TR-596

ICOTにおける並列処理の研究概要

瀧　和男

September, 1990

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

ICOTにおける並列処理の研究概要

瀧　和男
Kazuo Taki
財団法人　新世代コンピュータ技術開発機構

1．はじめに

ICOTにおける並列処理研究の概要についてお話しする。研究進捗については何を作り，それがどこまで進んでいるのか，また並列処理の研究については，よそで行われている研究との対比においてその特長や種類を，また並列応用の実験プログラムを現在はしらせているが，その結果と課題について，そして並列処理マシンが将来どのように展開してゆくのかなどを説明することにする。

2. 研究進捗

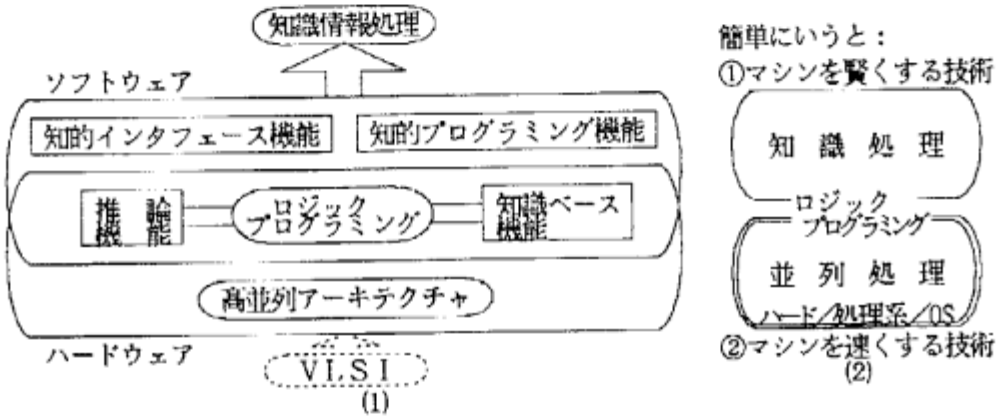


図1　第五世代コンピュータ研究開発プロジェクトの枠組

図1は第五世代コンピュータの研究の枠組みを図示したものである。第五世代コンピュータは高度な「知識情報処理」システムの開発を目指しているが，図1（1）だけでは難しくてよくわからない。わかりやすくいうと，知識情報処理を実現するためにはマシンを賢くするための技術としての知識処理と，マシンを速くするための技術としての並列処理に大別できる。そしてこの両者を融合させるもの，あるいはこの両者を研究してゆく上でのベースとしているものが，ロジックプログランミングである。この「並列処理」について今日はお話しする。知識処理はかねがね長期の基礎研究が必要といわれてきた難しい分野であるが，一方の並列処理も予想以上に難しい。決して簡単なものではないとの実感を覚える。

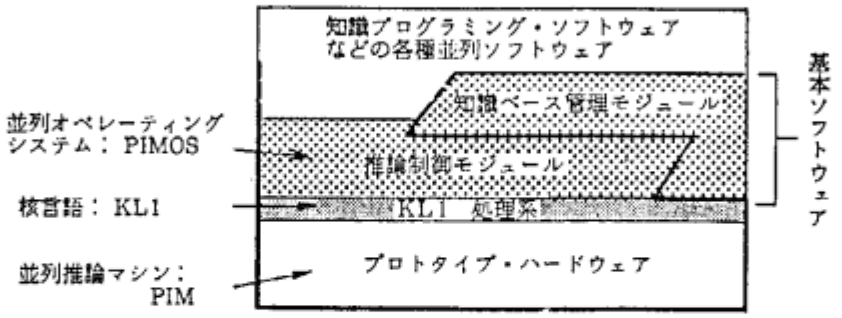


図2　後期目標の５Ｇプロトタイプ・システムの構成

図２は第五世代コンピュータの粗っぽい構成図であるが，最下段にハードウェアのレイアがある。これは並列推論マシンPIM そのものである。その上に核言語KLI があり，これがハードとソフトのインタフェースをつかさどっている。その上に並列のオペレーティングシステムPIMOS，これは推論制御モジュールという公式名称がついている。それから知識ベース管理モジュールがある。この太線で囲んだ部分を中心に説明を加えてゆきたい。

ICOTではハードウェアからOS，その上のソフトまで丸ごと作ってゆこうというアプローチをとっている。従来は，並列のハードウェアの研究とソフトウェアの研究はもちつもたれつというか，お互いに足を引張りあってあまり進んではいなかった。並列のハードがないと並列ソフトの研究ができないとソフト屋がいい，ソフトの要求がないのでどんな並列ハードを作っていいかわからないとハード屋がいって研究が進まない，ということがあっては前進しないので，先ず小さなハードを作り，そしてその上に小さなソフトを書き，そこからソフトの要求をハード側にフィードバックして次のマシンを作るといった段階的拡大と相互フィードバックを行ってゆくことがICOTでの研究のベースとなっている。

このような考え方で前期（1983－85年）ではまだ並列までゆかず，先ず自分たちの研究ツールを作る意味で逐次型のプロログマシンというか，論理型言語マシンPSI を作った。PSI を作りバージョンアップし，小型化し，ESP という言語 — プロログの拡張版 — を作り，その上でSIMPOSというオペレーティングシステムをロジックプログラミングで書いた。この経験の中で，ロジックベースのマシンというものをどのように作れば良いか，論理型言語でOSみたいなものを書くときにどのようなむずかしさがあるのか，どのように書けば良いのかという経験をしたが，これはあくまでも逐次型であった。

中期（1986年）に入ってハードとソフトの足の引張り合いをやめ，相互協調のブート・ストラップのフェーズが始まった。ベースとなる並列の論理型言語 GHCの設計が最初に行われた。これに基いてMulti-PSIの第１版　６台のプロセッサをつないだものの上にGHCのサブセ

ットFGHCという言語の分散処理系を作った。この上で小さなプログラムを書いた。このスピードはプロセッサ当り1KLIPS（1Kロジカルインファレンス・パー・セカンド）という非常に遅いものであった。これは処理系がESP で書かれていたためだが，スピードは遅くとも一応分散処理系になっていた。冷蔵庫位の大きさのPSIを６台つなぎ，たち上げには全部のPSIを人間が操作してまわるという面倒なシステムではあったが，分散処理系がのっていて，小さなプログラムを書き，いくらかの経験を積むことができた。これに基いて次は1988年にMulti-PSI第２版のシステムを作った。このプロセッサは，小型化PSI のプロセッサと同じもので，これを64台接続した。１プロセッサ当りのKL1 の処理スピードは130KLIPS で一挙に130倍上昇した。すべてマイクロプログラムで処理系が書かれており，並列処理用のまともなプログラムが書けるようになった。その上にPIMOS と呼んでいる並列オペレーティングシステムの第１版ともいうべきOSをのせた。このPIMOSは今後，PIMなどに継続して使用してゆこうと考えている。そして並列応用の実験プログラムをこれで作っていったのである。これはFGCS'88 のデモンストレーションでも使用した。早く1MLIPSを越えるシステムを作りたいと考えていたが，ようやくMulti-PSIのところへきて，M-LIPSをこえる性能が出たわけである。

これが現在のステータスであるが，その後（1990年）並列推論マシンPIM のいくつかのモデルの開発をやっているが，その最初のモデルはPIM-P と呼び，プロセッサの性能も４倍くらい上昇し，プロセッサ台数も倍増，1990年度の末にはハードウェアが出来，1991年 からはその上にソフトを試作してゆくフェーズに入ってゆくことになる。現在，富士通，日立，三菱，東芝，沖と，それぞれ研究のポイントを少しずつずらしたPIM のモデルを作っているが，その拡張版というかファイナルPIM というのが第五世代コンピュータのハードとなるものである。それぞれ独立で動くPIMがネットワークで統合される。これは1,000台のプロセッサで，全体の計算能力は100 M-LIPS以上のものとなる。

これらを研究進捗過程の面から整理してみると，最初は冷蔵庫のような大きさのものが６台結合されて３KLIPS 程度の性能しかでなかったものが，Multi-PSI/V2になって八つの箱に64のプロセッサが詰めこまれ，性能も２～５MLIPS も出るようになった。さらにPIM/Pになるとと Multi-PSI/V2の半分の数の箱に128 プロセッサが入り，つまり実装密度は４倍になったことになるか性能は10～20MLIPS になる。そして最終のマシンではさらに大規模化してゆくことになる。

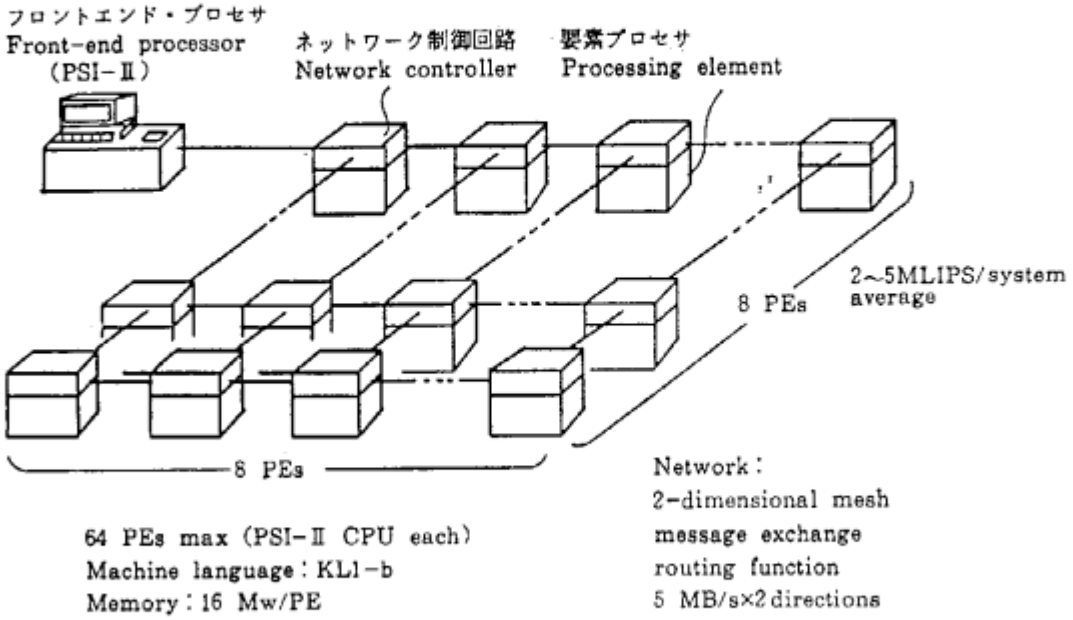


図3　Multi-PSI/V2

図3はMulti-PSI/V2で，すでにFGCS'88 等で実物を見た人も多いと思うが，図中の小さな箱がプロセシングエレメントである。それが8×8の二次元格子状に接続されている。一つの箱の下の方がプロセッサ，上の薄い箱はネットワークコントローラである。プロセッサ間はメッセージ通信で対話しているが，プログラマーから見れば単にKL1 のユニフィケーションが書かれているだけである。これが勝手にメッセージ通信におき替って，プロセッサ間をとび廻っているという仕組みになっている。したがって共有アドレス空間を想定した言語処理系が疎結合のマシン上にマップされて作られていると考えてもよい。ユーザは仕事をプロセッサへ割付ける際に，プロセッサ番号まで指定する。ユーザが指定しないで勝手に割付けられるモードというものも将来できなくはないが，下のレベルではプロセッサ番号を指定することになる。逆にそれを書かないと今のところ並列に動かない。GHC の言語で書かれているレベルではプロセッサはでてこないが，それをはしらせる場合，プロセッサ間に仕事を分配させたいという問題が生じたときに，自分の書いた問題をどのように分配したらいいかを考え，アットマークプロセッサ番号何番と書くか，計算された結果の番号がそこにアサインされるようにして記述してゆくことになる。

以上がMulti-PSI/V2であった。

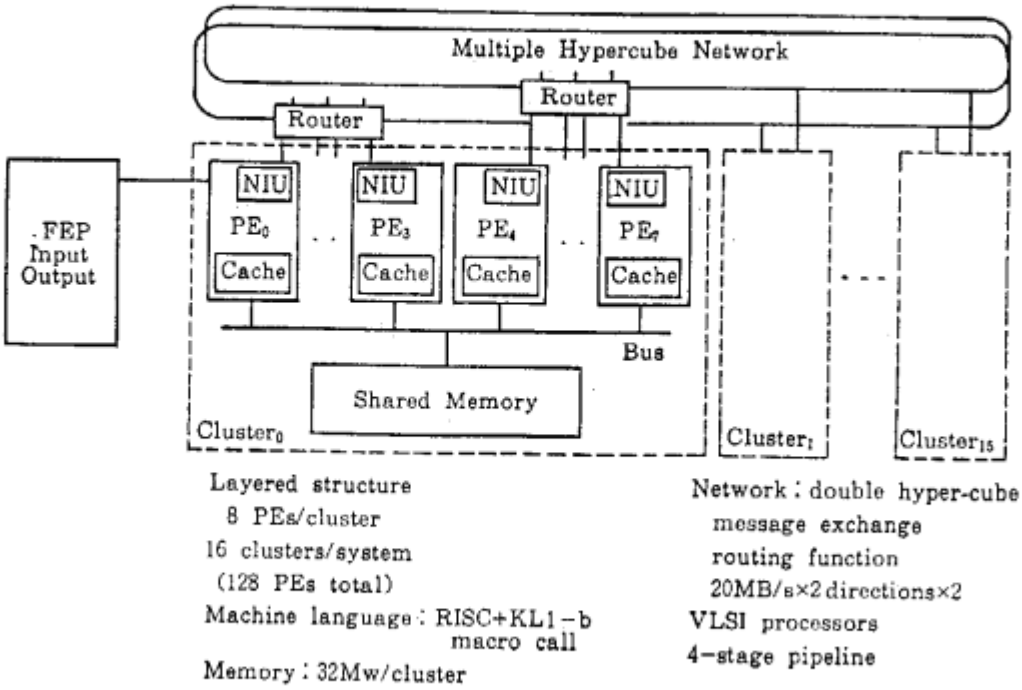


図4　PIM/p

次に，PIM/p（次のマシン）の話をする。PIM/pはマルチPSI の一つのプロセッサのかわりに八つのプロセッサが密結合され，共有メモリを持っている。これをクラスタと呼ぶ。クラスタがネットワークで結合された構造のものがPIM/pのシステムとなる。図4ではPEnと書かれた小さな箱がプロセッサで一つのクラスタに8個収納されている。各プロセッサには並列キャッシュ(Cathe) が入っており，共有バスを介して共有メモリにアクセスする。このシステムはハイパーキューブ(Hypercube) のネットワークで結合されている。このようなマルチクラスタの構造は将来の大規模なMIMD型と呼ばれるマルチプロセッサシステムの非常に素直な結合方式というか，実現方式の代表になるものと思われる。大規模なマシンシステムを作ってゆくには，どのプロセッサからも別のプロセッサのメモリを同じ距離でアクセスするのは不可能で，必ず遠い所，近い所がでてくる。これを結合させるにはネットワーク結合というものが必要となる。大規模になればなるほど必須となってくる。

もう一つの問題としては，いくつかのプロセッサに共有メモリを持たせることによりメモリの利用効率が上がる。Multi-PSI のようにプロセッサとメモリの対がバラバラにしてあるとメモリを効率よく使うプロセッサとメモリを使っていないプロセッサができて，メモリの利用効率が上がりにくい。いくつかのプロセッサがメモリを共有することにより，メモリ利用効率も向上してくることになる。従って大規模なMulti プロセッサでは良い結合方式になると思う。ネットワークで結合しているところはこのMulti-PIM でも共通であり，並列処理に関する共通の研究課題を有することになる。特に大切なのは負荷の均等分散である。沢山

のプロセッサに均等に仕事をさせる，いわゆる負荷の分散である。これと正反対な問題が通信の局所化である。仕事を分散させること，分散した各々の部分の間では通信の必要が生じるが，一番通話を必要とする部分を遠い場所へもってゆくとネットワークが混んでくる。したがって対話の多い部分は同じプロセッサの中へ入れておくか，近くへおく必要があり，これを通信の局所化と呼んでいる。これは先ほどの分散化とは逆のもので，分散しないと仕事が均等化せずプロセッサが遊んでしまって性能が発揮できないし，一方近づけておかないとネットワークが混んでしまってオーバーヘッドが増えるという相反する要求を満してやらないと性能がでないという難しさがある。これはかなりソフトがかった研究になる。

(1) 核言語KL1

Multi-PSIやPIMの上に実装している言語はKL1である。ハードとソフトのインタフェースをつかさどっている言語であるが，ベースはGHC と呼ばれる言語である。これはコミッティド・チョイス型のロジックプログラミング言語ともよばれ，論理型言語の一種である。KL1 は論理型言語といわれている割には論理型言語らしさが減っており，汎用の並列プログラミング言語に近いものになっている。並列処理が書ける言語は種々発明されているが，KL1 はそのどれと比べても記述能力があり，いろいろなタイプの並列処理が思うように書けるものである。また，論理型言語の枠を守っている点ではプログラム変換などにとっては有難い性質をもっている。データフロー並列（暗黙の並列性と同期機構）という特徴もある。これは通信とか同期というものを従来型の並列言語を使って書くと，それを色々と書かなくてはならない。すなわちこのモジュールとこのモジュールがここで同期をとるとか，このように通信させるとかを見えるように書いてゆく必要がある。並列に実行する単位を変えるとか，これまで一つの塊と思っていたものをいくつかに分断して，またそこに通信や同期が出るとなるとまたそれを書き直さねばならないという不都合があったが，KL1 の場合，通信とか同期がすべて普通のユニフィケーションのところへ出てくるため実行の塊の大きさを変えてプロセッサに割付けようと思えばアットマーク・プロセッサ番号のところの値を変えるだけで変更でき，プログラム全体をそれほどいじる必要がないという優れた点がある。また通信と同期がユニフィケーションの記述で済んでしまうので，通信とか同期のところに出てくるバグも極めて少ないものになる。これも大きな利点の一つである。

GHC に機能を拡張した点の一つ目は，プラグマと呼ばれているものである。これはプログラムの構造あるいは意味を最初に記述した部分とは別に，外したり，つけたりすることができるという記法となっており，プロセッサの指定であるとか実行の優先度の指定などを書くことができる。パフォーマンスをチューニングするとか，仕事の分配をチューニングするなど1回作ったプログラムを変えずに色々な実験ができるという点で優れたメカニズムといえるものである。

さらにもとのGHC に入っていなかったメタレベルの制御機能に「荘園」という変った名

前の機能がある。これは，OSの行う資源管理，タスク管理を助けるものである。これが入ってOS記述が随分と楽になった。従来のセンスでいうとOSの機能の一部がこの言語機能の「荘園」の中に入ってしまっているという面があるが，これについては説明を省略する。通信，同期などに関してこのような先進的な言語は，他に関数型言語などでトライしている例もあるが，KL1 では論理型言語らしさがある程度減少しているので，関数型言語ともかなりの共通点をもっている言語といえる。

(2) PIMOSの設計

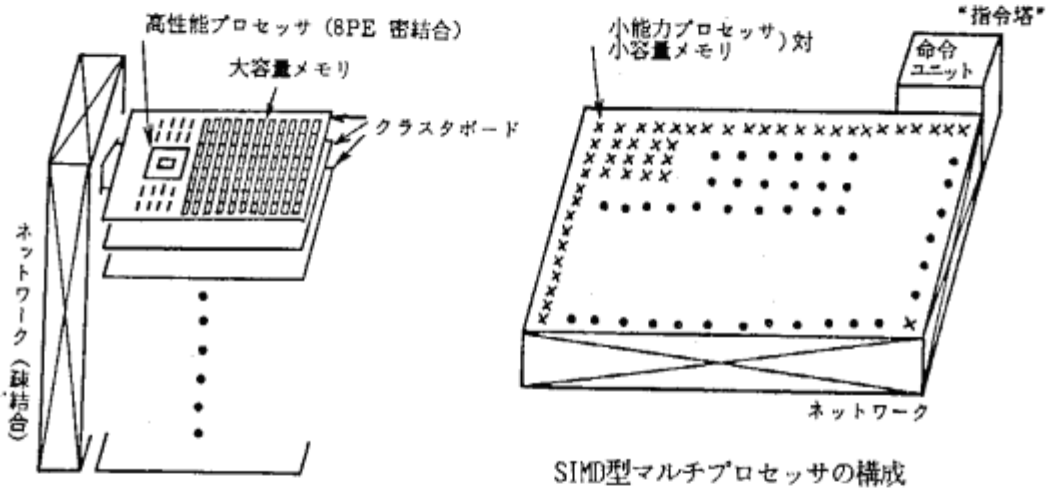
オペレーティングシステムPIMOS は実験マシンのための実用OSといえる。まだ並列マシンのために必要なOS機能という点ではすべてをリストアップしつくしたわけではないが，Multi-PSI，PIMを色々な並列ソフトの実験に使うため実用に耐える機能を網羅している。また沢山のプロセッサを制御するOS－並列で動くOS－であるが，全体で単一のOSでもある。これは分散OSとの違いである。分散OSとはW/S がネットワークでつながれていて，それぞれのW/S 上に独立のOSがあり，それが何らかのインタラクションをとりながら一つの世界を構成しているように見せるOSであるが，PIMOS は全体で一つのOSとなっている。PIMOSはすべてKL1で記述されており疎結合の並列計算機を想定して書いてあるが，これはMulti-PSI，PIM，その後にくる将来の大規模な汎用並列計算機の上にのることを期待している。OSが集中化されているとOSのボトルネックで並列処理の性能が出ないということがある。従って極力OSが集中化しないような配慮が必要である。

3．どんな並列処理を目指すのか　－大規模疎結合並列マシン上の汎用並列処理－

どのような並列処理を目指しているかというと大規模疎結合MIMDマシン上の汎用並列処理ということになるが，これは将来の汎用並列マシンのメジャーな姿であると思う。ロジックプログラミングあるいは知識処理だけを目指しているものではなく，かなり汎用の並列処理を考えている。世の中には全くタイプの異なる並列マシンは数多くある。その中のMIMDと呼ばれているタイプのマシンを指している。1988年にFGCS'88 のデモを行った時「ICOTが作ったのは64台結合のマルチプロセッサでしかないが，世の中には64,000台もつながっているものもある。」という記事を載せた新聞があるが，64,000台もつないだのはSIMDと呼ばれているマシンであった。SIMDマシンとは非常に小さな能力の小さなプロセッサに小容量のメモリがくっついているようなものが数多く高速のネットワークでつながっているシステムのことである。これは小能力プロセッサと小容量メモリの対が均質に分布しているものと考えてもらいたい。実際のコネクションマシンではネットワークはハイパーキューブになっている。このようなSIMD型のマシンは，均質なデータに対して均質且つ定型の処理をくり返し実施してゆく用途に向いている。例えばマトリックス計算とか画像処理がそれに当たる。イメージ的には，右隣のプロセッサからデータをもらってきて自分の持っているデータの½の値を加えて係数をかけ自分のところへしまってしまうと同時に左隣のプロセッサへ送るというよう

な仕事をあっちでもこっちでもやっている。これを繰返し繰返しやっていると答えが求まってくるというようなイメージのものがまさにSIMD機である。これには数多くのプロセッサはあるものの命令を出す指令塔は1ケ所しかなく，ここから全プロセッサに同期して，まずこれを実行せよ，次はこれをと，指令している。全体が常に足並みを揃えているといったシステムである。

一方MIMDというのは，一つ一つのプロセッサはもっと賢い。大能力，高性能のプロセッサと大容量のメモリがネットワークでつながっているというシステムである。図2.3.5の左図はもう少し先のイメージを示したものであるが，8台の高性能プロセッサが密結合され大容量共有メモリとともに一枚のボードに実装されてクラスタを構成しており，これがネットワークでつながっているというイメージである。これが将来のPIM であり，将来の大規模な並列マシンの姿であろう。



規模MIMD型マルチプロセッサの構成
→ 将来の PIM
→ 将来の汎用並列マシン
1,000台規模の高性能プロセッサ

SIMD型マルチプロセッサの構成
→ 例 コネクションマシン
6万4千台の小能力プロセッサ

図5　大規模MIMD型マルチプロセッサ ec. SIMD

これは高性能のプロセッサが大容量のメモリをかかえているといった点でSIMDと異なり，もっと不均質のデータ，不定型の処理が入るようなものに適用可能なシステムになっている。例えば知識処理などではプロセッサは沢山あるが，こっちのプロセッサのもっている知識とあっちのプロセッサの持つ知識は異なっていたりする。また動的に知識処理をやっている場合には入ってくるデータによりこっちの知識を利用する，あっちの知識を利用する，または一つの入ってきた情報に対して全く違った知識を動的に適用してゆくようなことも行う。そうすればこのあたりに入った知識と別のところへ入った知識が全く違ったり，入ってくるデータによってあるところは忙しく，ある所は仕事がないといった負荷の発生がわからないよ

うな場合も出てくる。このような場合，仕事の粒の大きさが場所によって大きかったり，小さかったり，その差が大きくなる。小さい仕事の粒と大きい仕事の粒では仕事の終る時間が全く違ってくる。すぐ仕事の終ったプロセッサが，いつまでも仕事の終らないプロセッサを待っているようでは性能が出ないわけである。従ってその間に違った仕事ができるように必然的にプロセッサにつくメモリも大容量でなければならないということで，MIMDマシンは大容量メモリと高性能プロセッサがその特長となっている。ICOTが目指しているのはこのMIMDマシンであり，不均質データ，不定型の処理，代表的な例として知識処理を対象としており，さらにはSIMDでは効率よく扱えないような処理一般に適用できるシステムを考えている訳である。換言すればより広い適用範囲に適用できるシステムであり，より難しい技術でもあるが，このような汎用並列処理システムを開発しようとしているのである。

SIMD : Single Instruction Stream Multiple Data Stream
インストラクションストリームとは命令列であるが，命令列か指令塔から全部のプロセッサに共通に供給されてゆくので命令はどこでも同一だということで Single Instruction Streamという。

MIMD : Multiple Instruction Stream Multiple Data Stream
それぞれのプロセッサは全く独立の命令列をそれぞれ実行している。それぞれ賢いプロセッサが動いているということになる。

次にマシンをどう速くしたいかについて説明する。どのように速くしたいのかということでしばしば誤解がみられるのであるが，ICOTでやっているMIMDマシンはN台(最大でも1,000台というところをめざしているが) のプロセッサを用いて一番速くてもN倍になれば良い。場合によっては，それはコンスタント分のN倍であったり，$\sqrt{N}$倍しか速くならないということも問題によっては当り前と考えている。一方これと異なる考え方として無限に近い多数のプロセッサをもってきて計算時間のオーダ，問題のサイズに対する計算時間のオーダを下げるというアプローチがある。N^2だとかNlogNだとかアルゴリズムでいっているものである。コネクションマシンのようなアプローチでは64,000台を無限に近いと思って扱う場合がある。それより小さいデータをもってきたときに並列処理によって計算時間のオーダが下がるといったことを議論している場合もあるようだが，ICOTではそのようなーアプローチはとっていない。プロセッサ台数Nに対して最大N倍の高速化しか期待しない。これは問題の大きさに対してはコンスタント倍しか速くならないという人もいるが，MIMDマシンのアプローチはこれしかないのである。（N倍，N/K倍，$\sqrt{N}$ 倍だけ速くなる。）

世の中には並列にしても速くならないではないかという人もいるが，並列処理によって計算時間のオーダを下げるというアルゴリズムは存在しない。だから並列処理では，スピードは速くならないという議論をしていることがよくある。これらはまどわされてはいけない点であって，これは並列計算量の理論からの主張であるが，この場合は「単に，良い並列アルリズムがないといっているのではないか」と問いかけてやるのがいい。

最後にN倍のプロセッサをもってきてN倍よりも充分速くなるという話があるが，スーパーリニアスピードアップという言葉で呼ばれているものである。N台もってきてN倍より速くなるということはあり得る。これは並列に実行可能であるがスケジューリングにより計算量が変化するような問題を1台で実行するときに，そのスケジュールが悪かったり，スケジューリングオーバヘッドが出たりしていたものが，並列処理によりスケジューリングが改善されたりオーバーヘッドが減ってしまうためにN倍よりも性能が出るというケースである。しかしN倍よりずっと性能が出るということがもしあったならば，それは逐次実行でもスケジューリングの改善により実現可能な場合のみである。MIMDのマルチプロセッサを使いこなすには仕事をいかに均等にバラまくか，それと反対に通信をいかに局所化するかという難しさはあるが，これがまさに負荷分散とかスケジューリングの技術になるわけであり，これとプログラミング技術が表裏一体のものであって，これらの技術が確立してはじめてMIMDマシンを使いこなすことが可能となる。同時に応用の開拓，それから安く作る技術なども重要な要素となるのである。

4．並列ソフトウェアの課題

— 並列処理は New Culture —

先ず意識のずれの問題について説明する。ハードからOSまでを含めたものを並列処理のシステムと考えよう。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気持ちよくマシンがそれを処理してくれる。それにより性能が向上するという期待があったし，現在でも期待している（図6(1)）。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。この数年並列処理の研究をやってきて応用問題と並列処理システムの間に並列ソフトウェアの諸技術と呼ばれる層，すなわち並列アルゴリズム，プログラミングパラダイム，負荷分散，通信の局所化など結構厚い中間レイヤがあって，これが殆ど未解決になっているといったことがわかってきた。

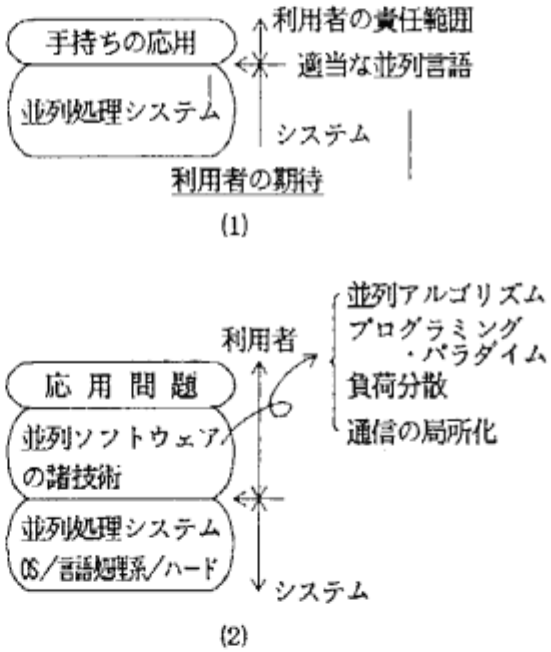


図6

並列ソフトウェアの諸技術には上の方には並列アルゴリズムというのがある。従来マシンで使われている有名なアルゴリズムは，順に実行することを利用し性能を向上させているのが殆どである。アルゴリズムがそうであるからこれを並列の言語で書き下して無理やり並列マシンにマッピングしても，沢山あるプロセッサが順に動いてしまうだけである。これはアルゴリズムが逐次性をもっているからに他ならない。並列マシンを効率よく動かすためには並列向きのアルゴリズム，逐次性の少ないアルゴリズムに作り直さなければならないという局面にしばしばつき当たる。手持の応用言語だけ変えればすむという話ではなくなる。どうすれば手持の問題が解けるか1回はプログラムになっているから分析は出来ている筈である。その中のアルゴリズムというところをにらんで逐次性を減じたアルゴリズムに書きかえてやらなければならない。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かえると，計算量のオーダが増えてしまうようなミスを犯すことがある。これは問題のサイズに対してNlogNで計算できていたものがN^2もかかるアルゴリズムに変ったりする。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ってしまう。コンスタントくらいなら増えてもいいが計算量のオーダをキープするような並列向きアルゴリズムに書き直すことが必要である。

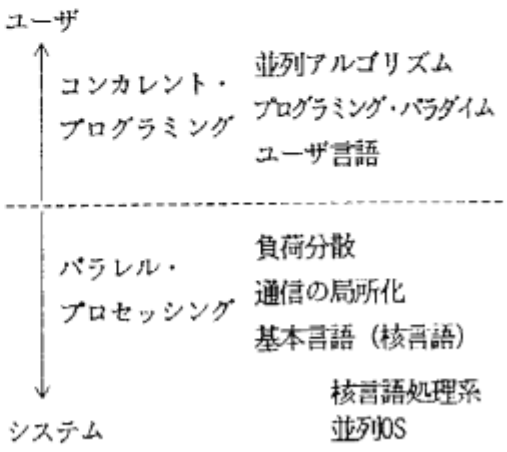


図7

次にプログラミングパラダイムについてであるが，あまり聞きなれない言葉かも知れないが，低いレベルではプログラミング，テクニック，もう少し上のレベルではプログラムの設計技法とかプログラムの構造をきめてゆくときの技法などが含まれるが，これらが蓄積されていない，つまりお手本がないので並列プログラムにおとそうと思っても手が出ない。従ってこれもお手本を作ってゆく必要がある。プログラミングの中でこのあたりの仕事は最低限ユーザが行わねばならない仕事であるが，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。

ここに含まれている並列性とは論理的並列性で，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いわれていない。アルゴリズムの段階で10,000の並列性をもっていても，それを10プロセッサのマシンにマッピングしても良いし，100 プロセッサのマシンにマッピングしてもよい。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面倒をみれば有難いわけであるが，内容としては負荷をいかに均等に分散させるか，いかに通信の局所性を保つか，それを書く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，などの技術課題が存在するわけであるが，これらの技術

はまだ確立されていない。従ってユーザがやらなければならないコンカレントプログラムの話もシステム側にやって欲し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とはそれ程距離が離れてはおらず，システム作りに手をそめている人が応用プログラムを書いているが，両方ひっくるめて研究をしている段階である。このようにアルゴリズムとかプログラミングテクニックから負荷の分散，通信の局所化などなにかと新しい。並列処理はNew Cultureであるといってもよいのではなかろうか。

実際に逐次型マシンの方で利用されていたコンピューティングの技法が使えないことの方が多い。今まさに研究開発の種が撒かれたという時期である。研究を進めてゆく上で先ず道具だてを整理し，－これは言語処理系，並列マシンなど ― その上で実験応用プログラムを書き，またマシンにフィードバックするという回転をやってゆかねばならないが。そのとき，問題の選定に当っては「何をやりたいか」ということが判っているものを選ぶことが重要である。負荷の分散，通信の局所化など並列処理自体の課題として難しい場合が多いので「どうすれば解けるか」よく分かっていない問題では困る。知識処理などで解き方の判らないものをもってくると判らないものだらけになって全く手がつかない。並列処理の研究には「早く解きたい問題」「やりたいことがよく判っている問題」を選ぶのがよいのではないかと思う。「賢く解きたい」というのは賢く解くための方法がわかってから並列化するのが順序ではなかろうか。

並列ソフトウェア研究の進めかた

応用問題の領域で研究をしている人は，「負荷分散なんか聞いたことがない」「書きたくない」という。一方システムを作っている人は応用問題を知らないのである。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。小さくても応用問題をかいて雪ダルマを回転させてゆくように技術を太らせてゆかねばならない。今やろうとしているのは，応用側の人とシステム側の人が協力して，応用問題対応の研究チームを作ってゆくというアプローチである。応用１，応用２，応用３という研究チームは応用の人とシステムの人との組合せになっており，さらにシステム側の人は横断的な一般化チームを作り，それぞれの応用の中で工夫し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だとか，プログラミングテクニック集にまとめ上げたり，標準の負荷分散方式をOSのレベルでサポートするとか，色々なフィードバックをソフトだけでなくアーキテクチャやコンパイラーなどにもフィードバックするように努力を始めている。

このような基本的な考え方にもとづいて，すでに稼働しているMulti-PSI の上でいくつかの並列応用実験プログラムを書いている。

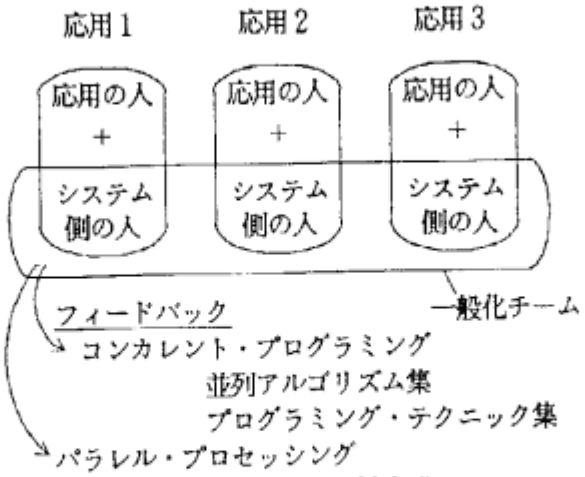


図8　標準負荷分散方式

5．具体的取り組み　— 四つの並列応用実験プログラム —

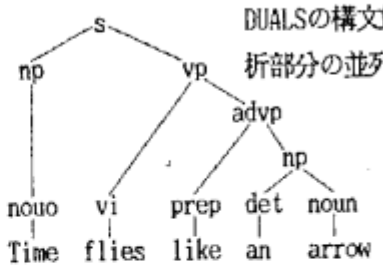
問題を選定するに当っては，なるべく異なるタイプの問題をもってきて，それをプログラムした時も，異なるタイプのプログラム構造だとか動的特性を示すようなものを選定した。それぞれのプログラミングの中で並列アルゴリズムの工夫だとか，負荷分散方式の実験をやってきて，これらをいづれは一般化してゆきたいと考えている。

(1)　自然言語構文解析

一つ目は自然言語構文解析プログラムのPAX である。DUALS という談話理解システムを見た人もいると思うが，そこに使われている自然言語の構文解析の部分を並列化したものである。この問題はマルチPSI のようなマシン上の並列処理にとってあまり有難いものではなかった。非常に沢山のプロセスが共同して動くがプロセス間でグローバルな通信が出て，プロセッサ間の通信のコストの高いMulti-PSI のようなマシンにマッピングすると，通信のオーバヘッドが出て性能が出にくいというような特性をもっている。ここでは通信を局所化し，オーバヘッドを減らすような負荷分散方式を試してみた。問題の種類としては，これはボトムアップの全解探索と呼ばれている問題である。

問　題

・与えられた英文の構文解析結果を全て求める



DUALSの構文解析部分の並列

— ボトムアップ全解探索

— 通信の局所化優先の負荷分散

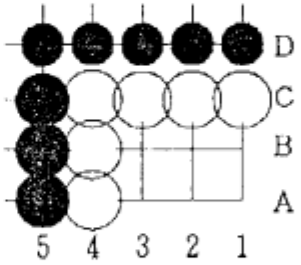
図9　自然言語構文解析

(2) 詰 碁

次は詰碁であるか囲碁の部分問題である。AIの問題での分類でいうとゲーム木の探索（$\alpha-\beta$枝刈り法）という問題である。探索の空間を狭める効率のよい枝刈り法が昔から使用されているが，これは非常に逐時性がある。順にやらないと枝刈り効率が上らない。並列でやると枝が刈れないという事態がおこる。並列性を出しながら枝刈り効率も低下させないためのアルゴリズムの実験をやり，また負荷分散の実験も行った。

問 題

・与えられた詰碁の問題で，白黒双方が最善を尽くした時の結果（生き・死に・コウ）を求める



— ゲーム木探索（$\alpha-\beta$枝刈り法）の並列化アルゴリズム

図10 詰 碁

(3) 詰込みパズル

次は一番性能の出やすい詰込みパズル（ペントミノ）の問題である。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が，色々な形の違うプラスチックのピースが12個ある。これを長方形の枠の中に詰め込むもので，この問題では数千通りの方法があるが，これを並列で求める問題である。これはOR並列型の全解探索といい，探索木の異なる枝どうしでは，お互いに通信しなくても探索が進められているので，仕事をバラ撒くのが楽になる。したがって台数効果の出やすい問題である。この問題を使って動的負荷分散と静的負荷分散の実験を併せて行った。

問 題

・長方形の箱へピースを詰込むのに何通りあるかを求める

— OR並列型の全解探索

— 動的負荷分散と静的負荷分散

図11 詰込みパズル（ペントミノ）

(4) 最短経路問題

最短経路問題であるが，これはネットワーク上の，各枝にコストが与えられている場合に，ネットワーク上に始点と終点を適当に設定したときに，始点から終点に至る経路の中で最もコストの少ない経路を一つ見つけよという最良解探索問題である。ここでは並列アルゴリズムの工夫とこの種類の問題向きの負荷分散の検討を行った。

以上の問題は，問題としてはシンプルなものでTOY プログラムではないかといわれるが，プログラム自体は結構大きいものが含まれており，KL1ソースの行数で1500行から800行程度である。また最短経路の問題では最大で40,000ノードのネットワークを実験しており，こうなるともう，たとえTOY Program であったとしてもToy Problem ではない。

問　題

・与えられたグラフの始点ノードからの最短経路を全てのノードについて求める

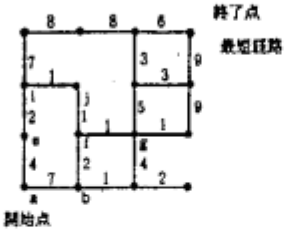


図12

— 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

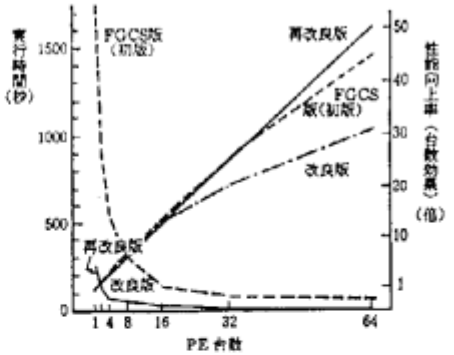
(5)　開発工数と開発量

プログラムの初版を開発するのに四つの問題に対する開発者数は8名，期間は3ヶ月であった。並列の複雑なプログラムでこの人数でできたとは驚く人も多いが，感触としてはKL1 は結構使いものになる。書き易さはまだ改善の余地がある。オブジェクト指向の機能を入れればもっと良くなるとの声もあるが，思いの他使いものになっている。特にプライオリティの機能が良かったし，KL1 はプロセス指向のプログラムには好適である。負荷分散の実験にも好適であり，これは負荷分散の指定（プラグマ）がプログラムの構造と独立して着脱できることに因っている。特筆すべきは通信，同期のバグが皆無であったことである。これはプラグマをつける前の，プロセッサへのマッピングとは関係ない論理の世界で，すなわち GHCのレベルでデバックをしておいて，そこでムシのなくなった状態でプロセッサのマッピングをつけて並列のマシンの方へ持ってゆくというアプローチが取れることによる。GHC のデバックのところで通信，同期に関係しそうなバクが殆ど出てしまう。従って実並列マシンにもっていってから並列のデバッグで苦しまないですむ。

(6)　性能評価の中間結果

右は詰込みパズルで一番性能の出たものである。左軸は「プロセッサ台数が増えればこのように時間が減る」ということである。右軸は性能向上率で台数効果と呼ばれるものである。プロセッサ台数に対し，性能がどれだけ上がったかというグラフである。

FGCS版が最初のもので実行時間は図2.3.13の通りであるが，台数効果は結構良かったものの実行時間がかかりすぎている。下手なプログラムで実行時間がかかっているが，台数効果は比較的良好である。改良版では実行時間は減ったが台数効果は頭う



詰込みパズル(5×8マス，10ピース)

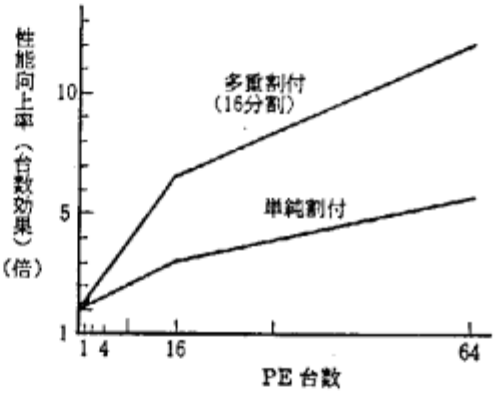
-- FGCS版，静的負荷分散
-·- 改良版，動的負荷分散
—— 再改良版，（二階層）動的負荷分散
・改良版は速くなったが，台数効果は下がった。
・再改良版で台数効果も改善

図13　性能評価の中間結果

ちになった。これは動的負荷分散というものを入れたのであるが，負荷分散のセンターが分問題を作ってはヒマなプロセッサに仕事をバラまいてはいたものの，64台の面倒を見ているのでセンター機能がネックとなっていた。問題を要求されてもセンターが追いつかないという事態が発生していることが判った。再改良版ではセンター部分を階層化し，2段階動的負荷分散というものを用いることにより最高性能を記録することができた。64台で50倍の性能が出ており，MIMD型プロセッサでこれ位の性能が出ると大変なものである。どんな問題でもこのくらいの性能が出れば，すべての人が並列マシンを使用するようになると思う。

一方最短経路の問題での台数効果であるが，単純割付けで64プロセッサを用いて性能は6倍位しか上っていない。この問題は理論的に解析してもN台のプロセッサに対して $\sqrt{N}$ 倍の性能しか上らない。アルゴリズム自体その程度しか向上しないアルゴリズムになっている。これは問題自体の難しさという方がいいかも知れないが，負荷の割付けの仕方を工夫することによって倍ぐらいの性能向上が期待できるが，それでも $\sqrt{N}$ に比例するような性能の上昇である。

構文解析のPAX の場合には，単純な負荷分散であると，性能は0.45倍におちてしまう。これでは分散しない方が良いということになる。通信を極所化するように負荷を分散する。寄せるものと分散するものをうまくミックスさせても，16プロセッサの場合でも1台の3倍くらいの性能しか出ていない。64プロセッサにしても3倍から6倍くらいにしか向上しなかった。これは問題自体のアルゴリズムがネットワーク結合型のMIMDマシンにはつらいアルゴリズムになっているという例になるが，このPAX のアルゴリズムというのは1台で解いても非常に効率のよいアルゴリズムである。これは従来の構文解析のアルゴリズムに比べれば2桁か3桁くらい良いアルゴリズムになっていて，1台でやっても速く解けるので並列にする必要がないといえるかも知れない。一方，曖昧性の大きい構文解析結果の出るような文章を同時に多数投入するとプロセッサは効率よく動くので，機械翻訳などで異なる文章をパイプラインで投入して構文解析をやってゆく場合にはこのマシンでも効率よく使えるのである。一つの文章だけを並列で解こうと思えばこの程度しか性能が出なかったということである。



最短経路問題（10,000ノード）

・単純割付ではPE台数nに対しほぼ $\sqrt{N}$ の性能向上
・多重割付ではさらに倍ぐらいの高性
・最新の実験では64台で20倍の性能向上

図14

6．今後の予定，もう少し先のこと

第五世代コンピュータプロジェクト終了後どうなるかということについて，私見を述べるならば，並列処理技術のうちハードウェアはその作り方が示されたというステータスであると思う。ソフト研究開発用マシンの安定供給を行うことがその課題であり，さらに小型化を追求する必要があるだろう。一方ソフトウェアでは問題点が明らかにされ，解決手法のいくつかが示されたというステータスではないかと思う。研究の展開，汎用化のため，まだ10年位は必要という段階ではないかと思う。これはやはりNew Culture と呼ばれるゆえんのものであるという感じである。したがって並列ソフトに力を入れ，そのためより本格的な応用の並列化を行ってゆかねばならない。その一つとしてLSI-CADのシステムをPIMの上に乗せたいと思う。その他自然言語処理については，現在構文解析だけがのっかっているが，もっと多くの項目をのせてゆきたいと思う。また詰碁ではなく囲碁の対局全体のシステム ― 新しい強調型の問題解決のパラダイムとなるものであるが，この研究を行いたい。この他定理証明，エキスパートシステムの並列化などやるべきテーマは多い。

もっと先のこと・・・

現在４Ｍチップが出現する状況にあるが，FGCS'92 の頃には16Mも出現する雲行きである。90年代後半にはクラスタ・チップも夢ではない。プロセッサとキャッシュのペアが８個，共有メモリ側の大きなキャッシュとネットワークインタフェースがのっかっているチップである。最近のマイクロプロセッサはキャッシュまで一緒に搭載されるものが一般的となっているが，100万トランジスタを少し下廻るくらいのトランジスタ数が１チッププロセッサとして商品化されてきている。このようなものが10個分ぐらいまとめて１チップにのっかるとクラスタチップと同等の集積度となる。集積度は３年に４倍になる位いで，これが２世代まわると90年代後半にはクラスタチップというものの出現も可能と思われる。これが実現するとすれば１チップで８プロセッサ（プロセッサ当り50MIPS）最大400MIPS くらいのクラスタチップもできるものと思われる。これが実用化されだすとデスクトップマルチプロセッサ・ワークステーション(W/S)(400MIPS)のものも生産可能となる。現在のパソコンはマルチプロセッサにおきかえられるであろう。さらにデスクサイドの

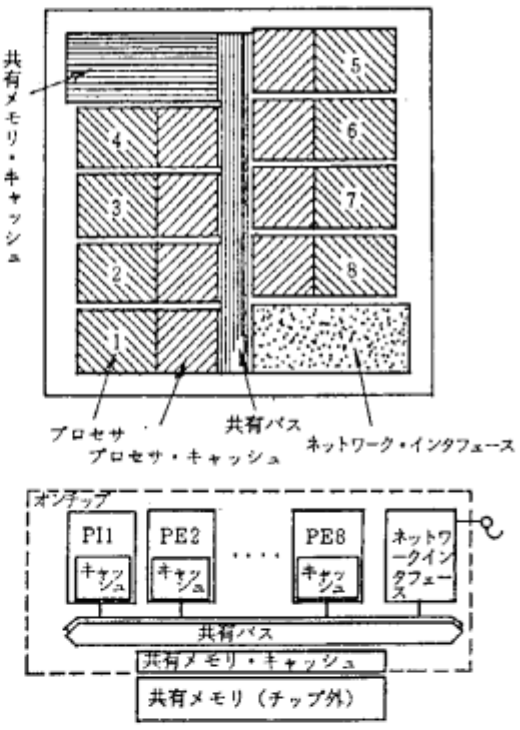


図15　クラスタ・チップ

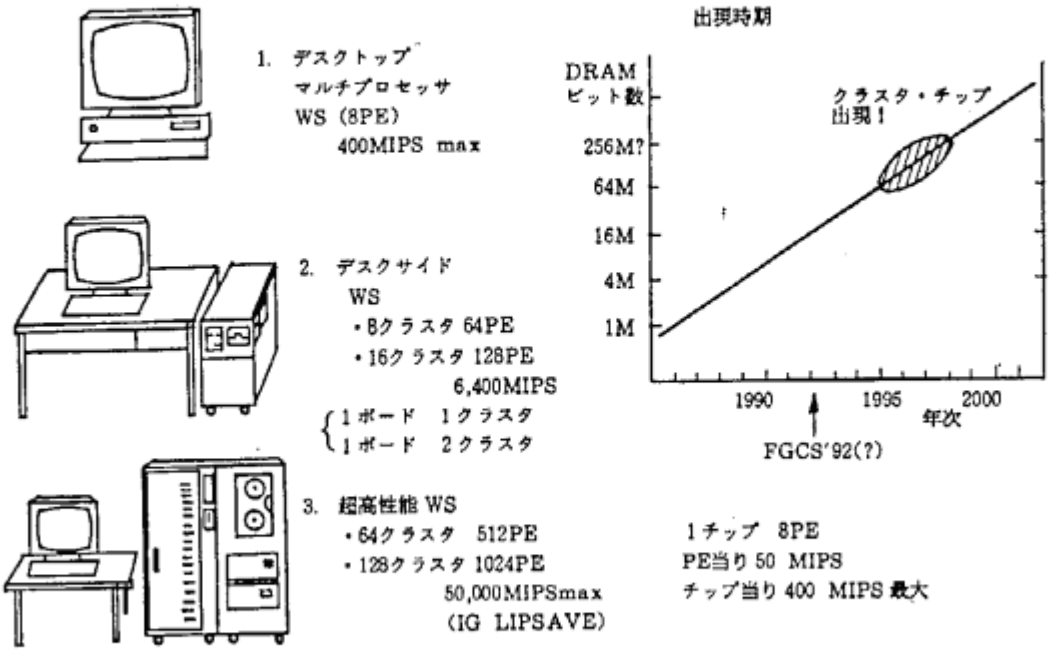


図16 システム構成（クラスタ・チップ使用時）

図17

筐体に8クラスタあるいは16クラスタつめこむと最大で6400MIPSくらいになる。また大型筐体を用いディスクやMTをつけてゆくと，これは今のSUN-W/Sのフルセットに当たるようなW/Sの大きなもののイメージとなり，超高性能W/S ができるであろう。この中には64クラスタから128クラスタが入る。 128クラスタは1,024プロセッサであり，ICOTの最終目標としているマシンの姿がこの位の大きさで実現できることになる。性能はMIPS値で50,000MIPS，推論性能は1GLIPS — 現在の目標最大値 — が平均アベレージでできるのではないかと思われる。このようなハードがはたして思いどおりに動くかであるが，大規模MIMD型の並列マシンを使いものにならしめる技術がしっかりしないことには折角できたハードも動かないということになる。

並列のプログラムがきちんと作れる技術，すなわちコンカレントプログラミング技術であり，また負荷分散，スケジューリング，並列OSの技術，すなわちパラレルプロセッシング技術，さらにそれから応用がついてくることであり，さらに安く作る技術が必要とな

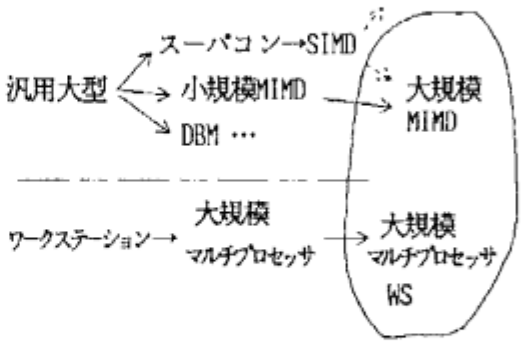


図18 MIMD型マルチプロセッサの占める位置

る。ICOTは将来の汎用大規模並列マシンのための並列処理というものを念頭において論理型言語に片寄らずに研究を行っている。ここでMIMD型大規模プロセッサはどのような位置付けになるのかを考えてみる。W/Sの方は小規模マルチプロセッサW/Sに移行し，これが大規模化してゆくことになるだろう。現在10プロセッサくらいのものが試作されているし，30プロセッサくらいのものはW/S ではないが市販の共有メモリ型システムとしてすでに存在している。汎用マシンはスーパーコンや科学技術計算用のものもあるが，汎用マシンの本体は IBM などすでに12プロセッサくらいまでの並列（共有メモリ型）のものを考えている模様である。汎用マシンはいずれ小規模なMIMD型へ移行してゆくのは確実であろう。現在の複数のプロセッサの使用方法は，互いに独立なタスクを別プロセッサで実行し全体としての性能を高める話しが中心であるが，一つの仕事を並列で行うというのはその次にやってくるだろうと思われる。さらにその次は大規模なMIMDにつながってくると思う。ICOTでやっている大規模なMIMDマルチプロセッサの技術は，ワークステーションの将来と，汎用大型マシンの将来の両方に適用できる技術と考えている。

先に述べたクラスタチップは，90年代後半には確実に実現すると考えられるが，現在行っている並列処理の研究はこれらを動かすために必須の技術になるはずのものである。