しようとする試みが数多く行なわれてはいるが、その多くはマッチング処理 (条件部を満足するルールインスタンスの生成処理) の高速化のみをねらったものである[4-7]。さらにその議論には、Reteアルゴリズム[1]や TREATアルゴリズム[2]など特定のマッチングアルゴリズムの範囲に限られたものが多い。これらは、 OPS83[3]のような特定の言語の処理系を高速化するという立場からは有意義なアプローチであるが、 PSの並列実行という観点からはより一般的でかつより掘り下げた議論が必要になってくると思われる。本研究の目的の一つは、計算モデルとしてのPSに注目しその並列実行のための並列計算モデルを確立することにある。しかし実用的見地から見ればモデルのみでなくその実現方法の検討もまた重要な課題であり、しかも理想化された並列計算モデルの実現は、一般には技術的制約によって制限を受けることが多い。

一方、並列処理システムを構築するための構成要素技術も、ここ数年飛躍的に進歩しつつある。中でも三次元集積回路技術は、単に1チップあたりの集積度を向上させるというだけでなく、非常に高い能力を持ったシステム構成要素を実現するという点において、大きな期待が持たれている技術の一つである。我々はこれまでにいくつかの三次元集積回路向きアーキテクチャの提案を行なってきた[8-10]。さらに現在本学集積化システム研究センターにおいては、三次元共有メモリ[11]の実現が具体的な設計段階に入っている。本研究のもう一つの目的は、三次元集積回路(特に三次元共有メモリ)の有効な使用方法を提示し、その実用化を推進していくことである。

本稿では、三次元集積回路技術を用いた並列PS実現法の提案を行なう。以下では2.で、並列PSにおいて考慮すべき事項の検討を行なう。その中でも特に複数ルールを並列に実行させるための競合解消方式に注目し、同期型,非同期型の二通りの実行モデルを示す。3.,4.ではそのそれぞれについて実現上の問題点を指摘し、三次元集積回路を用いた各問題点の解決法を提案する。各方式の効果は、シミュレーションによって示される。

2. 実行モデル

2.1. OPS型逐次実行モデル

PSのモデルとして現在最も一般的なものは次のようなものである (以下これをOPS型モデルと呼ぶ)。このモデルは基本構成要素としてルールベース,データベース(いずれも有限サイズ)および推論エンジンを持ち、以下の三つのフェーズからなるサイクルの繰り返しによって逐次的に計算(すなわち前向き推論)を行なう。ここでルールベース中の各ルールは条件部と行動部から成っており、それぞれ、 "現在のデータベースにおいてそのルールの条件部が満たされるならば、そのルールの行動部の実行によってデータベースの変更を行なってもよい" ことを表している。

[OPS型モデルにおける計算]

(1)データベースとルールベースを参照することで、実行可能なルールを生成する。(マッチングフェーズ)

(2)実行可能なルール集合の中から、適当なものを一つ選択する。(競合解消フェーズ)

(3)選択されたルールの行動部を実行しデータベースの変更を行なう。(実行フェーズ)

ルールの表現法としてはいろいろな方式が考えられるが、通常は実用性を考慮して変数の使用を許している。この場合、マッチングフェーズで生成されるのは (有限サイズの)データベースによって具現化(instantiate)された (有限サイズの)ルールインスタンス集合である。以下、この集合を競合集合(conflict set)と呼び、ルールインスタンスを単にインスタンスなどと呼ぶ。

このモデルは、その後のOPS5,OPS83などの基本的な枠組みとなっており、前述したように、PSの高速化・並列化のための数々の試みがこのモデルの範囲内でなされている[4-7]。

2.2. インスタンス間の依存関係

これに対し、PSのより一層の高速化をめざしてOPS型モデルを拡張し、複数のインスタンスを並列に実行させるための試みも行なわれている[12]。ところが一般にインスタンス間は独立ではないことから、複数のインスタンスを並列に実行するためにはその間の依存関係を考慮

する必要がある。このようなインスタンス間の依存関係には、大きく分けて意味的(semantic)なものと文法的(syntactic)なものがある。

インスタンス間の意味的な依存関係とは、インスタンス間で暗に仮定されている実行順序のことであり、具体的には、特定の競合解消戦略を意識したルール記述に起因するものである。複数ルールの並列実行を可能にするためには、この依存関係に(並列実行のための)ある程度の自由度がなくてはならない。しかしながら現在までのところ PSのための明快な意味論は展開されておらず、そのような状況下で意味的な依存関係を明示的に表現することは、 PS本来の良さである非決定性やルールの独立性を失わせることに成りかねない。したがって本稿では、ルール間に意味的な依存関係が全く存在しないPSを想定し、その上でインスタンスの並列実行を考えることにする。

ところがこのような制限されたPSにおいても、インスタンス間には文法的な依存関係、すなわちあるインスタンスの実行が別のインスタンスの実行可能性に影響を与えるような関係が存在する。文法的に依存関係のある二つ以上のインスタンスを同時に実行した場合、逐次的な実行ではありえない状態にデータベースが到達する可能性がある。以下ではこのような状態を(逐次実行に対して)"矛盾している"と呼び、矛盾の起こらない範囲に限定してインスタンスの並列実行を考えていくことにする。このことは他の書き換え系を並列実行する場合の制約と同様であり、並列実行に対する自然な要請であるといえる。 (以降では、二つのルールのインスタンス間に文法的な依存関係がありうるとき、この二つのルール間に文法的な依存関係があると呼ぶ。) ルール間に文法的な依存関係のあるPS上で複数のルールインスタンスを並列に実行するためには、その文法的な依存関係を動的に検出しかつそれをインスタンスの実行可能性に反映させるための高機能な競合解消メカニズムが必要となる。

2.3. 並列実行モデル

以下の各節では、ルール間に意味的な依存関係のない任意のPSの並列実行のためのアーキテクチャについて検討する。具体的には並列発火のための競合解消の実現方法に注目し、以下のような二通りの並列実行モデルに分けて議論していく。

(a) 競合解消を集中的に(かつ同期的に)行なう方法。
この方法では、大域的な依存関係の検出とインスタンスの実行可能性のチェックが可能である。(以下、これを同期型並列PS、**SPPS** (Synchronized Parallel PS)と呼ぶ。)

(b) 競合解消を各プロセスで非同期的に行なう方法。
したがって系における矛盾の発生を動的に検出し、系を無矛盾な状態に復帰させるためのメカニズムが必要である。(以下、これを非同期型並列PS、 **APPS** (Asynchronized Parallel PS)と呼ぶ。)

(a)(b)ともルールの並列実行による高速化の可能性はあるが、その半面、系の無矛盾性の保持に要する時間のためにかえって処理速度が低下する可能性も持っている。したがってこのような並列PSを実用的なものにするためには、如何に高速に文法的な依存関係を検出しかつ効率よくルールの並列実行を行なうかが鍵となる。

以下の各節では、 (a)(b)それぞれの立場から並列PSアーキテクチャを考察し、三次元集積回路を用いた一実現法の提案およびその評価を行なう。

3. 同期型並列プロダクションシステム (SPPS)

本節ではSPPSの実現に際して考慮すべき問題点を指摘した後、三次元共有メモリを用いた一実現法の提案を行なう。提案する実現方式の有効性は、簡単なPSプログラムのシミュレーションにより評価される。

3.1. 実現における問題点

PSにおけるルール間の文法的な依存関係の検出方法が Ishidaら[7]によって示されている。この手法は、PSのデータベースがデータタイプによって分類(クラスタリング)できることを利用しており、分類されたクラスタ単位でルール間の依存関係を静的に検出するというものである。競合集合中の各ルールインスタンスはいずれかのルールの具現化であるから、ルール間の依存関係を静的に検出し競合集合中のインスタンスをルールごとに管理することで、インスタンス間の依存関係を実行時に効率的に検出することができる。 (なお[12]の文献の考察は、静的なルール間の依存関係の検出に留まっており、依存関係の動的な評価については、その可能性に触れているだけである。) SPPSではこのようにして、ルールの並列実行に伴う(逐次実行に対する)矛盾をあらかじめ除外することができる。

この方式を成功させるためのキーポイントとなるのは、以下の二点である。

(1) 競合解消フェーズにおけるインスタンス間の依存関係の検出を如何に効率よく行なうか。(SPPSでは競合集合

は唯一であるから、たとえば競合解消を複数プロセッサで並列に行なうことで、処理の高速化を図ることができる。)

(2) マッチングフェーズにおける並列処理を如何に効率よく実現するか。(この方向は、OPS型モデルの範囲内での従来からの高速化のアプローチ[4-7]と同様である。具体的には、各プロセスによって共有されるデータベースをどのようにして物理的にインプリメントするかが問題となる。)

以下では、三次元共有メモリ[11]を用いたSPPS構成法の提案を行ない、その有効性をシミュレーションによって評価する。

3.2. 三次元集積回路を用いた実現法

三次元共有メモリは構造的には多層化メモリであり、メモリ平面間のデータ転送を層間結合を用いて高速に行なうことができる(図1)。またこのメモリによりほぼ理想に近い共有メモリが実現されるであろうことが期待されている。メモリの各層に接続されたプロセッサがメモリに書き込んだデータは、メモリ内部の層間転送メカニズム[11]により、数マシンサイクル内に全層の同一アドレスに書き込まれる。

さて我々はすでに、三次元共有メモリと内容アドレス可能メモリ(CAM)を併用した高速マッチングプロセッサの提案を行なっている[10]。その概要は、以下のようなものである(図2)。まずデータベースをデータタイプによってクラスタリングし、そのそれぞれを各プロセッサのローカルメモリにマッピングする。マッチング中のデータベースへのアクセスは、本質的に、データのいくつかのフィールドの値をキーとする連想アクセスであるから、

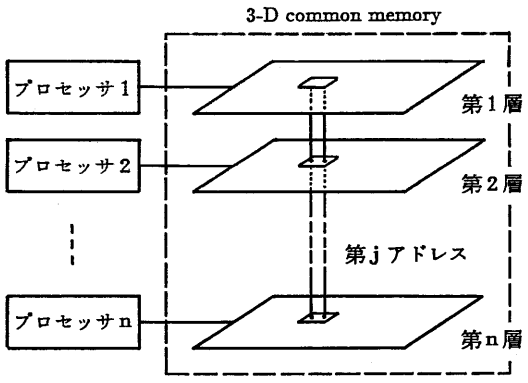


図1　三次元共有メモリ

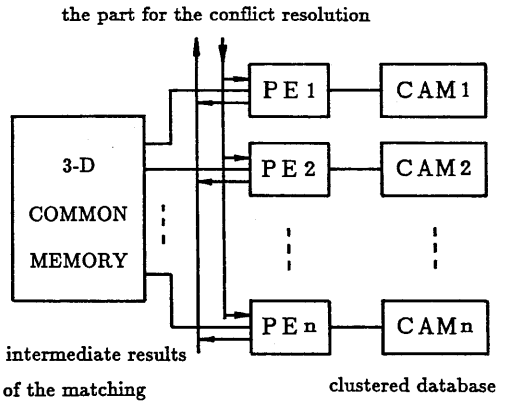


図2　SPPSにおける高速マッチングプロセッサ
(各PEにはデータベースクラスタを割り当てる)

それぞれのローカルメモリをCAMによって実現することで高速なデータアクセスが実現される。一方PSにおけるマッチング処理は、いくつかの条件要素の積であるルールの条件部を満足するデータの組を検出する処理であり、したがってこの処理はストリーム並列による効率的な実現が可能である。ところが実際問題として、プロセッサの稼働率を向上させしかも CAMによって得られた高速サーチの効果を低下させることなくストリーム並列処理を行なうためには、非常に高速でしかも通信オーバヘッドの少ない通信媒体が必要となる。我々の基本的なアイデアは、その媒体(グローバルメモリ)を三次元共有メモリで実現しようというものである。

しかしこの方式には、メモリの実現可能性の他に以下のような問題点が残っている。

(1) メモリが高価であること。(現在の技術では、CAM,三次元共有メモリとも SRAMより一桁以上低い集積度となる。)

(2) グローバルメモリを三次元共有メモリで効率的に実現するためには、ルールの記述法に制約を付けなくてはならないこと[10]。(これは三次元共有メモリが立方格子状に構成されていることによる制約である。)

(3) マッチングの順序に関する最適スケジューリングが困難であること。(この問題は、NP完全な問題のクラスに属することがわかっている。)

次にSPPSにおける競合解消のためのアーキテクチャを示す(図3)。この実現法ではまず、ルール集合をあらかじめいくつかに分割し、各部分ルール集合間に強制的に全順序関係をつけておく。(前述したように本稿では意味的な依存関係のないPSを想定しているから、この順序づ

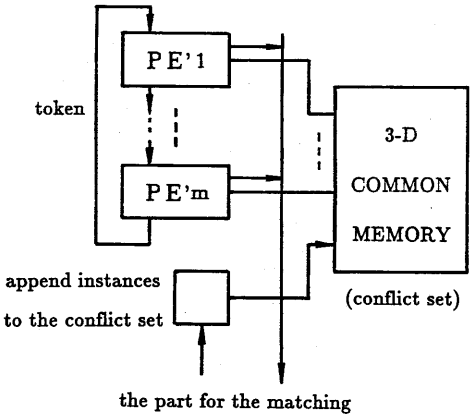


図3　SPPSにおける競合解消プロセッサ
（各PEには部分ルール集合を割り当てる）

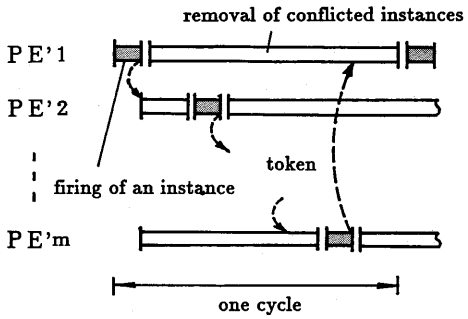


図4　競合解消のタイムチャート

けによって推論の正当性が失われることはない。）また各プロセッサには、この部分ルール集合を静的に割り当てておく。競合解消時におけるインスタンスの選択は、優先度の高い部分ルール集合から順に一つずつ行なっていく。（この処理はトークンを受け渡していくことで実現する。）各プロセッサは、自分の管理するルール集合より優先度の高いルール集合の発火済みインスタンスと競合しないように、自分の管理するルール集合のインスタンスから一つを選択すればよい。（タイムチャートを図4に示す。この際、静的に得られるルール間の文法的な依存関係を利用することができる。）さらに各プロセッサは、他のプロセッサがどのルールを発火させたのかを、三次元共有メモリを介して知ることができる。

この方式で性能を左右するのは負荷分散問題、すなわちルール集合をプロセッサにどのように割り当てるかという問題である。現在、そのためのヒューリスティックアルゴリズムの検討を行なっている。

3.3. 評価

三次元共有メモリを用いた高速マッチングプロセッサの効果を、シミュレーションにより評価した。評価に用いたプログラムは以下の二つである。（ここでの目的はマッチングフェーズの高速実行であるから、必ずしもルール間に意味的な依存関係のないものを用いているわけではないことに注意。）

(a)　農夫のジレンマの問題----川の対岸へ渡す狐・鵞鳥・トウモロコシの数をそれぞれn(オリジナルは1)に拡張したもの。(ルール数:5 クラスタ数:7)

(b) 猿とバナナの問題---部屋の中に問題の目的と関係ないn個のオブジェクトを散乱させ、マッチングの時間を意図的に増減できるように初期状態を拡張したもの。(ルール数:20 クラスタ数:22)

図5および6にシミュレーション結果を示す。図5には、CAMの使用によるサーチステップ数の減少の様子が示されている。図には、問題(a)の解を求めるまでに実行されたプロダクションサイクル数に対して合計何ステップのデータサーチが行なわれたかを、 RAMとCAMのそれぞれについて示している。図より、 RAMを使用したときはサイクル数の二乗に比例したサーチステップ数を要するが、 CAMを使用すればサイクル数に比例したサーチステップ数で済むことがわかる。また図6は、CAMの効果に加えて三次元共有メモリの効果もある例を示している。このシミュレーションでは、問題(b)の解を求めるために要するステップ数を (1) RAMを用いたバス型マル

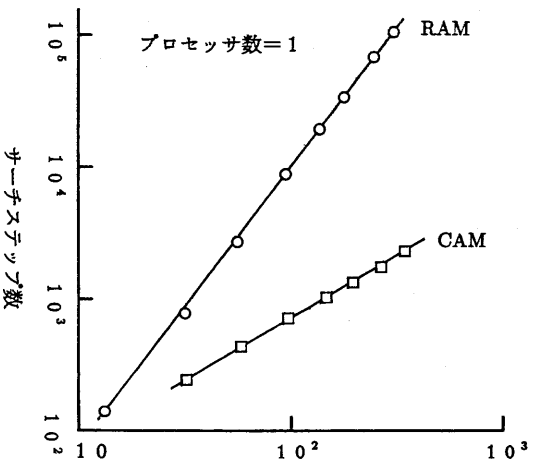
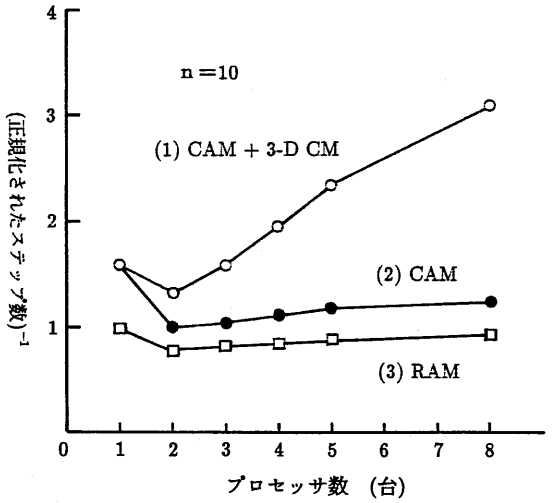
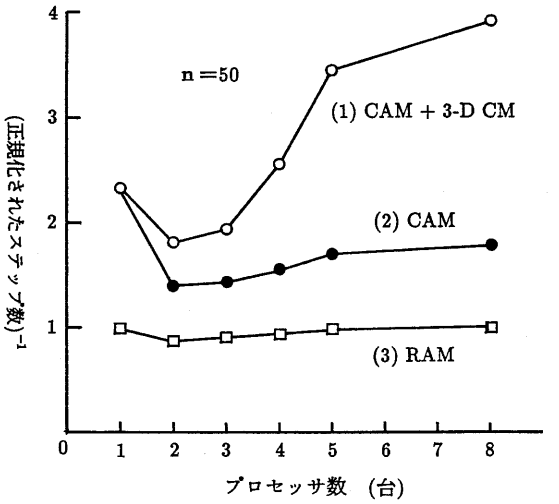


図5　高速マッチングプロセッサにおける
CAMの効果（問題(a)）

チプロセッサ、(2) CAMを用いたバス型マルチプロセッサ、(3) CAMと三次元共有メモリを併用したマルチプロセッサ、のそれぞれを想定して評価したものである。ここで係数成分を除外するため、1データをアクセスするのに要する時間と1データを転送するのに要する時間が同一であると仮定している。ただし三次元共有メモリを用いた場合、複数データの同時転送が可能である。図では横軸にプロセッサ数をとり縦軸にRAMを用いたシングルプロセッサに対する速度向上比をとっている。また同図(a)では問題に関係のないオブジェクト数nを10、



(a)



(b)

図6 高速マッチングプロセッサにおける三次元共有メモリの効果 (問題(b))

(b)では50としている。これからもわかるように、CAMの使用による高速化の効果はnが大きくなるほど増している。さらに、三次元共有メモリを用いることによって通信のためのバンド幅を大きくすることができ、これによって一層の高速化が実現される。

4. 非同期型並列プロダクションシステム (APPS)

本節では、APPSの実現のための方針と三次元集積回路を用いたその一実現法について述べる。提案する方式の効果は、バス結合マルチプロセッサUNIP[14]を用いたシミュレーションにより評価される。

4.1. 実現のための方針

APPSの実現において一番問題となるのは、文法的に依存関係のあるインスタンスを、どのようにして非同期的に実行させるかということである。可能性としては以下の方法が考えられる。

(1) 系を矛盾に導く可能性のあるルールを実行したときは必ず、全プロセスで矛盾の回避を一斉に行なう。

(2) ある一定数(たとえば10個)のルールを実行するたびに、系の無矛盾性のチェックを行なう。

(1)(2)のいずれの場合も、一度発生した系の矛盾を取り除き系の無矛盾性を保持するための何らかのメカニズムが必要である。(1)の方式では矛盾の発生する度にこのメカニズムを起動することができるため系内を確実に無矛盾に保つことができるが、その半面、無駄にこのメカニズムを起動してしまう可能性がある。また(2)の方式では遅延評価の効果などによって実行時間を短くすることはできるが、系内を常時無矛盾に保つことは困難になる。いずれを選択するかは実際の用途に依存するが、ここでは(1)に議論を限定する。以下では、三次元集積回路を用いた並列実行のためのアーキテクチャと、そのメカニズムを用いた無矛盾性の保持方式について述べる。

4.2. 三次元集積回路を用いた実現法

具体的な手法として、データベース中の各データをその存在理由 (すなわちそのデータを再び作りだすためには、現在のデータベースの中のどのようなデータの組合せが必要かを示すリスト) とともに記憶する方式を採用する。この方法は非単調推論を行なうためのメカニズムの一つである ATMS[13]などで用いられるものと本質的には同じである。

本稿で取り扱うPSで系を矛盾に導く可能性のある要因はインスタンス間の文法的な依存関係であり、具体的

には (a) インスタンス実行に伴うデータの削除 および (b) 特定のデータの不在を条件とするような条件部の記述の二つである。したがって(a)(b)以外の部分を実行している限りは、如何に非同期的にルールインスタンスの実行を行なおうとも系は無矛盾に保たれることになる。むろん一般には(a)(b)を他から完全に分離することは難しく、したがって非同期並列実行の効果よりもオーバヘッドの方が支配的となるが、データの消去が外部イベントによってのみ誘起されるような特殊な状況(例えばリアルタイム環境など)を想定すると、この分離にうまく意味づけできる場合がある。

ここでは(b)のような記述を許さないPSを想定する。このことは一般には記述能力を低下させるが、意味的な依存関係を持たないPSに話を限れば以下に述べる方針の(b)を許す場合への拡張は容易である。さて本稿で提案する非同期的な並列実行のための基本的な方針は、以下のようなものである。

[方針]

[step 1] 矛盾を起こさないインスタンスを実行している限りは、各プロセッサが非同期的にインスタンスの実行(データベースへのデータの追加)を行なう。データの追加は、マッチング処理と各データの存在理由の追加を誘起する。

[step 2] 矛盾を起こしうるインスタンスを実行(データの削除を実行)した場合は、そのことを(削除されるデータとともに)他のプロセッサに知らせる。

[step 3] 全プロセッサで矛盾を回避する。すなわち存在理由の失われたデータを新たに削除し、そのことを[step 2]と同様、全プロセッサに放送する。この操作を存在理由の変更がなくなるまで繰り返す。(この変更は、競合集合へも反映される。)

[step 4] 存在理由の変更が発生しなくなれば、[step 1]の処理に戻る。 □

ここで[step 1]で行なわれる追加データに伴う存在理由の追加は、並列処理においては他のインスタンスの実行とオーバラップすべき性質のものである。さらに、各データの存在理由を効率的に探索するためには、その時点の追加データに伴う存在理由の変更分だけを検出するようにしたほうがよい。(すなわち、その後に追加されたデータに伴う存在理由の変更は、別のプロセッサによって検出されるようにしたい。) また[step 3]の処理では、存在理由の変更とデータの削除を他のプロセッサに速やかに知らせる必要がある。

これらの要請は、三次元共有メモリをデータベース

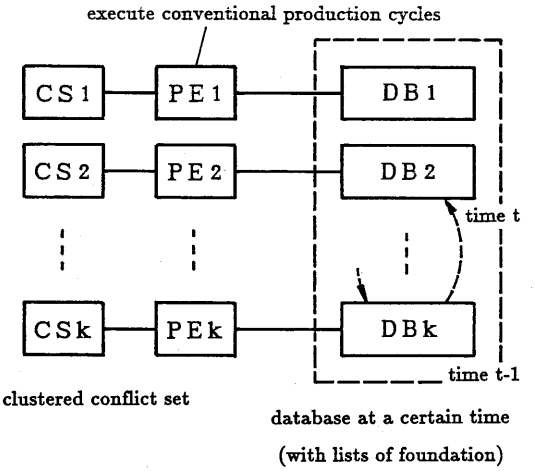


図7 理想的な三次元共有メモリを用いたAPPS

に使用することによってかなり満たすことができる。まずメモリの層間転送機能を用いて各層に別の時刻に対応するデータベースを割り当てることで、[step 1]の要請を満たすことができる(図7)。次に[step 3]の処理は、これを共有メモリとして用いることで達成される。具体的には、管理すべきデータクラスタを各プロセッサに静的に割り当てておき、各プロセッサにおける処理をデータ駆動的に進めていけばよい。

4.3. 評価

簡単な各データ間の存在理由の依存関係を仮定し、系の無矛盾性の回避処理[steps 2-3]を並列に(データ駆動的に)行なうことの効果を、当研究室のバス結合マルチプロセッサUNIP[14]上での実験により確認した。

図8に結果を示す。図では1データの消去に要する時間を固定(58msec)し、消去データの転送を15msecで行なう場合(実測値)と 0secで行なう場合(計算値)のそれぞれについて、プロセッサ数の増加に伴う速度向上比が示されている。なおここで、データベースそのものが共有メモリであり、データベース中の全データが[steps 2-3]によって消去されると仮定している。

図中の各曲線の飽和は、データ間依存関係の持つ並列性に因るものである。すなわちデータ数の増加に伴って並列化の効果は顕著となる。さらに同一のデータ数であっても依存関係が複雑であれば、通信時間に対する処理時間の割合が増大するため、より理想に近い速度向上が得られることが予想される。その場合プロセッサの負荷分散が重要な問題となるが、このことは、実際のシステムを想定した実験を含めて今後の課題である。

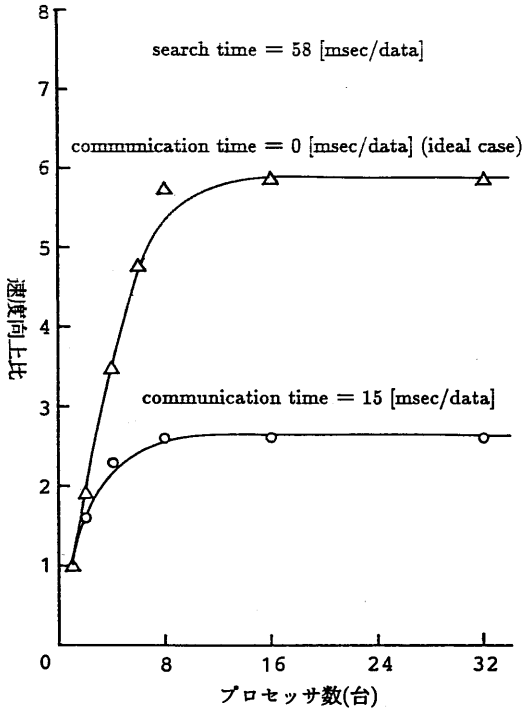


図8　UNIPによる[steps 2-3]のシミュレーション

5. おわりに

本稿では、プロダクションシステムの並列実行における留意点を示し、三次元集積回路を用いた新しい並列プロダクションシステムアーキテクチャの提案を行なった。従来から行なわれているこの分野の研究の多くはマッチングの高速化のみに注目したものであるが、プロダクションシステムの(推論の)性能を大きく左右しうるのはマッチングよりむしろ競合解消戦略であろうと考えられる。換言すれば、(OPSに限定されない)一般的なプロダクションシステムのための専用マシンを考える場合には競合解消戦略についての検討が不可欠であり、どのような戦略を想定するかによってマシンの設計方針が変わってくるものと思われる。

本稿では意味的な依存関係の全くないプロダクションシステムを想定したが、今後は、意味的な依存関係を含む場合について、その依存関係の表現方法も含めて検討していく予定である。

[謝辞]

UNIP上でのシミュレーション実験を行なってくれた現立石電機(株)の竹内拓二氏に感謝します。また三次元集積回路のデバイス技術について日頃から有益なコメントをいただきます、本学集積化システム研究センター長広瀬全孝教授および同センター山西正道教授に感謝します。

[参考文献]

[1] C.L.Forgy, "Rete: A Fast Algorithm for the Many Pattern/Many Object Pattern Match Problem," *Artificial Intelligence,* 19, pp.17-37 (1982).

[2] S.J.Stolfo, "Five Parallel Algorithms for Production System Execution on the DADO Machine," *AAAI-84,* pp.300-307 (1984).

[3] C.L.Forgy: *The OPS83 User's Manual,* Department of Computer Science, CMU (1985).

[4] A.Gupta: *Parallelism in Production Systems,* Pitman Publishing (1987).

[5] J.Miyazaki, et al., "A Shared Memory Architecture for MANJI Production System Machine," *Database Machines and Knowledge Base Machines* (ed. M.Kitsuregawa and H.Tanaka), pp.517-531, Kluwer Academic Publishers (1988).

[6] A. Gupta, et al., "Results Parallel Implementation of OPS5 on the Encore Multiprocessor," CMU-CS-87-146 (1987).

[7] S.J.Stolfo and D.P.Miranker, "The DADO Production System Machine," *JPDC,* 3, pp.269-296 (1986).

[8] R.Aibara: An Architectural Study on Massive Multiprocessor Systems, *Doctoral Dissertation,* Hiroshima University (1986).

[9] S.Fujita, et al., "A Template Matching Algorithm Using Optically-Connected 3-D VLSI Architecture, *14th ISCA,* pp.64-70 (1987).

[10] S.Fujita, R.Aibara and T.Ae, "A Real-Time Production System Architecture Using 3-D VLSI Technology," *Database Machines and Knowledge Base Machines* (ed. M.Kitsuregawa and H.Tanaka), pp.532-543, Kluwer Academic Publishers (1988).

[11] M.Koyanagi, et al., "Optically coupled three-dimensional common memory," *Optelectronics--Devices and Technologies,* 3, 1, pp.83-98, MITA PRESS (June 1988).

[12] T.Ishida and S.J.Stolfo, "Toward the Parallel Execution of Rules in Production System Programs," *Proc. ICPP,* pp.568-575, IEEE (Aug. 1985).

[13] J.de Kleer, "An Assumption-based TMS," *Artificial Intelligence,* 28, pp.127-162 (1986).

[14] 相原,阿江, "マルチマイクロプロセッサによるソート/サーチエンジンの試作", 情処学論, 26, 2 pp.349-355 (昭60.03).