TR-616

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 －

瀧　和男

February, 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

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

瀧　和男

1. はじめに

大規模汎用並列処理とは，大規模並列計算機を広範な応用領域で，効率よく思い通りに使いこなすこと，そしてそのためのソフトウェア技術とハードウェア技術一式のことである．

西暦2000年には，LSI 1チップ上のトランジスタ数は1億個に達する．この技術を使えば，高性能プロセッサ1000台を含み，今のワークステーションの1万倍のピーク性能をもつワークステーションが，机の脇に置けるようになる．このようなハードウェアの実現性は，今や約束されたも同然である．問題なのは，このようなスーパーハードウェアを今のワークステーション同様に使いこなせるかどうかである．そのためのソフトウェア実現への道程は，まだ明らかとはいえないのである．

本稿では，第5世代コンピュータプロジェクトのうち，並列処理の面にスポットを当てる．そして，進められている並列処理の研究を上に述べた「道程」の一つのメインストリートと捉え，研究の方向性と現在の研究状況，そして未来への展望を語りたいと思う．

2. 第5世代コンピュータプロジェクトと並列処理

研究の枠組み

第5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することを目的として通産省の主導の下に進められている．1982年に開始された10年計画の大規模プロジェクトで，AIブームの火付け役となったり，海外に同様のプロジェクトを数多くスタートさせるきっかけとなったことで，ご存じの方も多いことであろう．ただし，「Prologと AIのためのプロジェクト」ではない．

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，プログラミングのわずら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ときに，脇にいて助けてくれる賢い助手」である．たとえば，わたしが計算機の設計をするとき，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ときには，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至る道程の，今はまだ中ほどを進んでいるとお考えいただきたい．

このような高度知識情報処理を実現するための技術要素をごく簡単に表現するならば，マシンを賢くするための技術としての**知識処理**と，知識処理が必要とする膨大な計算能力を提供する技術，すなわちマシンを速くする技術としての**並列処理**の二つとなる．通産省からの委託でプロジェクトを推進している（財）新世代コンピュータ技術開発機構（略称 ICOT：アイコットと読む）では，プロジェクトの開始に当たって，研究開発の枠組みを図1のように考えた．知識処理の研究と並列処理の研究という，ともにたいそう難しいであろう二つの研究テーマを遊離することなく並行して進めてゆくためには，

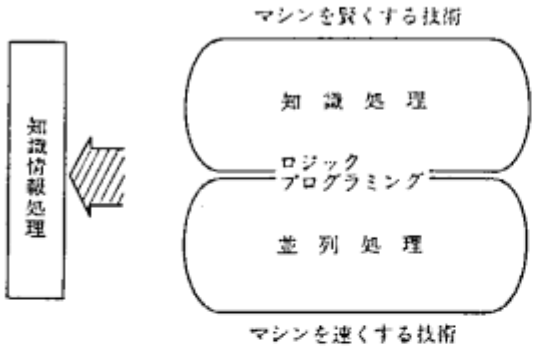


図1 第5世代コンピュータ研究開発の枠組み

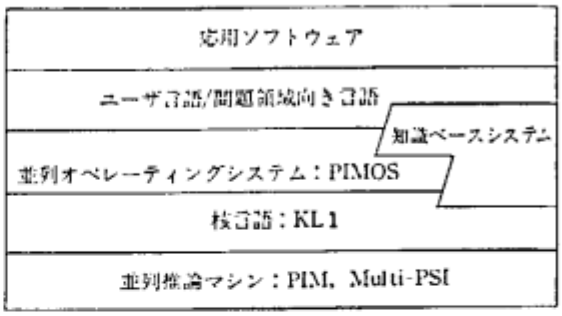


図2 第5世代コンピュータ・プロトタイプ・システムの構成

それらのインタフェースになるとともに，理論的にしっかりした共通の研究基盤がぜひとも必要である．その役割を担うために選ばれたのが，ロジックプログラミングであった．これは，分析と議論の末の帰結であるとともに，ICOTの先人たちの鋭い洞察によるところが大きい．ただし，ここでロジックプログラミングと呼んでいるのは，Prologという単一の言語のことではなく，もう少し幅広く，一階述語論理に理論的基礎をおく計算機言語の一群とその周辺技術を指している．

こうして，ロジックプログラミングをベースとした並列処理が，ICOTの研究開発の一つの柱として位置づけられ，研究が続けられてきたのである．研究の進展につれて，その難しさと重要性への認識は高まり，今や当初の予想と期待を超えた主要な研究開発テーマになった．また研究の内容について見ると，実はロジックプログラミングに特化した並列処理技術というものの比率は結果として低く，研究開発されている技術の多くが，広く並列処理一般に通用する，あるいはそれらをリードする最先端技術となっている．

プロトタイプ・システムの構成

並列処理に重点を置いて見た場合，第5世代コンピュータ・プロトタイプ・システムの構成は図2のようになる[1]．最下層はハードウェアの層である．プロジェクトの最終ターゲットとなる並列推論マシンのPIM，そしてPIMの実験機であり，すでに稼働中のMulti-PSIはここに入る．すぐ上の層が，核言語KL1の層である．KL1は並列処理のために設計された論理型言語であり，ハードウェアとソフトウェアのインタフェースとなる．一種の高級言語マシンシステムであるため，核言語の層はOSよりも下にきている．その上に，並列オペ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とほぼ同層にある．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，プログラマの負担軽減のために，ユーザ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もここに含まれる．

ICOTでは，これらすべての層にわたって，まったくのゼロに近いところから研究開発の積重ねを行なってきた．それほどまでに徹底した理由は，ロジックプログラミングに基づいて，ハードウェアからOS，応用ソフトウェアに至るまで，並列処理に関するすべてを再構成することにより，従来技術をひきずりながらの研究では得られないような，飛躍的な前進があり得るとの信念によるものであった．それともう一つ，過去20数年にわたる数多くの並列マシンの失敗の歴史を振り返るとき，重大な失敗の原因の一つが，並列のソフトウェアに関する研究環境を整備しきれなかったためとの分析によるものであった．したがって，今回のプロジェクトでは，ハードウェアとソフトウェアが協調しながら，互いに正のフィードバックをかけ合って次第に成長していくことができるような研究開発のしかけを作ることに大きな努力が払われた．すでに研究成果について十分な手応えが感じられるが，このようなアプローチに対する客観的評価は，これから後10年程度の間に固まることになろう．

始めに言語ありき——核言語 KL1——

新世代の並列処理のために，ハードウェアの研究とソフトウェアの研究を何もないところから同時に並行してスタートすることは容易ではない．どちらから先に手をつけるべきか，鶏と卵のようなものである．そこでまず，両者のインタフェースとして中核となる言語を設計し，それを足がかりとして，下側のハードウェアと上側

のソフトウェアの両サイドの研究を膨らませてゆくことにした．その言語が，核言語 KL 1 である．

KL 1[1] は，以前に本誌でも紹介されたことのある論理型言語 GHC[2] の拡張版である．正確には，ガード部分でユーザ定義述語の呼出しを許さないフラット GHC に基づいている．拡張部分はおよそ次の通りである．

（1） メタコールと実行管理の機能を合わせもった「荘園」機能，exception 取扱い機能などの OS 向け機能
（2） ゴールの実行プライオリティおよびプロセッサへの割付け制御のための pragma
（3） ベクタおよびストリングデータ型
（4） 各種組込み述語
（5） マクロ展開機能

（1）の OS 向け機能を除くと，ユーザがプログラムを組む際の考え方は，基本的には GHC と同じと思ってよい．ただし，一つだけ新しい概念として（2）の pragma が追加されている．これはスケジューリングやプロセッサ割付けを記述して，並列実行時の性能をチューニングするための機能で，プログラムの意味（問題解法のアルゴリズム）とはほぼ直交した関係にある．プログラムの本体を書いたあとで，必要なゴールに付加して使用する．

KL 1 プログラミングの特徴を述べる前に，言語そのものについて簡単に説明しておこう．プログラムは，次に示すような形の節をならべたものである（この範囲では GHC とまったく同じ）．

H :-G 1，…，Gm | B 1，…，Bn.

H をヘッド，G 1，…，Gm をガードゴール，B 1，…，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もち，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わかりやすい説明をしよう．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もつ．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一つだけ決まる（複数の節が成功した場合は処理系が一つを選ぶ）．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):- ture | Y=1.
fact(X,Y):-X>0 | sub(X,1,X1),
fact(X1,Y1),mult(X,Y1Y).
```

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なわれる．第1引数にまだ値が決まっていなければ，待ちに入る．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なわれる．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入る．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もたないと，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．値がゼロ以上なら，条件判定に従ってどちらかの節がコミットされる．たとえば，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．

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 X 1 を通して 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つ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入ることになる．KL 1 (GHC) では，このようにプログラマ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，KL 1 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 pragma について見ておこう．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 という三つのプロセスをフォークして，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，値を次々に生成しては filter に送る．このと

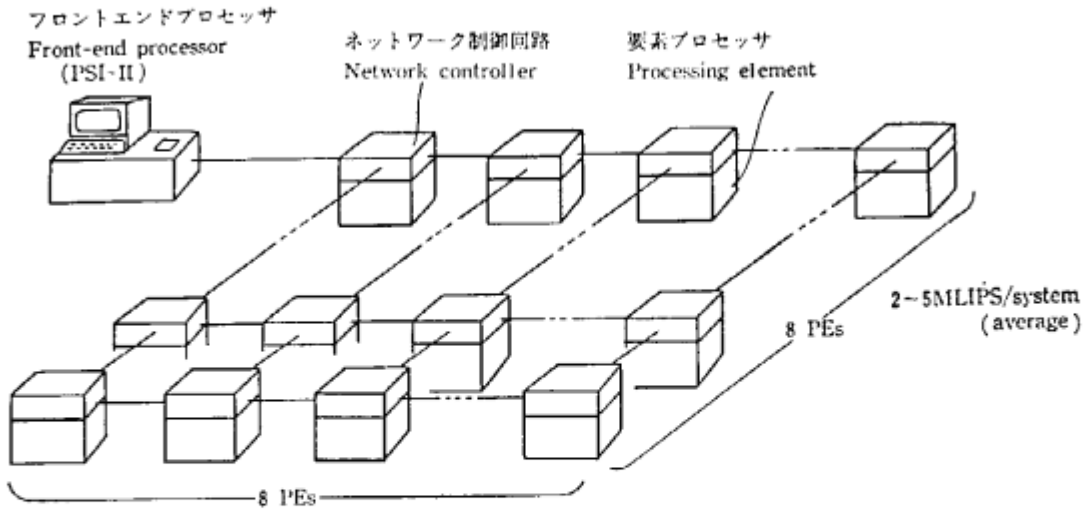


図 3　Multi-PSI/V 2 の構成

き変数Xはストリームとして使われる．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 は次々に送られる値から答えを計算し，Out に返す．プログラマ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かかわるバグは除かれる．

次に並列実行のために pragma をつける．

```
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).
```

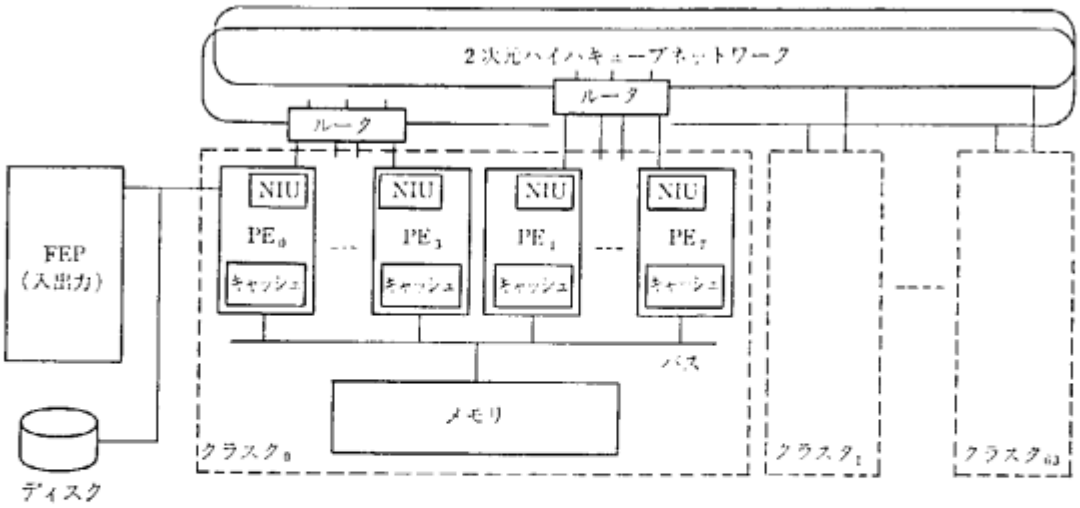
pragma をつけ替えることで，負荷バランスなどの性能にかかわる調整が，プログラムの意味を変えることなく簡単にできる．また実行優先度については，たとえば4096段階というように細かい指定を許しており，ユーザプログラムの中で処理効率向上のために使われることを意図している（4節の「詰め碁」の項参照）．

プログラマにとっての KL 1 の特徴を整理しておこう．(1)並列がベースとなった言語である（従来の逐次言語に通信と同期をもち込んだものではない）から，並列性の高いプログラムを書くのに適する．(2)通信と同期に関して超高級言語である．(3) pragma 機能は，負荷分散の実験や性能のチューニングに好適である．(4)バックトラックによる自動探索機能がない分，Prolog に比べて「推論言語」らしさは減少し，汎用並列言語に近づいた．(5)一階述語論理の枠を守っているので，プログラム変換などに対応しやすい．手短にまとめるならば，KL 1 はロジックから生まれた並列処理のための汎用言語ということができよう．

ハードウェア（その1）——Multi-PSI——

Multi-PSI は，KL 1 がある程度の性能で本当に並列に走る環境を少しでも早くプログラマに提供できることを目的に開発された実験マシンである．それと同時に，KL 1 の分散処理系の研究を進めるための重要な道具でもあった．Multi-PSI は，1990年秋現在，大小合わせて約20システムが ICOT と関連の研究機関で並列ソフトウェアの研究に使用されている．

図 3 に Multi-PSI の構成を示す．これは現在稼働中の2機種めの Multi-PSI で，最大構成時の図である．64台の要素プロセッサ (PE) が，縦横8台ずつの2次元格子状に接続されている．それぞれの要素プロセッサは，逐次型推論マシン PSI-II の CPU 部分を転用したものである．ネットワーク部分は新規開発した．また，マイクロプログラムは KL 1 用に入れ替えた．機械語命令として，Prolog の場合の WAM コードに相当する，KL 1-b と呼ぶ命令セットを使用している．ネットワー



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 4　PIM/p の構成

クは可変長のメッセージ交換を行ない，worm-hole routing と呼ばれる経路制御方式をとっている．各メッセージには，行き先プロセッサ番号が示されており，ネットワーク制御回路ではこの行き先を調べてメッセージを目的地まで送り届ける．格子接続の端に4台までのフロントエンドプロセッサを接続でき，入出力装置として使用する．

ネットワーク上ではメッセージ通信が行なわれるが，すでに述べたように KL 1 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．変数を共有するゴールの一方が，pragma によって別の PE に移されたとしても，変数の参照関係は KL 1 処理系が自動的に維持し，PE 境界をまたがる変数読出しが発生すると，処理系が通信メッセージを組み立てて送出する．Multi-PSI の KL 1 処理系は，各 PE 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．これはひとえに，すでに述べた KL 1 言語のよさを損なうことなく利用者に提供するためである．

性能面について触れよう．1台の PE は200ns のサイクルタイムで動作し，KL 1 の append プログラムの性能は約 130 キロ LIPS (Logical Inference Per Second) である．これは MRB と呼ぶ実時間塵集めを含めた性能であり，通常の append では塵は出ない．ちなみに Multi-PSI では，簡略化した実時間塵集めと一括型のコピー方式の塵集めを併用して，一括型の発生頻度を抑えている．PE 当りのメモリ容量は16メガ語，システム全体では1ギガ語を有する．応用プログラムの平均的な LIPS 値性能は，append の3分の1から5分の1であり，その64倍がシステム全体でのピーク性能となる．2メガ LIPS を超える実力がある．

ネットワークは，一つのチャネル当り行きと帰りの独立の通信路をもち，おのおの 5 MB/s の転送能力がある．また隣接ノードへメッセージ送出するとき，ノード一つ通り抜けるための遅れは，1マイクロ秒以下となっている．現在のところ，ネットワーク通信のネックはメッセージの組み立て分解のところであり，ハードウェアの転送能力には十分余裕がある．これはすなわち，利用者から見ると，2次元格子接続よりは完全結合のネットワークに近い特性として見える．二つの KL 1 プロセスの間で通信をするとき，それらが同じ PE 内にあるか別 PE かで，通信コスト（時間）が20倍程度異なる．この値は，負荷分散や通信の局所化を検討するとき，重要な指標となる[5]．

ハードウェア（その2）——PIM/p——

PIM は，プロジェクトの最終成果となるハードウェアで，Parallel Inference Machine の頭文字をとって現在数種類の異なるモデルを開発している．一つの代表的なタイプは，これから説明する PIM/p であり，もう一つは Multi-PSI を拡張したタイプである．

図 4 に PIM/p の構成を示す[6]．PIM/p は，Multi-PSI の一つの PE を一つの密結合クラスタに置き換え，ネットワークを2次元格子からハイパキューブにかえたものと考えることができる．クラスタは，8台の PE

が共有メモリをもち，共有バスにより密結合されたものである．ユーザからは，クラスタは非常に計算能力の高い一つのノードと認識される．したがって，クラスタ間の負荷分散は Multi-PSI の場合と同様，pragma を用いてユーザが制御するが，クラスタ内の 8 台の PE 間では，処理系が並列実行可能なゴールを配分して自動的に負荷のバランスをとる．クラスタ間接続のバンド幅を広げるため，同じハイパキューブネットワークを 2 組使用し，クラスタ内の 4 PE ずつがそれぞれのネットワークに接続される．また，各 PE は SCSI のインタフェースをもち，多くのディスク装置およびフロントエンドプロセッサを接続することができる．

PIM/p では，最大で64個のクラスタを接続予定である．このとき PE の数は合計で512となる．PE の単体性能は，Multi-PSI の場合の 3 ～ 4 倍で，PE 数は 8 倍だから，全体のピーク性能では Multi-PSI 比24～32倍ということになる．PE は新規開発の VLSI プロセッサで，約 60 ns の命令サイクルタイムをもつ．4 段のパイプライン制御を行なっている．マシン命令は RISC 技術に基づき，他に KL 1-b 実現のために，内部命令メモリへの効率のよいサブルーチン呼出し機能を備えている．PE 当り64Kバイトのコヒーレントキャッシュメモリをもち，クラスタ当りの共有メモリ容量は32メガ語である．

PIM/p では新たに，共有メモリに適した KL 1 処理系を開発するとともに，コピー方式に基づく並列の塵集めなども実装する．

PIM/p で採用した 2 層構造，すなわち多くのクラスタがメッセージ交換用の上位ネットワークで相互接続された構造は，将来の大規模並列マシンの一つの代表的な姿になると期待している．これは分散メモリ型並列マシンの一形態であり，大規模システムへの拡張性を考えると極めて自然な解と考えられる．一方，下層構造としての密結合クラスタは，次の 2 点で意味をもつ．一つはクラスタ内の 8 台の PE が，高レスポンス・高スループットで結合された部分構造を成すことで，クラスタ内に限ってはある程度粒度の小さい並列処理が効率よく実行できることである．これは，適用できる応用の幅を広げるのに効果があろう．もう一つは，クラスタ構造が PE 当りのメモリ量を減少させるのに役立つことである．簡単にいうと，Multi-PSI のように PE が独立のメモリをもつ場合，PE ごとにメモリ消費が異なって有効利用されないメモリが出るのに対し，クラスタ内の PE 間ではメモリを融通し合えるので，メモリのむだを減少させる効果が期待できるのである．Multi-PSI にしろ PIM にしろ，ハードウェアの実装体積中に占めるメモリの比率は高く，PE 当りのメモリ量減少は，小型化を考えるときに重要である．特に，数値計算に比べて処理の不均質性の高い記号処理を行なう場合，PE 当りに必要とされるメモリ量が大きくなることに注意が必要である．

並列オペレーティングシステム——PIMOS——

PIMOS（パイモスと読む；PIM の OS の意）は，PIM と Multi-PSI に共通の並列オペレーティングシステムである[6]．並列ソフトウェアの研究開発を行な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かけるバックエンド OS ではなく，それ自体で完結したスタンドアロン OS である．また 1 個の OS が並列マシン上に分布して存在するように作られており，要素プロセッサごとに独立の OS を置くような分散 OS ではない．PIMOS はすべて KL 1 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．

PIMOS の機能を説明する前に，KL 1 と PIMOS の関係で特徴的なところを述べておこう．KL 1 言語のよ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 KL 1 言語処理系の中で実現している．

（1） ユーザ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
計算資源の割当ても荘園単位に行なっている
（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当てと解放
（4） PE ごと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．

KL 1 プログラムでは，ゴールの 1 個 1 個をプロセスと考えることもでき，これが最小粒度となる（Multi-PSI の場合，最小で10マイクロ秒を下回る）．一方，境界を越えてプロセスを生成する場合の粒度の考え方は，生成したプロセスまたはその子孫が，結果を返してくるか新たな PE 間通信を発生するまでに，どれだけの時間走るかで計る．Multi-PSI の場合，このような PE 間における粒度の実用的最小値は，100から200マイクロ秒である．これを下回ると，通信の手間（前処理，後処理の

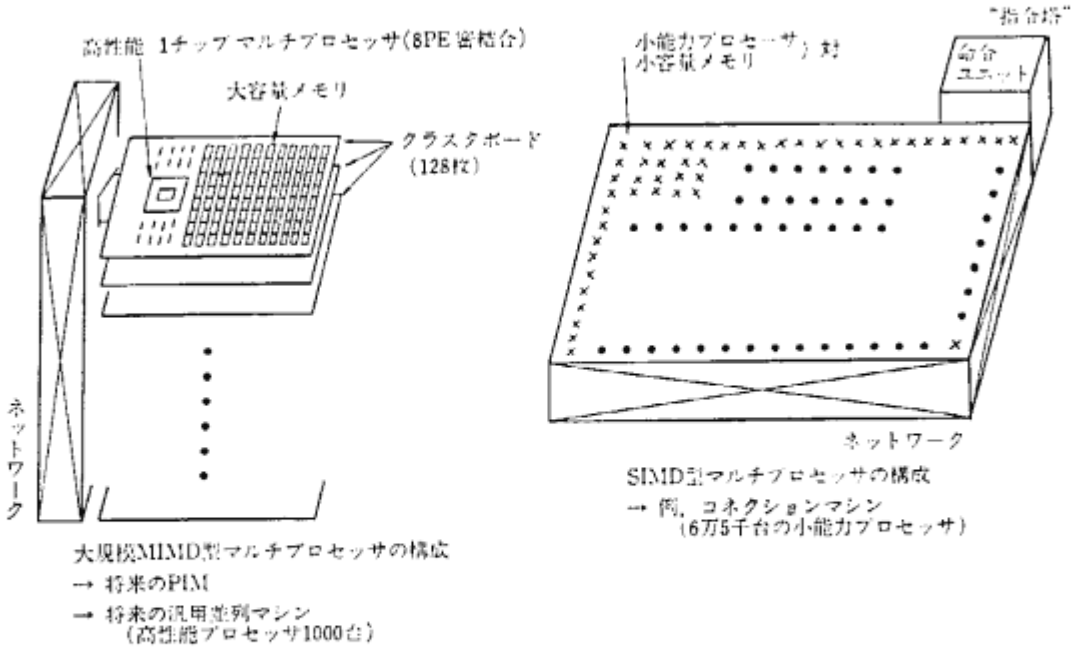


図 5 大規模 MIMD 型マルチプロセッサ 対 SIMD

時間）のほうが余計にかかるようになり，別 PE で実行する価値がなくなる．それでもこれは，従来常識からすると十分小さな値であろう．

処理系が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
- シェル機能をはじめとするユーティリティ
- コンパイラ，並列処理用トレーサ，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などの各種ライブラリ

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占めている．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．

† MIMD と SIMD は並列計算機の命令実行方式の分類である．MIMD: Multiple Instruction stream, Multiple Data stream ; SIMD : Single Instruction stream, Multiple Data stream)

3. どんな並列処理を目指すのか

KL 1 は汎用の並列言語を指向していること，クラスタがネットワーク結合された PIM/p のような構造は，将来の大規模並列マシンの一つの姿と期待されることについては，すでに述べたとおりである．ここでは，ICOT の指向する並列処理の研究の目指すものと，ICOT の研究ではカバーしていない領域とを対比の上で見ることにしよう．

分散メモリ型大規模並列マシン上の汎用並列処理 —— MIMD 対 SIMD ——

PIM/p も Multi-PSI も，分散メモリ構造をもつ MIMD 型並列マシンである†．ネットワーク接続された計算ノード（PIM/p ではクラスタ，Multi-PSI ではPE 1個）の間ではメッセージ通信が行なわれる．各計算ノードは十分に高い計算性能と，大量のメモリをもっている．MIMD 型並列マシンが対象とすべき処理の特徴は，SIMD 型並列マシンと比べることで鮮明となる．

1988年に第5世代コンピュータ国際会議（FGCS'88）を開いたとき，はじめて 64 PE の Multi-PSI を出展した．このときある新聞は，「ICOT の作ったのは64台接続のマルチプロセッサでしかないが，世の中にはすでに6万5千台もつながっているものもある．」という記事を載せたことがある．この6万5千台つないだのが SIMD

マシンであった（よく知られているコネクションマシン）．SIMDマシンの特徴の一つは，図5に示すように，小能力のプロセッサと小容量のメモリの対が，数多く高速のネットワークで接続された構成をとることである．これは，計算能力をもつメモリが空間内に均質に分布していると見ることもできる．もう一つの特徴は，すべてのプロセッサはただ一つの命令ユニットから同期して同じ命令を受け取り，足並みを揃えて動くことである．このようなマシンは，均質なデータに対し均質な処理を定められた手順で実行するような用途に向いている．たとえば，密なマトリクス計算とか画像処理がそれに当たる．命令が共通だから，個々のプロセッサがもつ計算の途中経過に従って，受け取った命令を実行するか無視するかといった程度の単純な選択は可能であるが，分布して配置されているデータのそれぞれに依存して，処理アルゴリズムを大きく変えてゆくようなことは難しい．SIMDマシンが得意とする均質なデータ処理の中の並列性のことを「データパラレリズム」と呼ぶことがある．SIMDマシンは，データパラレルなアルゴリズムが存在する問題に限って，非常に高い性能を発揮する．

一方，MIMDマシンというのは，一つひとつのプロセッサがもっと賢くて自立している．高性能のプロセッサと大容量のメモリの組が，ネットワークで接続された構成をとり，個々のプロセッサは独立した命令列を実行する．SIMDで，均質なデータに対して同期的な処理が行なわれるのとは対照的に，MIMDでは，プロセッサが違えば異なる質のデータが置かれていて処理の質が違ったり，たとえ同質のデータに同質の処理が行なわれる場合でも，処理の進み方が異なって当り前との立場をとる．同期が必要なときだけ，プログラムで同期をとる．すなわち，本質的に処理の不均質さが生じることを前提としたシステムである．したがって，SIMDに比べて不均質なデータに対する不均質な処理のある問題や，実行時の計算負荷が動的に変化する問題などにも適用しやすいシステムといえる．知識処理はその代表である．プロセッサによって格納されている知識の種類が違うために処理アルゴリズムが異なることもあり，入ってくるデータに応じて駆動される知識や推論のための計算量が動的に変化する．このように不均質な度合いの大きい問題を実行する場合ほど，同期をとる際に待たされる可能性が高くなる．あるプログラムが待ちに入ったときにプロセッサが遊ぶようでは性能は出ない．そこでプロセッサをいつも忙しく保つためには，たくさんの仕事をプロセッサに与えておく必要があり，必然的にプロセッサに付くメモリも大容量でなければならない．

一般に，メモリ容量はプロセッサ性能に比例して大きい必要があるが，同時に対象とする処理の不均質さの度合いに応じて，大容量化が求められる．逆に，ネットワーク性能が高いと，同期の待ち時間の短縮や仕事の分配，再配置を高い頻度で行なえるようになり，プロセッサ当りに必要なメモリ量は減少する傾向にある．

図5の中の左の絵は，将来の大規模MIMD型汎用並列マシンおよび将来のPIMのイメージを表わしたものである．8プロセッサを1チップに集積したLSIを用いて，ボード1枚でクラスタを構成する．クラスタごとに大量の共有メモリをもつ姿が描かれている．クラスタボード128枚をネットワーク接続し，1024 PEからなるシステムを構成する．筐体1本ないし2本に実装する．

ICOTが目指しているのは，このようなMIMDマシンであり，大量の計算を発生する問題のうち，不均質なデータに対する不均質な処理や，動的処理の比率の高い問題を対象と考えている．知識処理はその中の主要な問題領域の一つであるが，その他にもSIMDマシンには向かないが非常に高い計算能力を要する問題ならば，知識処理の要素が少なくてもよい処理対象と考えている．SIMDでは効率よく扱えないような処理に対し，より広範囲に適用できるシステムを指向しているわけであり，それだけ難しい技術に挑戦しているともいえる．

並列処理でどのように速くしたいか

われわれが対象とするのは，MIMDマシンを用いてプロセッサ台数に比べてはるかに大きい問題を解かせるような場合である．このとき，N台のプロセッサを用いて最大でもN倍の高速化が図れればよい（Nの値は今のところ1,000程度）．問題の性質によっては，定数分のN倍だったり，Nの平方根倍でも当り前と考えている．

一方，これとは異なる考え方として，問題サイズと同程度あるいはさらに多数のプロセッサを使用して，問題サイズPに対する計算時間のオーダを下げてしまおうというアプローチがある．逐次実行でPの2乗かかる計算が，並列処理で$P\log P$に短縮できるというような議論がそれに当たる．実際にコネクションマシンでは，65,000台のプロセッサ数を問題サイズと同程度と考えてよい場合に限り，この議論が当てはまる．確かにオーダが下がるのはありがたく見えるが，そのようなアルゴリズムは限られており，Pの大きさも限定される．

N台のプロセッサでN倍しか速くならないのかといわれたら，大きな問題をMIMDで解かせるときに，それ

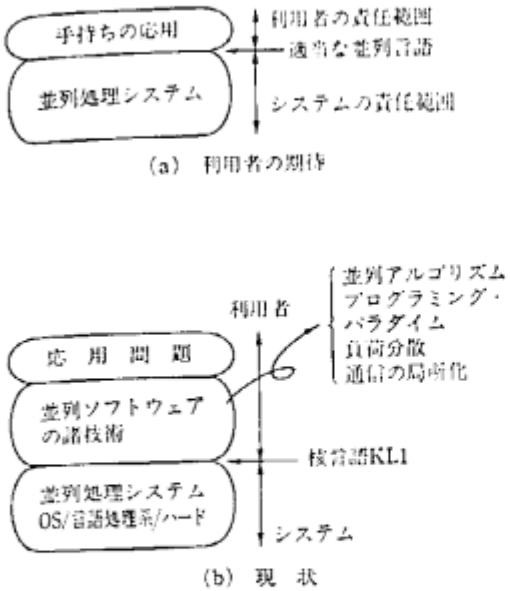


図 6 並列処理システムに対する期待と現状

以上の何を望めるのかと問い返すことにしよう．しかしながら，世の中にはもっと過激な人もいて，並列処理を使っても速くならないと主張することがある．よくよく尋ねてみると，十分なプロセッサを用いても，並列処理によって計算時間のオーダを下げるようなアルゴリズムが存在しない．だから並列処理では速くならないとの議論展開である．単に，よい並列アルゴリズムがないといっているのだと思うことにしよう．

ところが，N台のプロセッサをもってきて，N倍よりも速くなる場合がある．スーパーリニア・スピードアップと呼ばれる．たとえば，スケジューリングを調節することにより探索の枝刈り効率が改善され，計算量が削減できるような問題があったとする．N倍よりも十分速くなるのは次のような場合である．1台のプロセッサで，よくない擬似並列のスケジューリングをしていたために，枝刈り効率が悪くむだな計算をしていたことを知らなかったとする．これをN台プロセッサで実並列実行したら，たまたま非常によいスケジューリング状態が得られ，枝刈り効率が飛躍的に高まって，総計算量が著しく減少したとする．この場合は，N台並列実行による高速化のほかに，総計算量減少による時間短縮が加わるため，後者の分だけN倍よりも高速化したように見えるのである．このような場合は，実は1台プロセッサでの実行時間も，スケジューリングを改善することにより短縮可能なことがおわかりであろう．

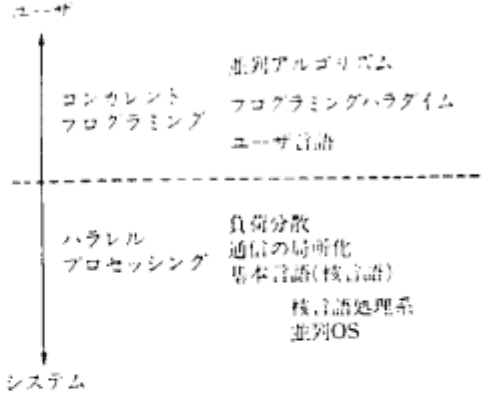


図 7 並列ソフトウェアの研究課題

4. 並列ソフトウェアの研究課題

期待と現状のギャップ

まず意識のずれの問題について説明する．ハードウェアからOSまで含めた層を並列処理のシステムの層と考えよう．利用者の立場からすると，図6(a)に示すような意識となる．並列の言語をインタフェースとして，手持ちの応用をその言語で書き下すと，システムが気持ちよくそれを並列実行してくれる．もちろん性能も上がる．そのような期待が，ICOT 設立当初にはシステム開発者にもあったし，ユーザの中には今でも存在している．この場合，手持ちの応用を書き下すところまでは利用者の責任範囲で，その言語で書かれたプログラムを効率よく動かすのはシステム側の仕事といった役割分担を想定していた．ところが，やってみるとどうも違う．そんな簡単な話ではなかったのである．ここ数年，並列処理の研究を続けてきて，図6(b)に示すように応用問題と並列処理システムの間には，並列ソフトウェアの諸技術とでも呼ぶべき厚い中間層があって，これがほとんど未解決になっていることがわかってきた．

並列ソフトウェアの研究課題

ここで並列ソフトウェアの諸技術と書いたものは，そのまま並列ソフトウェアの研究課題といえる(図7)．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よいアルゴリズムは，逐次処理であることを利用して高い効率を実現しているものが多い．アルゴリズムが強い逐次性をもつから，これを並列言語で書き下して無理矢理並列マシンにマッピングしても，たくさんあるプロセッサが頭に動いてしまうだけである．並列マシンを効率よく動かすためには，並列処理向きの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

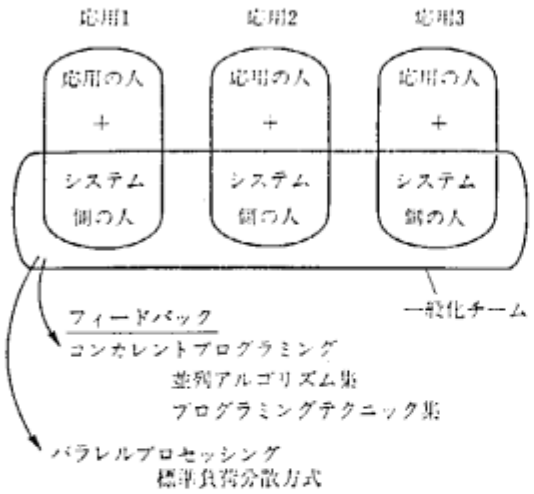


図 8　並列ソフトウェア研究の進め方

ばならないと考えるべきである．手持ちの応用を，言語だけを並列言語にかえて書き直せば済むという話ではない．ここで注意が必要なのは，並列性を多く含んだアルゴリズムに書き換えるとき，計算量のオーダが増えてしまうようなミスを犯しやすいことである．問題サイズ P に対し，$P\log P$ で計算できていたものが，P の2乗かかるアルゴリズムに化けたりする．これではいくら並列マシンをもってきても，それ以上に計算時間が増えることになり，よくない．コンスタント程度は目をつぶるとして，計算量のオーダを増加させないことが重要である．

次は**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しれないが，プログラミングの型，手本とでもいおうか．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が，これらの技術が蓄積されていない．つまり，お手本がないから問題を並列プログラムに落とそうとしても手が出ない．お手本の蓄積がぜひとも必要である．プログラミングの型を与えるような並列ユーザ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あたりの仕事は，最低限ユーザが行な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10万の並列性をもっていても，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．

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ありがたいが，まだその段階にははるかに至らない．パラレルプロセッシングにかか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付けを行なうのが**負荷分散**である．これは仕事をばらまく話である．一方で，部分問題の間で多くの通信を行なうものどうしは，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．これは仕事をばらまかずに固める話であり，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など，確立されてないものばかりである．

したがって，ユーザがやらねばならないコンカレントプログラミングの話も，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話も，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．現在の並列マシンでは，ユーザ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7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ICOTのアプローチでは言語もOSも，すべて新しいことに気づく．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．

並列ソフトウェア研究の進め方

問題を並列プログラム化しようというとき，応用問題の領域で日常の仕事をしている人たちは，「えっ，負荷分散？　聞いたことがない」「書きたくないなあ」という．一方，システムを作っている人たちは，「システム評価用にプログラムは書きたいけれど，アプリケーションには far away」と，つい言ってしまう．これでは実用レベルの応用問題の並列化はおぼつかないし，技術の蓄積もままならない．小さくても応用プログラムを書き，走らせては結果をシステムの改良に反映して，ちょうど雪だるまをころがすように技術を太らせてゆかねばならない．そのために図8のようなアプローチをとっている．並列化したい応用問題があるとき，応用側の人とシステム側の人が共同研究チームを作る．応用1，応用2，応

用3というように，問題ごとに共同チームを配置する．さらに，その中に含まれるシステム側の人は，個々の応用から生まれた技術の一般化を行なうための横断的なチームを形成する．ここでは，生まれてきたコンカレントプログラミングの技術，パラレルプロセッシングの技術を並列アルゴリズム集やプログラミングテクニック集にまとめる．また，標準の負荷分散方式をOSのレベルでサポートするなど，システムに対していろいろなフィードバックがかかるように努力している．

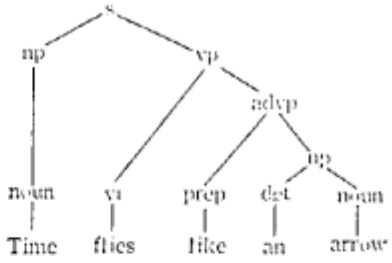
一方で並列処理の研究の立場からは，どの問題を題材に研究をしようかと迷うことがある．問題の選定に当たっては，「何をやりたいか」「どうすれば解けるか」がよくわかっている問題を選ぶことが重要である．負荷分散や通信の局所化など，並列処理自体の難しい課題があるので，知識処理などの解き方のよくわからない問題を持ってくると，わからないものだらけで手がつかなくなる．並列処理の研究には「早く解けるとありがたい問題」「やりたいことがよくわかっている問題」を選ぶことが重要である．並列処理で「賢く解きたい」という場合は，賢く解くための方法が定まってから並列化するのが順序であろう．解き方のよくわからないものをいきなり並列プログラム化することは，問題領域のプロが並列処理のプロと重なるときにだけ可能な，ぜいたくなアプローチである．

5. 並列プログラミングの具体的取組み

先に述べた研究の進め方を実践して，並列プログラム作りに励んできた．ここではまず，Multi-PSIの上に最初に作られた四つの実験的プログラムとその中で行なわれた研究を紹介する．それらは，それぞれ異なる種類のプログラム構造と異なる実行特性をもつようなものが選ばれた．また最近の応用プログラム開発についても簡単に触れたい．

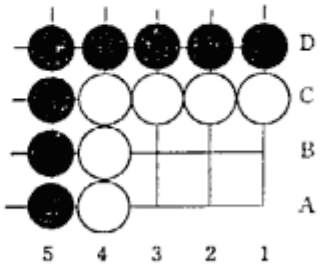
自然言語構文解析

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9）．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なう．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やすい．この種のプログラム構造をもつ問題は，分散メモリ型並列マシンにとって，最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．



－ボトムアップ全解探索
－通信の局所化優先の負荷分散

図9 自然言語構文解析



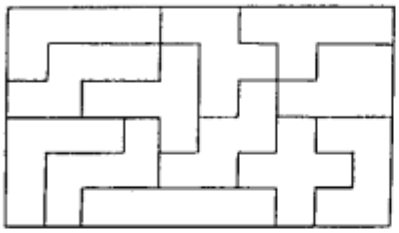
－ゲーム木探索(α-β枝刈り法)の並列化アルゴリズム

図10 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3～6倍で頭打ちとなった．ただ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．

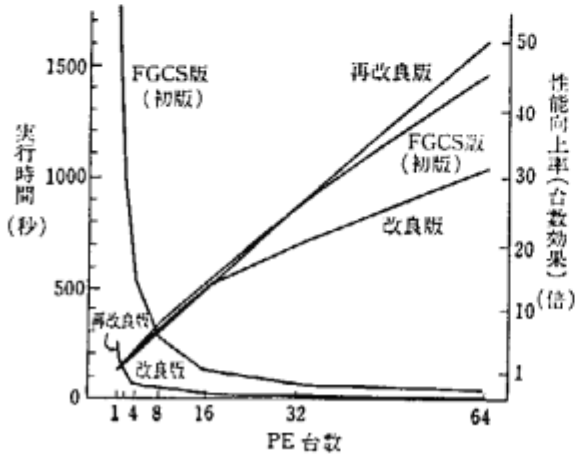
詰め碁

詰め碁は，囲碁対局の部分問題である（図10）．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よいアルゴリズムとして，アルファー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，枝が刈れずむだな計算ばかり増えるという困った事態が起こる．これはまさに，本誌90年10月号の7 **bit** sで，THEMSKY氏が書かれた記事中の「無駄な見込み計算：speculative computation」に当たる．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



－ OR並列型の全解探索
－ 多階層動的負荷分散

図 11　詰込みパズル（ペントミノ）

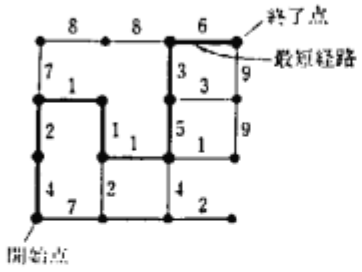


詰込みパズル(5×8マス, 10ピース)

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 12　詰込みパズルの測定結果



－ 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 13　最短経路問題

ルゴリズムの研究と負荷分散の実験を行なった．

KL 1 の実行優先度制御の機能を用い，枝の並列展開のときに，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，よ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，知識が不適当で見込みはずれを起こし，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．

詰込みパズル

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 11）．正方形 5 個分の面積をも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[7]．

10ピースの詰込みパズル（テトロミノ）を用いて性能測定した結果を図 12 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，64EP で45倍でありよかった．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，台数効果はかえって悪くなった．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，よく聞く話である．分析の結果，動的負荷分散のマスタ PE がボトルネック傾向であることがわかり，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64 PE で50倍が得られた．MIMD マシンにとって十分よい値である．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 が得られていることがわかった．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速いはずである．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 マシンに飛びつくことで

あろう．

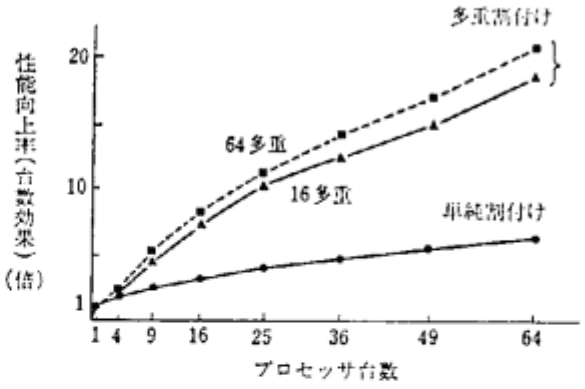
最短経路問題

最短経路問題は，図13に示すように，各辺に異なるコストをもつグラフ上で，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，他のすべ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，すべ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．KL 1 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．静的負荷分散を使用し，グラフをプロセッサ数ちょうどに分割してそのままマッピングする単純割付け，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けを実験した[7]．

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14に示す．単純割付けでは，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，単純割付け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けである．一つのプロセッサには，部分グラフがサイクリックに割り付けられるため，波面は繰り返し訪れて負荷の平坦化が図られる．64多重の割付けでは，単純割付けに比べ3倍高い性能を示した．このとき，64 PE を用いた実行時間は約4秒である．ただし，通信の局所性は単純割付けが最も高く，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わかりやすい例である．

開発経験

これらのプログラムは，KL 1 のソースプログラム行数で1,500から8,000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，KL 1 を使った感触は「使いものになる」ということである．オブジェクト指向機能を入れればもっと書きやす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，効率のよ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．KL 1 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ありがたい言語機能である．また PIMOS の開発で特に印象的だったのが，通信や同期にかか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．KL 1 が通信と同期に関してユーザ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もつに至った．



最短経路問題（40,000ノード）
・単純割付けではPE台数nに対しほぼ$\sqrt{n}$の性能向上
・多重割付けではさらに2～3倍高性能

図14 最短経路問題の測定結果

本格的並列応用プログラムの開発

四つの実験的プログラムの開発に続き，より本格的な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．

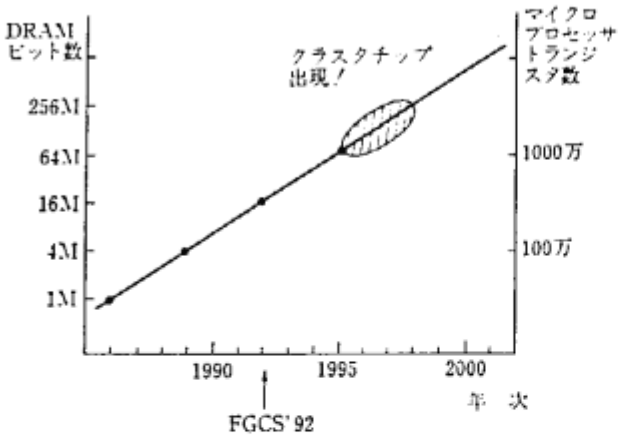


図 15 LSI の集積度変化とクラスタチップ出現時期

- 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 やタンパク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ともに，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および解釈の事例ベースをもち，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．解釈により異なる結論に到達し得る全過程を並列処理で求める．

（4） 囲碁対局：囲碁の対局を行なうプログラムは，探索空間の巨大さゆえに，チェスのような力づ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．

6. これから，そして少し先のこと

プロジェクト終了時の研究ステータス

第5世代コンピュータプロジェクトは，1992年に終了の予定である．その頃の並列処理の研究ステータスがどうなるかについて，私見を述べる．

ハードウェアは，PIM の数モデルが稼働し，こう作れば一通り満足のゆくものができるという例が示された段階であろう．ポイントは，KL1言語処理系との関連において，ハードウェアのどこを頑張るべきかが明らかとなることである．並列処理のエッセンスが詰まった KL1 言語を正しくサポートするハードウェアは，並列処理の本質を突くはずである．そのあとは，ソフトウェア研究開発用マシンの安定供給を行なうことが課題であり，さらに小型化と安く作ることも重要となってくる．

一方ソフトウェアでは，PIMOS や応用プログラムの開発を通して，分散メモリ型大規模並列マシン用ソフトウェアの解決すべき一連の課題が示された段階であろう．その中の一部についてだけ，解決方法が明らかになっている．トライアンドエラーをまだまだ続ける必要のある時期であり，研究の展開と技術の汎用化のために，あと10年くらいは必要な段階と考える．New Culture であるならば，そのくらい手をかけて育てるべきものではなかろうか．また研究の継続に元気でありつづけるために，難しくても，より魅力的な並列応用プログラムの研究開発に取り組むことが重要と考える．

クラスタチップとシステムの姿

1990年現在，16メガビット DRAM のサンプル出荷が始まろうとしており，マイクロプロセッサでは100万トランジスタを少し越える集積度のものが流通している．1486などがこのクラスに当たる．1チップに集積できるトランジスタの数は，2年で2倍を少し上回る勢いで増加中であり（図15），1995年を越える頃には，図5で少し顔を出した1チップマルチプロセッサ（クラスタチップ）も夢ではなくなる．クラスタチップとは，PIM/p の項で述べたような密結合のクラスタから，共有メモリだけを外に追い出して1チップ化したものである（図16）．すなわち，プロセッサとキャッシュの組が8個，多重化された共有バスに接続され，さらにクラスタ間接続のためのネットワーク・インタフェースと，共有メモリ側のキャッシュまでを1チップに集積したものである．共有バスがチップ上にあるため，クラスタ内の PE 間通信速度が飛躍的に改善される．

6年後の集積度が10倍程度上がると思うと，今のキャッシュつきのマイクロプロセッサが，チップ上に8個載ることは容易に想像できる．ただし，共有メモリ側のキャッシュとネットワーク・インタフェースが面積を食うので，プロセッサ部分は RISC 技術を使って多少スリムに作る必要があろう．シリコンデバイスの進歩によるクロック周波数の向上は，年率15%といわれている．1995年頃は67MHz が標準値と考え，マシン命令1個当りの平均クロック数（CPI）を1.4とすると，プロセッサ1個の性能が50 MIPS となる．チップ全体では，ピーク

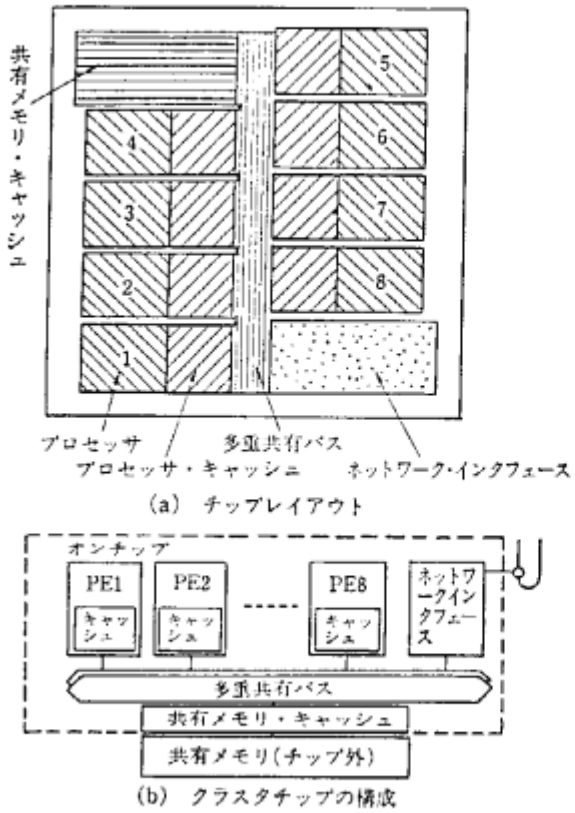


図 16 クラスタチップ

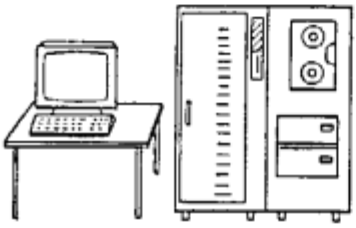
性能で 400 MIPS が得られる．これは米国の半導体メーカの試算よりも緩い現実的な値である．

このチップを使うと，図 17 のようないろいろな高性能システムが組める．チップを 1 個使って，400 MIPS のデスクトップ・ワークステーション，ボード 1 枚に 2 クラスタ実装して合計16クラスタ 128 PE を内蔵するデスクサイド・ワークステーション (6.4 GIPS)，そして極めつけは大型筐体に128 クラスタ 1024 PE を実装する50GIPS の超高性能ワークステーションが，I/O を含めて筐体 2 本くらいで実現できてしまう．このシステムは，できてすぐはネットワーク中の計算サーバやデータベース・サーバ／知識ベース・サーバとして使われ，しだいにスーパーワークステーションとしての利用が増えてゆくことであろう．ICOT のプロジェクトが目指したマシンが，1990年代の後半には，ほぼ確実にこの大きさで実現できてしまうのである．

このマシンを思い通りに動かす技術，使いこなす技術の開発が急がれる．

（a） デスクトップ・マルチプロセッサ WS
8 PE, 400 $MIPS_{max}$

（b） デスクサイド WS
16クラスタ 128 PE, 6.4 $GIPS_{max}$



（c） 超高性能 WS
128クラスタ 1021 PE, 50 $GIPS_{max}$

図 17 クラスタチップを使用したシステムの構成

7. 終わりに

ロジックプログラミングからスタートした ICOT の並列処理が，実は将来の大規模汎用並列処理に至るメインストリートの真ん中を堂々前進しているのだ，という感触を感じ取っていただけただろうか．分散メモリ型大規模並列マシンは，今やハードウェア技術において実現を約束されたようなものである．重要課題は，それを使いこなすためのソフトウェア技術であり，並列ソフトウェアの研究課題として示した諸々の技術群である．

並列処理技術の現状は，過去の汎用マシンの歴史でたとえるならば，1950年代の終りから1960年代初頭の技術レベルといえる．言語も未熟であり，OS も整備されておらず，利用者が問題を解くためのプログラムだけでなく，メモリや入出力の管理やスケジューリングまでこと細かく心配しなければならない．苦しさ余って昔びに通じる職人芸の世界である．たくさんの職人さんと少しの発明家の働きによって，この時期をたっぷり時間をかけて経過して，はじめて普通の人がほどほどに文句をいいながら使える，ほんとうの大規模並列マシンの時代に入ることができるのであろう．読者諸氏の中からも，この職人芸の苦しみと喜びの交錯する夜明け前の世界に踏み込むことをよしとされる殊勝な方々の多く現われんことを切望する次第である．

筆の勢いでこのような表現となったが，実は並列処理はものすごくおもしろい．とりわけ，プログラムをいじって負荷分散の実験にハックし始めると，中毒になるほどおもしろいのである．KL 1 では楽に書けてしまうことや，並列ソフトウェアの世界は何を試しても新鮮であることなどが原因だろうか．期限つきの仕事でない限りぜひとも味わってみることをお勧めする．

ところで，PIMOS や初期の応用プログラム開発にも使用された KL 1 の擬似並列処理系（PDSS）が，UNIX マシン上で利用可能である．ソースコードも，ICOT のテクニカルメモとして公開されている．夢多き並列処理の世界に通じる入り口の一つとして活用されることを歓迎する．お問合せは pdss@icot.or.jp まで．

参考文献

1) K. Taki : The FGCS Computing Architecture. INFORMATION PROCESSING 89, G. X. Ritter(ed.) (IFIP Congress '89), North-Holland, pp. 627–632, ICOT TR-460.
2) 古川康一他：並列論理型言語 GHC とその応用，共立出版，(1987).
3) 宮崎敏彦：PDSS 使用手引き，TM-437，ICOT(1987).
4) 佐藤裕幸 他：PIMOS の資源管理方式，情処論 Vol. 30, No. 12, pp. 1646–1655 (1989). ICOT TM-815.
5) K. Nakajima, *et al.*: Evaluation of Inter-processor Communication in the KL 1 Implementation on the Multi-PSI, TR-531, ICOT(1990).
6) 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, pp. 1584–1592(1989), ICOT TR-453.
7) 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.
8) 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.

* ICOT TR, TM は購入することができます．

（たき かずお （財）新世代コンピュータ技術開発機構）