ICOT Technical Memorandom: TM-1037

TM-1037

並列推論マシンのメカニズムと応用

瀧　和男

March, 1991

ICOT
Mita Kokusai Bldg. 21F
4-28 Mita 1-Chome
Minato-ku Tokyo 108 Japan
(03)3456-3191～5
Telex ICOT J32964

Institute for New Generation Computer Technology

並列推論マシンのメカニズムと応用

瀧 和男
(財) 新世代コンピュータ技術開発機構
taki@icot.or.jp

1 はじめに

並列推論マシンは、第五世代コンピュータの中核となるハードウェアで、知識処理を超高速実行する並列計算機である。その性能目標は、現在の汎用大型計算機の数百倍に及ぶ。将来の知識情報処理を支えるために、高度な知識処理メカニズムの研究開発に使える柔軟性と、並列処理による高い処理性能の実現を目指している。それらを実現するためのアプローチとして、多様でかつ規模の大きい知識処理と、高性能の並列処理の両方の要求を満たすように、中核となる言語、ハードウェア、そしてオペレーティングシステムをデザインするという方法をとってきた。この方法は、特定の知識処理方式を決めハードウェア化する方式とは、方向性が異なっているが、知識処理の高速化、高機能化を目指す点で共通する。

本稿では、並列推論マシンの研究開発について、はじめに研究開発の枠組を紹介し、つづいて主要技術となる並列処理に重点を置きながら、構成要素となる核言語、ハードウェア、オペレーティングシステムのおのおのについて、技術上の要点を解説する。また、並列処理の中でも研究要素の多い、ソフトウェア面での研究課題をまとめ、応用プログラムの開発事例についても紹介する。本稿により、第五世代コンピュータ研究開発の方向性、技術上の要点、その将来性や適用可能性に関する感触を多少なりとも持っていただくことができれば幸いである。

2 第五世代コンピュータと研究開発の枠組

2.1 第五世代コンピュータプロジェクト

第五世代コンピュータプロジェクトは、1990 年代そして 21 世紀に向けての高度知識情報処理を実現するために、その基盤となる技術を研究開発しようとするものである。通産省の主導の元に 1982 年に開始された、10 年計画の大規模プロジェクトである。

高度知識情報処理の目指すものは、知識を活用することによる品質の高い情報処理と、プログラミングの煩わしさから利用者をできるだけ解放することである。究極の実現イメージの一例としてよく引き合いに出すのは、「人間が知的な生産活動をする時に、脇にいて助けてくれる賢い助手」である。例えばわたしが計算機の設計をする時、設計方法の知識ベースや設計データベースを活用して設計作業を助けたり、新たな設計問題が発生した時には、その解決方法自体を可能な範囲で生成したりしてくれるような、スーパーワークステーションである。このような究極の姿に致る道程の、今はまだ中ほどを進んでいる。

このような高度知識情報処理の目標に比べると、現在の計算機で扱われている知識処理のレベルは極めて低く、まだまだ多大の研究努力が必要である。また将来の知識情報処理では、現状の 1000 倍程度大きな計算能力を必要とするであろうとも言われている。計算能力を稼ぐ方法として、特定の知識処理アルゴリズムをハードウェア化する方法も考えられるが、多様な知識処理に対応できるための柔軟性、要求性能の実現性の両面において、難しさがある。そこでわれわれは、多様な知識処理技術を研究開発するために共通に使える計算機言語を設計し、それを並列処理技術により高速実行するアプローチをとった。すなわち 図 1 に示すように、計算機言語には、知識表現や推論との関わりの深い論理型言語を基本として選び、それを並列処理により高速実行するシステムを研究開発するとともに、その上で知識処理の研究を行おうとするものである。

2.2 プロトタイプ・システムの構成

第五世代コンピュータ・プロトタイプ・システムは、論理型言語で書かれたプログラムを超高速実行する並列計算機システムである。その構成は、図 2 のようになる [瀧 91] [内田 91]。最下層はハードウェアの層であり、並列推論マシン PIM はここにはいる。次の層は核言語 KL1 で、本システムのために新たに設計された並列の論理型言語であり、ハードウェアとソフトウェアのインタフェースとなる。KL1 は、知識処理プログラムを記述するための、汎用記号処理言語としての側面と、強力な並列処理記述言語としての側面を合わせ持つ。並列推論マシン PIM は、KL1 を高速実行するための数々のメカニズムを備えたハードウェアである。その上に、並列オペ

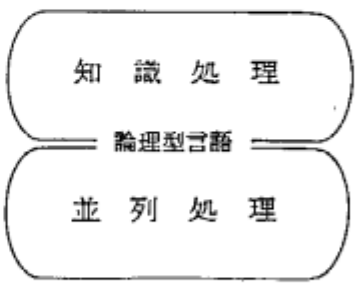


図 1: 第五世代コンピュータ研究開発の枠組

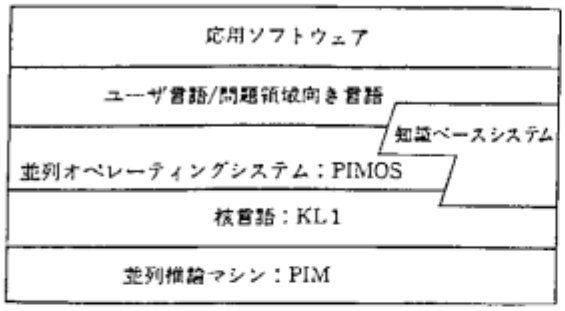


図 2: 第五世代コンピュータ・プロトタイプ・システムの構成

レーティングシステム PIMOS の層がある。ここから上は、すべて KL1 で記述されている。データベースを含む知識ベースシステムも、PIMOS と性質同層にある。知識処理を始めとする応用ソフトウェアは、現在はまだほとんど KL1 をそのまま用いて記述しているが、プログラマーの負担軽減のために、ユーザー言語や特定の問題領域向き言語が研究開発され始めている。応用ソフトウェアの層は実は大変に厚く、知識処理に関する諸々の応用ソフトウェアや研究ツールの他、並列処理で高速化することに重点が置かれた多くのソフトウェアがここに含まれる。

本システムは、すでに述べたように、並列処理と記号処理という 2 つの技術的側面を持っているが、中でも重要性が高いにもかかわらず、未解決研究課題の量が多くレンジも広いのが並列処理である。記号処理については、研究開発用の言語システムとして LISP や Prolog などが実績を積んでおり、これらの言語用の専用マシンも市販品が存在する。つまり手本にできる技術がかなり蓄積している (Prolog マシンに関しては本プロジェクトの貢献が大きい) ので、本システムでもそれらを多く利用した。一方の並列処理については、汎用的な大規模並列処理システムとして成功した例がほとんど見当たらず、とくに並列ソフトウェアに関する技術蓄積は皆無に近い。したがって、われわれは、第五世代コンピュータ・プロトタイプ・システムの言語、ハードウェア、オペレーティングシステムを研究開発するにあたっては、将来の知識情報処理などの、大規模並列システムの構築に耐え得るだけの、並列処理技術の研究開発を、最重要課題として取り組んできたのである。

3 核言語 KL1 の設計

KL1 は、第五世代コンピュータ・プロトタイプ・システムの中核言語であり、強力な並列処理記述言語と、汎用の記号処理言語の機能を合わせ持つ。ハードウェアとソフトウェアのインタフェースとして、両者に先立ち設計され、本システムの技術的根幹とも呼べるほど重要な位置を占める。ハードウェアの中の数々のメカニズムは、本言語を効率良く実行するために設計されたものであり、オペレーティングシステムの実現方法も本言語の機能仕様との関係が深い。そのため、以下では、プログラム例を含めて少し詳しく見てゆくことにする。

核言語 KL1 は、Prolog に似た論理型言語であり、次のような特徴を持つ。

1. 並列処理の記述のために、強力な言語機能を持つ。
   (a) 並列がベースとなった言語であり、逐次言語の拡張ではないために、並列度の高いプログラムを記述するのに適する。
   (b) 並列プログラムの中でもっともバグの原因となりやすい通信と同期の記述に関し、超高級言語である。
   (c) pragma の機能は、負荷分散の実験や性能のチューニングなどの並列処理の研究に好適である。
   (d) 並列オブジェクト指向プログラムが、極めて自然に記述でき、複雑な構造を持つ、あるいは規模の大きい並列プログラムの開発が可能である。
2. 汎用の記号処理言語である。
   (a) 言語の構文は Prolog に極めて近いが、Prolog が持っている知識表現言語、推論言語の性質は薄められており、汎用の記号処理言語に近い。Prolog や LISP の持つ、データ型の宣言を必要としない性質、リストや構造体操作機能、動的メモリ割り当て機能など、記号処理言語としての基本的性質は保存されている。
3. 一階述語論理に基づく理論的ベースのしっかりした言語であるため、プログラム変換や合成などの先端ソフトウェア技術を適用しやすい。
4. オペレーティングシステム記述のための機能を持ち、応用プログラムだけでなくシステムプログラムの記述が可能である。

KL1 は、論理型言語 CHC [古川 87] の拡張版であり、プログラムは次のような形の節を並べたものとして表現される。

```
H :- G1,.., Gm | B1,.., Bn.
```

H をヘッド、G1,.., Gm をガードゴール、B1,.., Bn をボディゴールと呼ぶ。シンタックスは Prolog とほとんど同じである。| は Prolog のカットと似た意味を持ち、コミットメントオペレータと呼ぶ。厳密性を我慢して直観的に分かり易い説明をしよう。ヘッドとガード部分は、パターンマッチと条件テスト、および同期の機能を持つ。この部分は、すべての節について意味上は同時に試される。パターンマッチと条件テストを試みて、同期待ちにもならずに成功した節が 1 つだけ決まる (複数の節が成功した場合は処理系が 1 つを選ぶ)。これをコミットするという。そうすると、その節のすべてのボディゴールが並列に実行される (正しくは実行可能状態になる)。

```
fact(0,Y) :-  true | Y = 1.
fact(X,Y) :- X > 0 |
   sub(X,1,X1), fact(X1,Y1), mult(X,Y1,Y).
```

これは階乗を計算するプログラムである。このプログラムで2引数のfactの呼出しがあると、第1節ではヘッドの中で、第1引数が0であるかどうかの条件テストが行われる。第1引数にまだ値が決まっていなければ、待ちにはいる。第2節ではヘッドの中での条件テストはなく、ガードゴールで数の大小比較が行われる。この場合も、比較対象のXの値が決まっていないと待ちにはいる。このように同期のメカニズムは簡単で、ヘッドおよびガードゴールでテストしたい対象の変数が値を持たないと、値が決まるまで実行が中断される。

第1引数に値が決まり、それがゼロ以上なら、条件判定に従ってどちらかの節がコミットされる。例えば、fact(10,Y) の呼出しがあったときは、待ち状態に入ることなく第2節がコミットされる。すると、ボディゴールの sub, fact, mult が並列実行可能となる。これらが1台のプロセッサで実行されるならば、処理系の都合で勝手に実行順序が決められるし、別のプロセッサならばほんとうの並列実行となる。第1引数の値が負の場合は、第1節でも第2節でも条件テストに失敗する。これはエラー状態 (exception) として報告される。

実行可能なゴールを1個選び、対応する節に対してヘッドとガードゴールのテストを実施し、コミットしてボディゴールが並列実行可能状態になるまでを、1つの論理的推論 (logical inference) と呼ぶことがある。KL1の実行メカニズムは、Prologと同様の「ホーン節上の後ろ向き推論」に基づくが、Prologに見られるバックトラック機能は存在しない。並列処理の機能との同居が難しかったためである。したがって、KL1とPrologでは、プログラミングスタイルが異なることに注意が必要である。

並列に実行されるボディゴールは普通は変数を共有しており、それらの変数を使って、互いに計算結果を伝え合う。fact(X1,Y1) は、変数X1をとおして sub(X,1,X1) の結果をもらう必要がある。そこでこの場合は、たとえ別プロセッサに割り付けられて並列に実行が始まっても、X1の値をテストしようとした時点ですぐに待ちにはいることになる。KL1 (GHC) ではこのように、プログラマーが陽に同期を記述する必要はなく、共有変数を用いたデータの受け渡しを記述しておくと、一種のデータ駆動型の同期メカニズムが自動的に働く。逐次型言語に通信と同期のプリミティブを入れてプログラムを書く場合には、通信と同期に関する難しいバグがしばしば問題にされるのに対して、KL1プログラミングでは、「データ駆動に基づく暗黙の同期」のおかげで、この種のバグが皆無といえるほど少ないことが経験的に確かめられている。

次にKL1言語に特有のpragmaについて見ておこう [宮崎87]。ゴールをどこで実行するかを指定する @node(X) と、ゴールの実行優先度を指定する @priority(Y) がある。次のようなプログラムを考えよう。

```
go(In,Out) :- true | producer(In,X),
                     filter(X,Y),
                     consumer(Y,Out).
```

これは、producer, filter, consumer という3つのゴール (いずれも末尾再帰呼出しにより並行プロセスとして生き続ける) をフォークして、パイプライン並列的に仕事をさせるプログラムである。producerはInからパラメータをもらい、値をつぎつぎに生成してはfilterに送る。このとき変数X はストリーム (リストの要素に次々に値が入るデータの流れ道) として使われる。filterはもらった値を加工しては、Y をやはりストリームとして用いながらconsumerに送る。consumerはつぎつぎに送られる値から答を計算しOutに返す。プログラマーは上のプログラムを書いた時点で、1台のプロセッサ上でそれをデバッグする。この段階で、論理的には正しいプログラムとなる。つまりアルゴリズムに関わるバグは除かれる。つぎに並列実行のためにpragmaを付ける。

```
go(In,Out) :- true | producer(In,X)@node(2),
                     filter(X,Y)@node(3),
                     consumer(Y,Out).
```

これにより、producerとfilterは、2および3の番号が与えられたプロセッサ上で実行される。consumerは、goが呼び出されたプロセッサ上に留まる。ノード番号は計算して与えることもできる。

```
go(In,Param,Out) :- true |
                compute_node_num(Param,J,K),
                producer(In,X)@node(J),
                filter(X,Y)@node(K),
                consumer(Y,Out).
```

pragmaを付け替えることで、負荷バランスなどの性能に関わる調整が、プログラムの意味を変えることなく簡単に行える。また実行優先度については、例えば4096段階というように細かい指定を許しており、ユーザープログラムの中で処理効率向上のために使われることを意図している (「詰め碁」の項参照)。

上のproducer, filter, consumerの各プロセスが、ストリームで相互に接続されている構造は、ストリームを流れるのが単なるデータではなくて意味を持つメッセージである場合、そのまま並列オブジェクト指向プログラムと考えることができる。オブジェクトをプロセスとして実現しているわけである。このスタイルは、複雑な構造を持つ規模の大きい並列プログラムを実現するときに特に有効である (「LSI配線プログラム」の項参照)。

4 並列推論マシンPIMの方式

並列推論マシンPIM (Parallel Inference Machine) は、核言語KL1で書かれたプログラムを超高速実行する並列計算機である [Taki 89] [Goto 89]。KL1の実行メカニズムが、非常に単純な論理的推論に基づき、それを多くのプロセッサで並列実行するため、並列推論マシンと呼ば

れる。人間が行うような高度な推論は機械が直接実行するのではなく、それはKL1で書いたプログラムにより実現する。

PIMの特徴は次の2点に集約される。第1に、ハードウェア構造の面で、将来の大規模並列計算機の1つの標準的構成方法となるべき雛形を目指している。第2は、千台規模のプロセッサが接続されているにもかかわらず、利用者には「システム全体が1つのKL1言語処理系として見える」ことである。

4.1 2種類の並列処理方式

並列計算機の構造としては、MIMD型(複数命令による複数データ処理)と SIMD型(単一命令による複数データ処理)の2種類がある。同じ並列計算機とはいっても、この2つは性質が大きく異なっている。

SIMD型は、小能力のプロセッサがたくさん接続され(例えばコネクションマシンでは6万5千台)、ただ1台の制御装置からの指令で、全プロセッサが足並みを揃えて同じ命令を実行する。したがって、SIMDは、画像処理や密な行列計算のように、「均質なデータに対して均質な処理を行う」場合に適している。

一方、MIMDでは、SIMDよりは少ないけれども、高性能のプロセッサが接続されていて(例えばPIM/pでは、汎用中型ないし大型計算機並みのプロセッサが512台)、それぞれが別の命令を実行する。したがって、処理するデータや処理の仕方が、プロセッサ毎に異なっていて、全体として不均質であってもかまわない。知識処理を始めとする多くの応用では、このような状況が頻繁に発生するので、MIMD型が必要とされる範囲は広いと期待されている。ただし、多くのプロセッサが遊ばないように、仕事を分配し制御する技術は、SIMD型に比べて難しい。PIMは、このようなMIMD型並列計算機の1つの標準的な姿を指向している

4.2 PIMの構造

現在、われわれは、構成が少しずつ異なったPIMを同時並行的に開発中であるが、ここでは、その内の代表的なマシンであるPIM/pを例にとって説明する(図3)。PIM/pでは、高い処理性能を実現するために、512台という多数のプロセッサを階層構造的に接続する。階層の上層では、多くのプロセッサの接続を重視した方式、中層では、少数のプロセッサが密に協調するときの性能を重視した方式、下層の個々のプロセッサでは、KL1言語の処理性能を高めるための方式を取っている[服部89]。以下たとえ話を交えながら、上層から順に説明する。

PIM/pの上層構造は、「マルチ・クラスタ」といって、「クラスタ」と呼ぶ処理ユニットが最大で64個、高速ネットワークで接続されている。個々のクラスタは独立のメモリを持っていて、クラスタ間は「メッセージ交換」で通信し合う。このメッセージを飛ばし合って通信する方式は、本質的に、大規模なシステムの接続に適している。 このような構成は、会社の部が別々の場所にあって、特急の社内便で結ばれているようなものである。ただし、通信するには、メッセージを決まった様式の便箋に書き、決まった封筒に入れて送らなければならないので、手間がかかる。したがって、頻繁に通信を利用すると、手間ばかり増えて本来の仕事に差し支えるので注意を要する。

一方、仕事の種類によっては、仕事の進み具合をお互いに頻繁に見せ合って、情報交換しながら仕事を続けるようなこともある。これは、いわば大部屋で書類を交換しながら仕事をしている場合である。PIM/pの中層構造にあたる「クラスタの内部構造」は、実は、このような種類の仕事に適した設計になっている。一般の会社組織であっても、緊密に情報交換し合う相手は限られている。クラスタの内部構成も同じで、8台のプロセッサが共有メモリを介して密結合されている。つまりイメージとしては、各プロセッサが社員1人1人に相当し、8人が、内部情報の非常によく通るグループを構成しているわけである。

最後に、1人1人の社員すなわちプロセッサは、KL1言語を実行することを非常に得意とする。ハードウェア自体が、KL1実行向きに作られているからである。ただプロセッサハードウェアのもっとも基本的な作りに関しては、普通の逐次型コンピュータと大きな差はない。

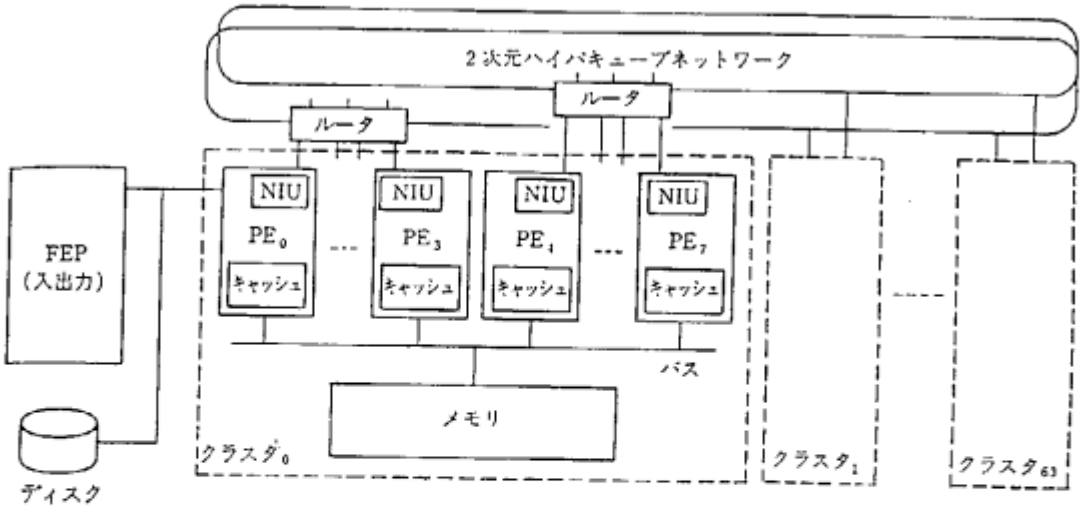
このように、PIM/pの構成は、多くの人間が協力しながら働く会社組織によく似ている。つまり、仕事が組織ごとに不均質に分配され、それぞれの負担も動的に大きく変化するような仕事に対して、多くのプロセッサが協力し合って対処できる構成になっている。

技術的なポイントをまとめておこう。

上層構造のポイントは、ネットワーク構成である。多くのクラスタの接続、高速のメッセージ転送、直径の小さいネットワークトポロジーなどが重要である。PIM/pではバンド幅を広げるために2重化した、ハイパーキューブ(超立方体)ネットワークを使用している。

中層では、クラスタ内のプロセッサ間接続方式がポイントである。プロセッサ間での頻繁かつ大量のデータ交換を可能とする方式が重要である。PIM/pでは、各プロセッサが並列キャッシュメモリ(スヌープキャッシュ)を持ち、共有バスを通過路として使用している。また、複数のプロセッサが協調動作するとき重要な、共有メモリ上のデータを排他的に参照・変更する機能を持つ。

下層のプロセッサ1台1台については、記号処理用プロセッサに共通な機能として、データの1語ごとにその属性を示すタグビットを持つこと、タグの操作や判定の機能、メモリ管理機能の1つである塵集めの支援機能などを持つ。また粒度の小さい並列処理の効率を高めるために、高速のプロセススイッチ機能も重要である。またKL1言語機能とマシン命令間のギャップを埋めるために、高速のサブルーチン呼びだし機能も備える。PIM/pでは、KL1言語で書かれたプログラムは、KL1-bと呼ぶ命令セット(Prologの場合のWAMコードに相当)に翻訳され、一部はさらに低レベルのマシン命令に展開されて、実行される。



PE：要素プロセッサ，FEP：フロントエンドプロセッサ，NIU：ネットワークインタフェース・ユニット

図 3: 並列推論マシン PIM/p の構成

4.3 KL1 言語処理系の実装

KL1 言語処理系は、PIM/p のシステム全体の上に、分布して存在する。プログラマは、問題解法のアルゴリズムを記述するとき、512 台のプロセッサを意識せず、システム全体を 1 個の KL1 言語処理系と考えてプログラムできる。処理性能を引き出すことを考えるときにだけ、64 個のクラスタを意識し、仕事を部分問題に分けてクラスタへの割り付けを指示する必要がある。これは、部分問題を代表するゴールに pragma を付加するだけで、簡単に行うことができる。

PIM/p のクラスタの間では、メッセージ通信が行われる。しかしながら、KL1 の設計の項で述べたように、KL1 言語のレベルには、通信や同期のための特別なプリミティブは存在しない。テストしたい変数に値が決まっているかどうかにより、ヘッドおよびガードゴール部分で、自動的に同期が行われ、必要に応じて通信が発生する。例えば変数を共有するゴールの一方が、pragma によって別のクラスタに移された場合、変数の参照関係は KL1 処理系が自動的に維持する。クラスタ境界をまたがる変数読み出しが必要になると、処理系が自動的に通信メッセージを組み立て送出する。PIM の KL1 処理系は、各クラスタでアドレス空間が独立した分散メモリ型ハードウェア上に、単一の処理系 (単一の論理空間を仮定したような処理系) を苦心して載せたシステムともいえる。これは一重に、既に述べた KL1 言語の良さを損なうことなく利用者に提供するためである。

4.4 PIM の性能

PIM/p の性能面について触れておこう。1 台のプロセッサは、汎用計算機でいうと、中型の上位機種から大型の下位機種相当の規模を持つ。新規開発の VLSI で実現され、約 60 ナノ秒の命令サイクルタイムを持つ。4 段のパイプライン制御を行っている。クラスタ当たりの共有メモリ容量は 32 メガ語である。512 プロセッサ構成の PIM/p の場合、知識処理によく現れる探索問題を実行した場合の最大性能は、並列処理によるロスを含めても、汎用大型計算機の最上位機種で同じ問題を解く場合に比べ、百倍を越えると期待している。またメモリ容量も、システム全体で 16 ギガバイトと格段に大きいため、大規模問題への適用が可能である。

5 並列オペレーティングシステム PIMOS

PIMOS (パイモスと読む; PIM OS の意) は、並列推論マシン PIM の並列オペレーティングシステムである。並列ソフトウェアの研究開発を行う実験マシンのための実用 OS ともいえる。PIMOS は実験マシンによく見掛けるバックエンド OS ではなく、それ自体で完結したスタンドアローン OS である。また 1 個の OS が並列計算機上に分布して存在するように作られており、要素プロセッサ毎に独立の OS を置くような分散 OS ではない。PIMOS はすべて KL1 で書かれており、アーキテクチャの細部からの独立性は高い [佐藤 89]。

PIMOS の機能を説明する前に、KL1 と PIMOS の関係で特徴的なところを述べておこう。KL1 言語の良さを殺さないために、以下に示すような機能は、OS の中でなく KL1 言語処理系の中で実現している。

1. ユーザータスクは小粒度のプロセスの集合と考えるべきなので、そのような集合を実行管理の単位として取り扱う機能 = 荘園機能。計算資源の割り当ても荘園単位に行っている。
2. プロセスの生成とスケジューリング
3. 実行時のメモリの割り当てと解放
4. PE 毎のローカルアドレスとグローバルな ID (変数番号など) との対応付け、および PE 境界をまたがるポインタの管理

5. ネットワークメッセージの組み立て、分解

すなわち、OS カーネルのかなりの部分が言語処理系に取り込まれた作りになっている。これらは主に、小粒度のプロセスを取り扱う場合のオーバーヘッド軽減策である。

処理系が OS カーネル部分の仕事の多くを引き受けてくれたので、PIMOS ではより上層の次のような機能を実現している。

- ユーザータスクの管理
- 入出力資源を中心とする資源管理
- プログラムコードの管理
- シェル機能を始めとするユーティリティー
- コンパイラ、並列処理用トレーサー、性能測定プログラムなどのプログラム開発環境
- 自動負荷分散ライブラリ等の各種ライブラリ

PIMOS を実現するに当たっては、資源管理の考え方がその骨格をなすものとして重要な位置をしめている。できる限り高い並列度を引き出して実行しようとしているユーザープログラムに対して、OS が従来通りのテーブルを中心とした集中管理をしていては、OS 部分が並列処理のボトルネックともなりかねない。そこで PIMOS では、あらゆる資源を木構造を用いて分散管理し、ボトルネックの発生を防ぎつつ、OS の発生する通信量の削減にも努めている。

6 並列ソフトウェアの研究課題

並列ソフトウェアの研究は、並列処理研究の中でも最重要課題である。MIMD 型の大規模並列計算機を効率良く動かすために、必須ともいえる並列ソフトウェアの基礎的な話題として、並列プログラミングは一体何が難しいのか、研究課題は何なのか、ごく簡単に整理しておこう。

6.1 期待と現状のギャップ

非常に良い並列言語があったとして、ユーザーが手持ちの応用をその言語で書き下すと、あとはシステム(並列マシン、その OS、言語処理系を含む)が自動的に、高い効率で並列実行してくれる、というのはほとんどユーザーにとっての理想の姿である。ところが現実には、プログラムを設計するところから、高い効率を引き出して並列実行するところまでの間に、技術的に解決されていない多くの課題が存在している。

6.2 コンカレント・プログラミングの研究課題

図4 に、それらの研究課題をまとめる。一番上は、並列アルゴリズムである。逐次マシンで使われている良いアルゴリズムは、そのほとんどのものが強い逐次性を

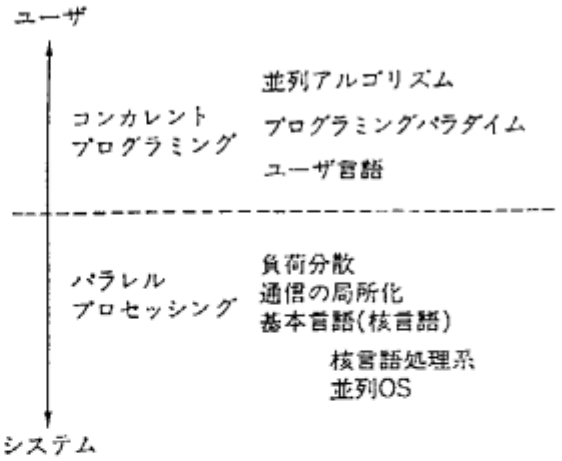


図 4: 並列ソフトウェアの研究課題

持つ。したがって、これを並列言語で書き下しただけでは、並列実行はできない。多くの場合、高い並列性を実現できるアルゴリズムあるいは逐次性の低いアルゴリズムに作り直さなければならないと考えるべきである。そしてそのとき、逐次アルゴリズムに比べて、計算量のオーダーを増加させないように、アルゴリズム設計の段階で十分注意が必要である。オーダーが増加してしまうと、大きな問題を持ってきたときに、プロセッサ台数で稼げる性能向上を越えて、計算量の方が増大し、実行時間は逐次実行より悪化する場合もある。2番目はプログラミング・パラダイムである。聞き慣れない言葉かも知れないが、プログラミングの型、手本とでも言おうか。プログラミングの上層では、問題のモデル化の技法やプログラム構造の決め方の技法であり、もう少し下層ではプログラミング・テクニックなどが含まれる。これらの技術も蓄積されていない。またプログラミングの型を与えるような並列ユーザー言語の開発も重要である。プログラミングの中でこれまで述べてきた辺りの仕事は、最低限ユーザーが行わねばならない仕事である。これをコンカレント・プログラミングの研究課題と呼んでいる。ここに含まれる並列性は論理的並列性で、それが実行時にマシンにどうマッピングされるかはこのレベルでは一切扱われていない。アルゴリズムが 10 万の並列性を持っていても、それを 1,000 プロセッサのマシンにマッピングしてもよいし、100 プロセッサのマシンにマッピングしてもよいのである。

6.3 パラレル・プロセッシングの研究課題

マッピングの話はパラレル・プロセッシングの問題である。これは本来、システムが丸抱えで面倒をみてくれれば有難いが、まだその段階には遥かに至らない。パラレル・プロセッシングに関わる技術課題は次のとおりである。問題を多くの部分問題に分けプロセッサの負荷が均等になるように、また暇なプロセッサが出ないように割り付けを行うのが負荷分散である。これは仕事をばら撒くはなしである。

一方で、部分問題の間で多くの通信を行うものどうし

は、ネットワークの負荷を軽減するためになるべく近く配置すべきである。これを通信の局所化という。これは仕事をばら撒かずに固めるはなしであり、さきに述べた負荷分散とは反対の要求である。これを同時に満たそうとするところに難しさがある。さらにそれらを記述するためにどんな言語を用意するか、OSでどこまでサポートするか等々、確立されてないものばかりである。

したがってユーザーがやらねばならないコンカレント・プログラミングのはなしも、システム側にやってもらいたいパラレル・プロセッシングのはなしも、現時点ではこれらをひっくるめてプログラマが書いている。現在の並列計算機では、ユーザーとシステムを作る人の距離は離れておらず、システム作りに手を染めている人が応用プログラムも書き、両方を含めた研究をしている段階といえる。

図4 をもう一度眺めなおしてみると、アルゴリズムも、プログラミングの型も、負荷分散の概念も、さらに言語もOSも、すべて新しいことに気付く。これはもう、並列処理は *Computing* の世界の *New Culture* と考えるべきではないだろうか。新しい文化をつくるという重大な仕事に、いま取り組んでいるのである。

7 並列プログラミングの具体的取組み

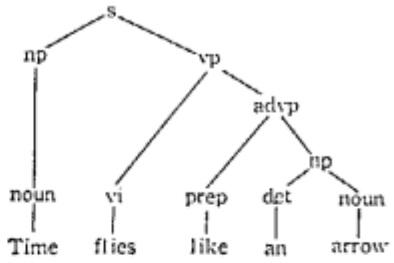
ICOTにおける並列プログラムの開発は、2つの面で大きな意味を持つ。第1は、個々のプログラム開発の中で、上に述べた並列ソフトウェアの研究課題に取り組み、その中で試された数々の工夫を一般化し、OSや言語処理系の中でサポートできるようにフィードバックをかけてゆくことである。第2は、大規模で実用レベルに近いプログラム開発を通して、並列処理および知識処理に関する第五世代コンピュータの適用性を実証することである。これらの2面を持つ並列プログラム研究開発の状況を、以下では3つの節にに分けて解説する。

7.1 初期の並列応用実験プログラム

ここでは、並列推論マシン実験機であるMulti-PSI [Taki 88] [Nakajima 90] の上に、最初に作られた4つの実験的プログラムと、その中で行われた研究を紹介する。それらは、各々異なる種類のプログラム構造と異なる実行特性を持つようなものが選ばれた [Ichiyoshi 89]。

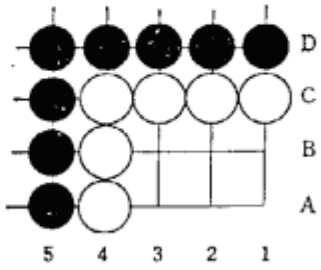
7.1.1 自然言語構文解析

ICOTで研究を続けているDUALSという名の談話理解システムから、構文解析だけを取り出して並列化したものが、自然言語構文解析プログラムPAXである（図5）。レイヤードストリームと呼ばれるプログラミング手法で実現され、重複を避けたボトムアップの全解探索を行う。文章の部分構造が決まるたびに新しいプロセスが作られ、それらがネットワーク状につながるため、プロセス間でグローバルな通信が発生し易い。この種のプログラム構造を持つ問題は、分散メモリ型並列マシンにとって、もっとも性能を出しにくい問題である。通信のオーバーヘッドを軽減するため、通信の局所化を優先する負荷分散手法が試された [寿崎 89]。



ーボトムアップ全解探索
ー通信の局所化優先の負荷分散

図 5: 自然言語構文解析



ーゲーム木探索(α-β枝刈り法)の
並列化アルゴリズム

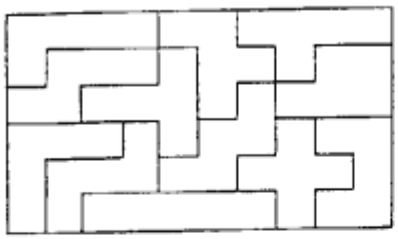
図 6: 詰め碁

負荷の均等化と通信の局所化が対立したこと、文1個の解析は、並列化前の解析プログラムでも効率が高く、計算量自体が小さいことの2点により、並列処理による性能向上はプロセッサ数をふやしても約3から6倍で頭打ちとなった。但し機械翻訳のように、多くの文をパイプライン的に投入する使い方が可能な状況では、より高い性能向上が得られる。

7.1.2 詰め碁

詰め碁は、囲碁対局の部分問題である（図6）。AI問題の分類でいうと、ゲーム木の探索問題に当たる。昔から探索空間を狭める効率の良いアルゴリズムとして、アルファ・ベータ枝刈り法が使われているが、これが非常に逐次性の強いアルゴリズムである。順に実行することで枝刈りができる。逆に並列に枝を展開すると、枝が刈れず無駄な計算ばかり増えるという困った事態が起こる。これを専門用語では「無駄な見込み計算: speculative computation」という。そこで並列性を引き出しながら、枝刈り効率も低下させない並列アルゴリズムの研究と負荷分散の実験を行った [沖 89]。

KL1の実行優先度制御の機能を用い、枝の並列展開の時に、有望な解を生じそうな枝の順に高い優先度を与えることで、そこから得られた結果を枝刈りに利用することを試みた。並列処理による性能向上の割合は、問題に依存して大きくばらついた。興味深いのは、良い枝か否かを判定する知識が当たる場合には、性能向上が低く、



— OR並列型の全解探索
— 多階層動的負荷分散

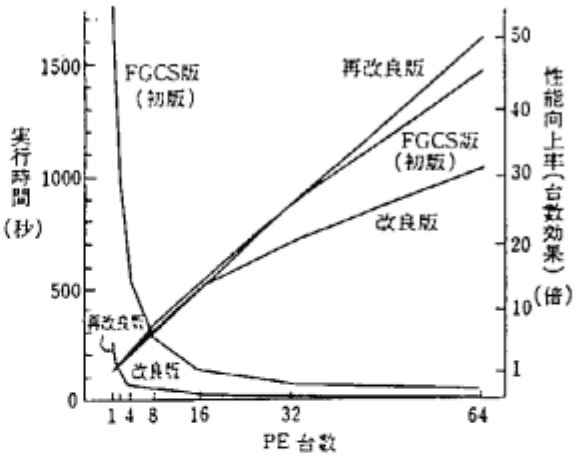
図 7: 詰め込みパズル (ペントミノ)

知識が不適当で見込み外れを起こし、逐次でやっても枝刈り効率が上がらないような場合には、並列の性能向上が大きいことであった。

7.1.3 詰め込みパズル

これは玩具屋でプラパズルという名称で売っているものである (図 7)。正方形 5 個分の面積を持つ形の異なるプラスチックのピースが 12 個ある。これを 6 × 10 マスの箱に詰め込むあらゆる詰め込み方を求める (別名ペントミノ)。全部で数千の解があり、これを並列に求める。OR 並列型の全解探索と呼ばれることもある。箱の左上コーナーからピースを置いてゆくことにし、異なる形のピースを選択することが探索木の兄弟の枝を伸ばすことに当たると考えよう。異なる枝どうしではお互いに通信することなく探索が進められるので、通信の局所化を考える必要性が少ない。すなわち、負荷分散さえうまくやれば、並列処理で高い性能を得やすい問題である。動的負荷分散を階層化した方式を考案し、性能を評価した [古市 89] [Furuichi 90]。

10 ピースの詰め込みパズル (テトロミノ) を用いて性能測定した結果を 図 8 に示す。右上がりのグラフが、プロセッサ台数に対する性能向上率 (台数効果) を表わし、左から急激に下がるグラフが、プロセッサ台数に対する実行時間を示す。初版の FGCS 版は、データ表現などのまずさのために、実行時間は長くかかった (それでも十分速かったが)。台数効果は、64PE で 45 倍であり良かった。改良版ではデータ構造などを改良し動的負荷分散も取り入れた。実行時間は数倍短縮されたが、台数効果は返って悪くなった。プログラムの改良で計算量が減った (問題が小さくなった) ためともいえるが、性能が上がって台数効果が下がるのは、よく聞くはなしである。分析の結果、動的負荷分散のマスター PE がボトルネック傾向であることが分かり、2 階層の動的負荷分散に直したのが再改良版である。最も高い絶対性能を示し、台数効果も 64PE で 50 倍が得られた。MIMD マシンにとって十分良い値である。このとき、全解 3106 個を求める実行時間はただの 5 秒であり、システムの全体性能として約 2 メガ LIPS (Logical Inference / Sec) が得られていることが分かった。同じデータ構造を用いて最高性能の汎用計算機で解くのと比べ、少なく見積もっても数倍以上早いはずである。どんな問題でもこの程度の高い効率が得られるならば、すべての計算機利用者が MIMD マシンに飛び付くことであろう。



詰込みパズル(5×8マス, 10ピース)

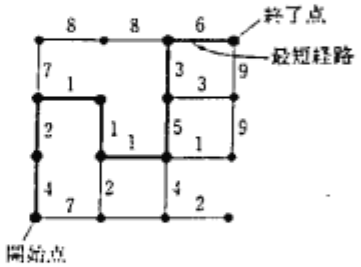
FGCS版: 静的負荷分散
改良版: 動的負荷分散
再改良版: (二階層)動的負荷分散

- 改良版は速くなったが, 台数効果は下がった.
- 再改良版で, 台数効果も改善
- リニア・スピードアップ

図 8: 詰め込みパズルの測定結果

7.1.4 最短経路問題

最短経路問題は、図 9 に示すように、各辺に異なるコストを持つグラフ上で、始点から終点に至る最小コスト経路を求めよという最良解探索問題である。その中の特別な問題として、1 個の始点から、他の全ての点に至る最短経路を求める問題を取り上げた。このための分散アルゴリズムを設計し、全てのグラフ上の点を独立のプロセスとし、それらがメッセージ交換しながら探索を進めるような、並行オブジェクトのモデルに基づくプログラミングを実現した。KL1 に適したプログラミングスタイルである。また計算量のオーダーは、逐次版のダイクストラのアルゴリズムと等しいものが実現できた。静的負



— 最良解探索問題
— 並列アルゴリズムの工夫
— 多重マッピング・静的負荷分散

図 9: 最短経路問題

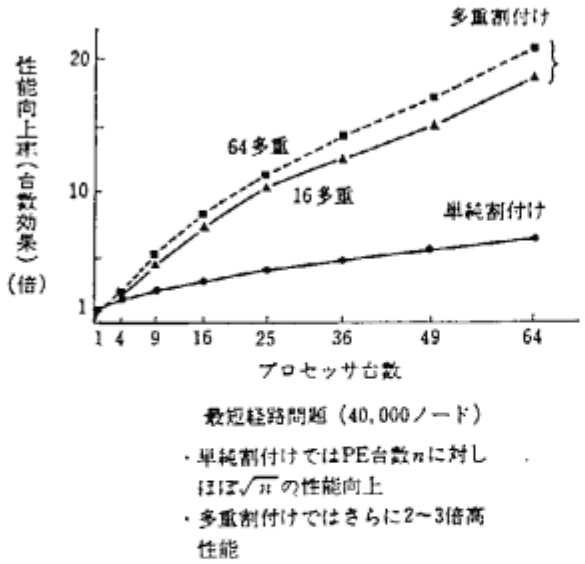


図 10: 最短経路問題の測定結果

荷分散を使用し、グラフをプロセッサ数丁度に分割してそのままマッピングする単純割付、プロセッサ数の整数倍に分割してプロセッサ上にサイクリックにマッピングしてゆく多重割付を実験した [和田 89]。

測定には乱数から作った 40,000 点からなるグラフを用いた。測定結果を 図 10 に示す。単純割付では、PE 数 n に対する性能向上は、ほぼ n の平方根となった。これは簡単なモデルを用いて、理論的に説明もできる。経路探索のメッセージは、開始点から波面状に広がるため、単純割付ではメッセージが通過した後のプロセッサは暇になり、効率が悪い。これを救うのが多重割付である。1 つのプロセッサには、部分グラフがサイクリックに割付けられるため、波面は繰り返し訪れ負荷の平坦化が図られる。64 多重の割付では、単純割付に比べ 3 倍高い性能を示した。このとき 64PE を用いた実行時間は約 4 秒である。但し通信の局所性は、単純割付がもっとも高く、多重度が上がるほど局所性が低下して、通信オーバーヘッドが上昇する。64 多重が性能のピークで、さらに多重度を上げると、性能は低下してくる。負荷の均等分散と通信の局所化のトレードオフが性能を決定する分かり易い例である。

7.1.5 開発経験

これらのプログラムは、KL1 のソースプログラム行数で 1500 から 8000 行程度の大きさであり、初版を開発するのに、プログラム当たり 1 ないし 2 名が 3 か月かかった。並列の複雑なプログラムをこの程度の期間で開発できたことについて、驚く人も少なくないが、KL1 を使った感触は「思いのほか使いものになる」ということである。オブジェクト指向のモジュール化や継承の機能を入れればもっと書き易くなるとの声もあるが、予想以上に何でも書けてしまう。むしろ問題は、書くところではなくて、問題を並列向きにモデル化し効率の良いアルゴリズムを設計するところのようである。たくさんのプロセスが通信し合いながら問題を解くモデル (並行オブジェクトモデル) は、高い並列性を実現したいときに利用価値が高い。KL1 は、このようなモデルに基づいたプログラムが、極めて自然に記述できる。また実行優先度制御の機能は、詰め碁の項で述べたように見込み計算を生ずる問題を中心として、処理効率の制御に威力を発揮している。さらに負荷分散指定の pragma が、問題解法のアルゴリズムの記述とは独立して着脱できることにより、同一の問題で異なる数種の負荷分散方法を容易に実験することができた。性能のチューニングに有り難い言語機能である。また並列 OS、PIMOS の開発で特に印象的だったのが、通信や同期に関わるバグが皆無といってよいほど少なかったことである。KL1 が通信と同期に関してユーザーを救う並列処理用高級言語であるとの感触をさらに強く持つに至った。

7.2 本格的並列応用プログラムの開発

4 つの実験的プログラムの開発に続き、より本格的で難度の高い並列応用プログラムの研究開発も始まっている。並列処理の知識と同時に、問題領域の深い知識が必要とされる。その一端を紹介する。

1. **LSI CAD**：LSI 設計の下流工程には、多大の計算を必要とする問題がいくつもある。その中から、次の 3 種を並列プログラム化し、評価を始めている。
   - バーチャルタイム方式に基づく論理シミュレーション — 高速化を目指す。
   - シミュレーテッドアニーリングを用いたブロック配置 — 解の品質向上を目指す。
   - 線分探索法を分散アルゴリズム化した LSI 配線 — 高速化達成ののち解の品質向上へ。
2. **遺伝子情報処理**：DNA や蛋白質の塩基配列から類似性を見つけ出す処理は、それらの機能を解析する基本であり、莫大な計算を必要とする。並列処理でこれを高速化すると共に、知識処理の助けを借り計算量を削減する。DNA の元データや計算結果は、データベースとして蓄積・利用される。
3. **法的推論**：事件に対する法律の適用及び解釈の事例ベースを持ち、事例に基づく推論によって、新たな事件の審理をガイドする。解釈によって異なる結論に到達し得る過程のすべてを並列処理で求める。
4. **囲碁対局**：囲碁の対局を行うプログラムは、探索空間の巨大さゆえに、チェスのような力尽くの探索で処理できず、多様な知識を複合的に使用する。異種の知的処理を追加してゆくとき、処理時間の増加を吸収するために並列処理を使う。

7.3 本格的並列プログラムの研究開発事例 — LSI 配線プログラム —

LSI の設計にはたくさんの工程があるが、その 1 つに、トランジスタなどの素子を LSI 表面に配置して、結線の

仕方を設計する仕事がある。ここでは、現在進めている「配置配線の自動設計」の並列処理について紹介する。

LSI チップ1個の上のトランジスタ数は、マイクロプロセッサですでに100万個を越え、規模の拡大につれて、計算機による設計時間も増加の一途をたどっている。特に配置配線の設計などでは、計算機の処理時間は素子数に比例するのではなく、その2乗に比例することもあって、それをいかに短縮するかが重要課題になっている。

また、高い品質が要求される場合(つまり狭い面積に多くの素子を詰め込む場合)、いまだに人手の介入を必要とすることが多い。このことからもわかるように、計算機で自動設計する場合の、品質の向上が強く要望されている。

このように、設計時間の短縮と設計品質の向上の2つの面で、並列処理と知識処理への期待が高まっている。われわれは、まず並列処理の側からの取り組みを開始し、処理時間短縮を目的に、LSI配線のための並列プログラムを開発した[5G シンポ 90]。

7.3.1 LSI 配線問題の並列向きモデル化

並列処理のためのプログラム作りには、幾つかの重要なポイントがある。ここではそれを2つに分けて説明しよう。

まず前半の話である。LSI配線プログラムは、10年以上前から逐次計算機の上で作られ改良を続けられており、数種類の異なる配線方法に分類される。この中の適当なものをKL1の様な並列言語に書き換えれば並列処理ができるかというと、一般に書き換えだけで取り出せる並列性(並列実行可能性)は僅かである(普通、数台のプロセッサでしか並列実行できない程度)。これは、従来使用されていた配線アルゴリズムが強い逐次性(順番に実行しなければ解が得られない、また順番に実行することを利用して計算の効率を高めるような性質)を持つためで、逐次計算機の上で作られたアルゴリズムの多くはこのような性質を持つ。

そこで1000台規模のプロセッサを用いて並列処理したいと思うと、アルゴリズムの設計をやりなおすことになるが、同時に問題のモデル自体を見直した方が良い場合も出てくる。つまり並列性が出易いように問題のモデルを作り直し、その上で並列に計算ができるアルゴリズムを組み立てる。

図11 はLSI配線の例である。ブロックは既に設計の終わった回路の固まりであり、周辺に接続端子を持つ。この例では3個のブロックが配置されており、配線問題は、各ブロックの端子のうち同じ名前(番号)の端子どうしを全て接続せよ、という問題である。配線はあらかじめ決められた格子上でのみ可能である。配線層は2層あり、第1層は縦方向の配線のみ、第2層は横方向の配線のみが許される。第1層と第2層の間はビアホールと呼ばれる穴で接続される。図11 では、すべてのネット(異なる名前を持つ端子に関する配線のそれぞれ)が、接触することなく配線を完了している。

このような配線のプログラムは、逐次計算機上では、

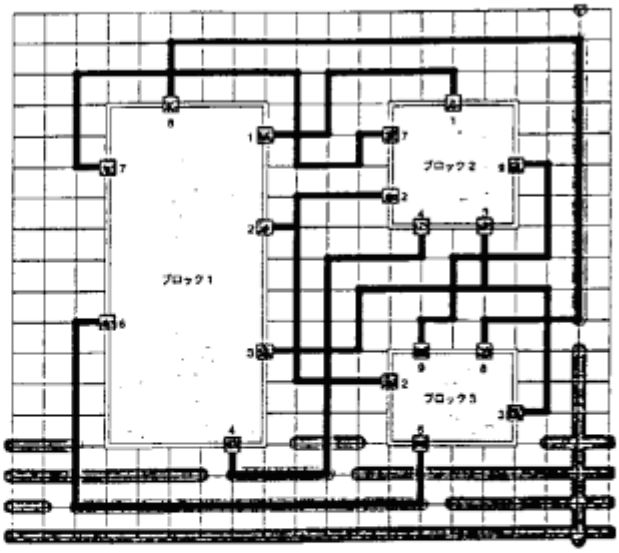


図 11: LSI配線問題とモデル化

格子点の数だけの配列を取り、その配列上に配線結果の値を書き入れたり、すでに書いてある配線情報を見回したりしながら、配線処理を進める。ところが並列処理では、複数のプロセッサからこのような配列を参照、更新しようとすると、配列操作がボトルネックとなり、十分な並列性は実現できない。そこで今回は、以下のような問題のモデル化を試み、その上で分散化された並列アルゴリズムを組み立てた。

まず並列性の高いモデル化の可能性を考慮しながら、基本的な配線手法として、線分探索法と呼ばれる方式の拡張型を選択した。これはちょうど、配線格子の上を伸ばせるかぎり線を伸ばしてみて接続の可能性を試してゆくようなものである。すなわち線分があらゆる処理の単位となる。そこであらゆる線分1本1本を自ら能動的に活動できるオブジェクトとし(図11 中の太めの着色部分)、交叉しているオブジェクトは通信路を持って、オブジェクトがメッセージ通信しながら配線経路を決めて行くというモデルを作った。未配線線分の一部が配線に使われると、オブジェクトは動的に複数オブジェクトに分かれる。あらゆる線分オブジェクトが同時に配線経路探索の作業を並行して実行できるモデルであるから、複数のネットについて同時に配線を進めようとするときに高い並列性が期待できる。このモデルに基づいて、線分探索アルゴリズムを並列向きに設計し直した。これらは、オブジェクトをKL1のプロセスとして実現することにより、容易にプログラム化できる。

7.3.2 複数のプロセッサへの割り付け

問題を解くアルゴリズムの中で表現された並列性は、まだ並列実行の可能性でしかない。これをどう複数のプロセッサによる並列実行に結び付けるかが次の課題である。たとえば平均して10万の並列性が期待できる問題を1000台のプロセッサを有する並列計算機で実行しても良いし、100台の処理装置の計算機で実行してもよい。

前述のモデル化では、オブジェクトが並列処理の単位であるが、それらをプロセッサが遊ばないようにうまく配置してはじめて、並列処理により処理装置の台数に応じた性能向上、そして実行時間の短縮が実現できるのである。これは思いのほか難しい。オブジェクトはいつも仕事をしているわけではなく、忙しくなったり暇になったりしている。全てのプロセッサが均等に忙しくなるように仕事(この場合オブジェクト)を割り付けることを負荷の均等分散という。一方、頻繁に通信し合うオブジェクトもいれば、通信のほとんどない相手というのも存在する。通信頻度の高いもの同士は近くに配置しておかないと、プロセッサ間の通信路が混んで並列実行のネックとなる。これを通信の局所性を保つ、という。この2つはしばしば対立する要求となるのでたちが悪い。

利用者の側はこんな期待をする。問題を解くための並列性の高いアルゴリズムをプログラム化した後は、その中の仕事をどうプロセッサに割り付けるかはシステムが勝手にやってくれると有り難い。ところが、異なる特性を示すあらゆるタイプの問題にたいして、負荷の均等分散と通信の局所化を常に同時に満たすような仕事割り付けの方法は、いまだかつて誰も考え付いたことがない。問題のタイプを限定しないかぎり、システムが自動的にそれを行うことは難しいとも考えられている。

そこで現在はまだ、アプリケーションを書く人とシステムを作る人が協力して、問題毎に最適な負荷分散や通信の局所化の方法を考えて、プログラムのなかに書いている。将来はいろいろな問題で試されたそれらの方式を一般化し、システムプログラムの中でサポートして、プログラム毎に考える苦労を軽減しようとしている。このように実験的によい負荷分散方法などをみつけ出してゆくとき、KL1の機能の中で、問題解決のアルゴリズムと、仕事をプロセッサに割り付ける部分が、うまく分離してプログラム化できる点は、他の言語では得難い長所である。

現在の並列配線プログラムでは、通信頻度の高いオブジェクトをグループ化し、それを単位としてプロセッサにランダムに割り付けている。処理装置台数に対する性能向上はまだ改善の余地はあるものの、並列推論マシンの実験機であるMulti-PSIを用いて、汎用大型計算機上の配線プログラムと同等以上の性能を得るところまできている。並列推論マシンPIMの上では、さらに数十倍の性能向上を期待して、プログラムの改良を続ける予定である。同時に、負荷分散方式などの一般化、ライブラリとしてのシステムへの組み込みなどにも注力してゆく。

8 おわりに

第五世代コンピュータ研究開発プロジェクトでは、知識処理技術を研究開発するための記述言語として、汎用の並列処理言語KL1を設計し、KL1を高速実行するためのハードウェアとして、並列推論マシンPIMを開発してきた。そのアプローチは、知識のハードウェア化、もしくは知識処理アルゴリズムのハードウェア化とは異なる方向であるが、より高度な知識処理をより高速に実行するという目的において、共通するものを持つ。

並列推論マシンPIMが完成すると、知識処理を始めとするいろいろな応用について、汎用大型計算機の数十倍から数百倍の性能が得られると期待される。このようなマシンは、現在の実装技術では、まだ筐体8本から16本の巨大サイズとなるが、1990年代後半にはプロセッサ1000台を内蔵する未来のPIMが、筐体1本に実装可能となり、ワークステーションや、計算機制御の目的にも、その高速処理性を利用可能となってくる。

このような大規模並列計算機を効率良く運転できるかどうかは、ひとえに並列ソフトウェア技術の研究開発の進展にかかっているわけで、本稿でご紹介したような並列プログラムの開発をとおして、数々の技術と経験を積み上げているのである。このような技術蓄積には、個人的見解ではあるが、まだ10年あるいはもう少しの研究継続が必要と考えている。その過程で、現時点では不十分な、リアルタイム性や耐故障性の検討も、なされてゆくことになろう。

参考文献

[瀧 91] 瀧 和男：大規模汎用並列処理の実現に向けて－ICOTにおける研究より－, bit Vol.22 No.2, 共立出版, 1991年2月

[内田 91] 内田俊一 他：特集「第五世代コンピューター」, 日経サイエンス1991年2月号, 日経サイエンス社, pp.8-19

[Taki 89] K.Taki : The FGCS Computing Architecture. INFORMATION PROCESSING 89, G.X.Ritter(ed.) (IFIP Congress '89), North-Holland, pp.627-632. ICOT TR-460.

[Goto 89] A.Goto : Research and Development of the Parallel Inference Machine in the FGCS Project, TR-473, ICOT 1989.6.

[古川 87] 古川康一 他：並列論理型言語GHCとその応用, 共立出版, 1987.

[宮崎 87] 宮崎敏彦：PDSS使用手引き, TM-437, ICOT 1987.

[服部 89] 服部彰 他：並列推論マシンPIM/pのアーキテクチャ, 情処論 Vol.30, No.12, 1989, pp.1584-1592. ICOT TR-453.

[佐藤 89] 佐藤裕幸 他：PIMOSの資源管理方式, 情処論 Vol.30, No.12, 1989, pp.1646-1655. ICOT TM-815.

[Taki 88] K.Taki : The Parallel Software Research and Development Tool : Multi-PSI System, Programming of Future Generation Computers, K.Fuchi and M.Nivat (Editors), North-Holland, 1988, pp.411-426. ICOT TR-237.

[Nakajima 90] K.Nakajima, et al.: Evaluation of Interprocessor Communication in the KL1 Implementation on the Multi-PSI, TR-531, ICOT 1990.

[Ichiyoshi 89] N.Ichiyoshi : Parallel Logic Programming on the Multi-PSI, TR-487, ICOT 1989.10.

[寿崎 89] 寿崎かすみ 他：マルチ PSI における並列構文解析プログラム PAX の実現および評価，並列処理シンポジウム JSPP'89 予稿集，pp.343-350，情報処理学会，1989.2. ICOT TM-659.

[沖 89] 沖 賢明 他：マルチ PSI における並列詰め碁プログラムの実現と評価，並列処理シンポジウム JSPP'89 予稿集，情報処理学会，1989.2. ICOT TM-658.

[古市 89] 古市昌一 他：疎結合並列計算機上での OR 並列問題に適した動的負荷分散方式とその評価，情処研資 89-ARC，1989 年 11 月. ICOT TR-517.

[Furuichi 90] M.Furuichi, et al. : A Multi-Level Load Balancing Scheme for OR-Parallel Exhaustive Search Programs on the Multi-PSI, Proc. of Second ACM SIGPLAN Symposium on Principles & Practice of Parallel Programming PPoPP, March 1990, pp.50-59. ICOT TR-526.

[和田 89] 和田久美子 他: マルチ PSI 上の最短経路問題の実現と評価，情処 研資 89-ARC，1989 年 11 月. ICOT TR-520.

[5G シンポ 90] 並列版 LSI-CAD 実験プログラム (1) - LSI 配線，第 8 回第五世代コンピュータに関するシンポジウム デモンストレーション説明資料，ICOT 1990.6, pp.1A(1)