報　告

パネル討論会

並列計算機の実用化・商用化を逡巡させる諸要因とは

——その徹底分析と克服——

並列処理 シンポジウム JSPP 92† 報告

パネリスト
稲上　泰弘[1)]，　小柳　義夫[2)]，　笠原　博徳[3)]
島崎　眞昭[4)]，　高橋　延匡[5)]，　瀧　和男[6)]
山田　実[7)]，　吉岡　顕[8)]
司会　富田　眞治[9)]

本稿は1992年6月15日から17日までパシフィコ横浜で開催された並列処理シンポジウム JSPP 92 の中で行われたパネル討論の要約である．このパネルは8人のパネラと300人に迫る参加者を得て，3時間近く，白熱した議論が展開された．残念ながら頁数の関係で，後半の議論と，パネラの発言の一部が割愛されている．

なお，JSPP 93 は1993年5月17日から19日まで早稲田大学総合学術情報センター国際会議場で開催される予定である．

1. はじめに

司会(富田)　ご承知のように CM-5 とか，Paragon とか，MasPar とか nCube とか欧米でかなり大きな並列計算機が出てまいりまして実用化がどんどん進んでいます．ところが日本の現状をみてみますと，この JSPP に最終的に参加された方は306名ということでものすごく盛り上がった（拍手）のですが，残念ながら日本で並列マシンの実用化はいかんせんなかなか進んでいません．非常にもどかしい思いを皆さんしておられると思います．こういう状況をぜひとも打破をしたいということでこのパネルを企画したわけです．

このまま放っておいてもとにかく並列処理の時代になるんだよという楽観的な見方もあるでしょうし，IBM とかCRAY 社がスーパコンピュータを出せば日本も追随をして並列処理の時代になるんだよという文化の後進性を容認するような見方もあるでしょうし，あるいは日本では，情報処理学会などの学会を通して，もうちょっと並列プログラムの共用化とか標準化とか，そういった面で強力にプッシュをしていかないと並列処理の時代はこないんだよ，というふうないろんなご意見があるかと思います．

きょうはまずは並列処理が日本で実用化が遅れている，研究は進んでいるんですが実用化が遅れているという点について，メーカ側の問題点はどういったところにあるのか，ユーザ側の問題点はどういったところにあるのか，大学などの教育研究機関，大学の大型計算機センタなどでの専門教育あるいは一般教育でどういった問題があるのか，といったところをメインのテーマとしてパネルディスカッションをしてみたらどうかと思っております．

きょうはパネリスト8名来ていただきまして，大きく分けまして向こう側におられる方が悪役，こちらの側の方が善玉でございます（笑）．

パネラの方を紹介したいと思いますが，まずは悪玉の最右翼ということで日立製作所の稲上さん(拍手)．並列計算機をなぜ商用化しないのかについて，メーカだけが悪いんじゃなくて，ユーザにも大いに責任がある．あるいはマーケット側にも問題があるといったようなテーマでスピーチをしていただけるんではないかと思います．

† 日時　平成4年6月16日（火）16：00〜18：30
場所　パシフィコ横浜　301＋302 会場
1) 日立，2) 東大，3) 早大，4) 九大，5) 農工大，6) ICOT，
7) 日本 TM，8) 東大，9) 京大

次が九州大学の島崎先生（拍手）．大型計算機センタでスーパコンピュータの研究をしていらっしゃいます．なぜ国立大学の大型センタにベクトルタイプばかりで並列計算機が入らないのか．文部省の予算が少ないからじゃないかとか，やっぱりベクトルタイプが汎用的だよ，といったようなお話をいろいろしていただけるんじゃないかと思います．

3番目が東京大学の小柳先生（拍手）．小柳先生は現在並列計算機 PAX で著名な先生でいらっしゃいますけれども，きょうは昔の名前で出ていますということで，ユーザがベクトル計算機から離れられない理由は何かということで，ベクトル計算機が普及したころのいろいろな経験を踏まえて並列計算機の現状を批判していただこうと思います．

4番目は東京大学の吉岡先生（拍手）．吉岡先生にはユーザからみたときの並列処理方式あるいは並列計算機の問題点を厳しく追求していただこうと思っております．

以上が悪玉の方々でございまして，これに対して善玉の方々ですが，並列計算機の将来はやっぱり明るいよというお話が中心になるかと思いますが，そのトップバッタとして早稲田大学の笠原先生（拍手）．並列化コンパイラこそ我が命というふうなお話が多分あるんじゃないかと思います．

それから ICOT の瀧さん（拍手）．ICOT の並列処理の研究の総括的なお話も含めて将来は明るいといったお話をしていただけるんではないかと思います．

次は日本シンキングマシンズ社の山田さん（拍手）．コネクションマシンはなぜこれだけよく売れるのか(笑)というお話をぜひしていただきたいと思っております．

最後は東京農工大学の高橋先生（拍手）．高橋先生は「情報関連学科のカリキュラムからみた並列処理教育の現状」についてお話ししてくださると思います．特に先生は，文部省の委託を受けて情報処理学会の中でカリキュラム検討があり，そこで重要な役割を果たされてきております．学会誌に検討された記事がすでに出ていますが大学教育，特に並列処理教育についてコメントをいただきたいと考えております．

それではトップバッタの稲上さん，よろしくお願いします．

2. ベクトル計算機から超並列計算機へ —置き換えのシナリオ—

稲上　悪玉の一番手として登場いたします稲上でございます．日立製作所の中央研究所にてベクトルプロセッサの研究開発に従事しております．

話を始める前に私の経歴を紹介させていただきますが，私は 79 年に日立に入りまして，以来ベクトル計算機を3世代つくってきた人間でございます．

そういう経歴をお含みおきのうえ，私の話を聞いていただければと存じます．

2.1 スーパコンピュータの性能トレンド

これ（**図-1**）はスーパコンピュータの性能トレンドを示すグラフでありまして，いまさら特別に申しあげることもございませんが，一番下の細い弱々しい線が私の青春をかけてまいりましたベクトル計算機のトレンドでございます．

このトレンドを持ち上げておりますのがベクトル計算機をマルチプロセッサ構成にしたマシンの性能で，昨今の主流のスーパコンであります．それに比べまして非常に勢いよく性能が向上してきているのが，高速の RISC プロセッサを要素プロセッサとして用い，急速に高速化が図られております超並列計算機でございます．このグラフを見て分かりますとおり，ピーク性能でのみスーパコンを云々するなら，もうベクトル計算機の時代は終わったといわざるをえない状況でございます．

もう一つ重要なファクタはコストパフォーマンスで，これは各マシンの正確な値段を把握するのがなかなか難しくて，ザックリと書いたグラフですが，一般的には超並列のほうが広い性能レンジをカバーし，かつベクトルプロセッサに比べコストパフォーマンスが一桁よいということになります．本当にそうか，議論のあるところですが，事実として認めるべきだと考えております（**図-2**）．

製品動向ですが，アメリカの会社から非常に優秀な計算機が市場に投入され，現在は3世代目のマシンが世に出ている状況です．ベクトル型のスーパコンをつくってきたメーカからの製品出荷

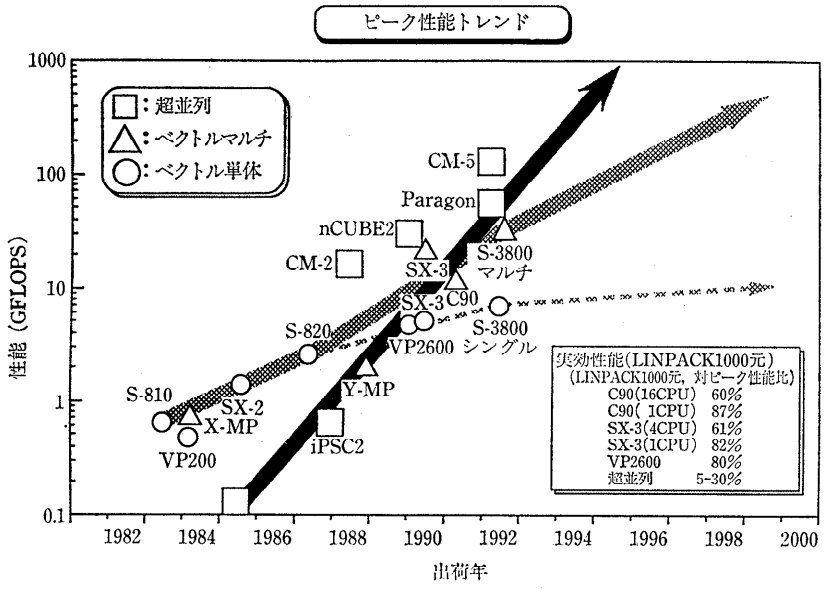


図-1

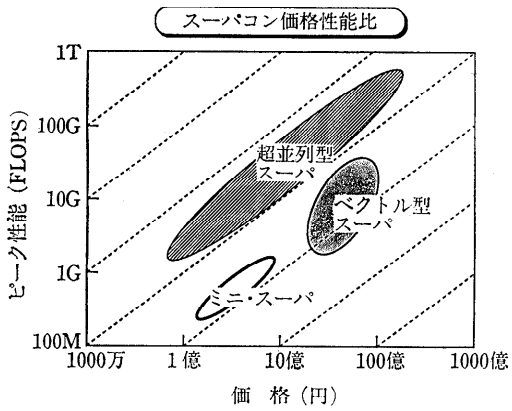


図-2

はまだありませんが，開発中とのうわさ話は多々あり，商用並列機を着々と準備中というのが現状ではないかと思います．

2.2　市場規模はどうか？

本日私はメーカの立場から話をさせていただいているということもありますので，市場規模といったところに少し触れさせていただきたいと思います．

市場規模の把握も非常に難しいんですが，世の中で言われているいくつかの話を私なりに分析してスーパコンの市場規模をまとめてみたのがこの表（**表-1**）です．

並列計算機というのは 90 年の時点ではスーパコン市場の 10% に満たない規模の小さい商売です．95 年になってもそんなに増えないという予測が一般的でございます．並列計算機の市場は立ち上がるのは 90 年代後半であり，ベクトルプロセッサの市場規模に追いつくのは 2000 年ごろであろうというのが一般的な見方であります．ただ，2000 年以降はベクトルプロセッサより並列処理のほうが市場規模が大きくなるというのも一般的な見方でございます．ということで，性能とかコストパフォーマンスという点では超並列がすでにベクトルプロセッサをしのいでおりますが，市場はまだまだ小さいという現状がございます．

表-1　スーパコンピューティング市場の規模

年	1990	1995
全体	2,300 億円	4,500 億円
並列計算機	200 億円 (8.7%)	650 億円 (14.4%)

じゃ，並列計算機の市場がなぜ小さいのかということですけれども，ベクトル計算機がなぜ成功したかという原因をふりかえりながら，超並列計算機がなかなか普及しない理由を私なりに考えてみたいと存じます．

2.3　ベクトル計算機がなぜ成功したか？

ベクトル計算機が世に出たのはアメリカが 1976 年，国産が参入しましたのが 1983 年でございますが，その時代はいまと大分違いまして，汎用機全盛，集中システム指向といった状況でございました．そこへベクトル計算機が登場するわけ

ですが，ベクトル計算機はきちんと使うとそれまでの汎用機に比べて格段に速かったという事実がございます．10 倍，使い方によっては 100 倍高速化されることもよくありました．計算機の世界でこれだけ倍率の高い高速化計算が急に達成できたのは歴史的にみてもめずらしいことで，ベクトル計算機の市場が急に開けた第1の原因がここにあったのではないかと思います．

次に，いまにして思えばベクトルアーキテクチャの完成度が非常に高かったといわざるをえません．とにかく亜流を許しませんでした．各社ベクトル計算機を製品化いたしましたけれども，カタログのシステム構成を見ますとほとんど同じでございます(笑)．各メーカ互いに相談をして開発したことは当然のことながら一度もございませんで，これはベクトルアーキテクチャを本格的に商用化したセイモアクレイ博士の偉大さに敬意を表さざるをえないということになります．とにかくベクトルアーキテクチャは非常に完成度が高く，どのマシンを使おうとも，その利用技術には共通部分が多く，ユーザも安心して導入できたのだと思います．

それから，ハードウェアの市場投入に合わせ，自動ベクトル化コンパイラというソフトウェアも出荷され，ユーザは標準の FORTRAN で書かれたプログラムをとりあえずそのままの形でベクトル計算機に乗せることができたということが非常に大きなポイントだったと思います．

ベクトル計算機という新システムへユーザがどう移行していけばよいかというビジョンが明確に示されていたわけで，これも，ユーザが大きな抵抗なくベクトル計算機を受け入れていった大きな要因であったといえます．

さらに，アルゴリズムやソフトの開発が非常に精力的に行われました．研究から実用化までには，分野によって違いますけれども，結構時間がかかるものです．実用化を強く意識した研究開発が積極的に行われてきたことも，ベクトル計算機の普及を助けた大きな要因です．それからミニスーパコンが出てきまして利用者層が大きく拡がっていったということもあります．そういったことが全部合わさって，第3者提供ソフトが充実し，ベクトル計算機の全盛時代を迎えたのだと思います．

2.4 ベクトル計算機側の危機感

ただ，ベクトル計算機の将来については，少し危機感をもっております．汎用機の最近大型の調子が悪いようでございます．ダウンサイジングの進行という現象です．それと同じことがベクトル計算機でも起こらないかという危機感が漠然とございます．今後は，ベクトル計算機においてもダウンサイジングが重要な課題になってくることは間違いありません．

また，超並列とまではいきませんが，並列化にもチャレンジする必要があります．ベクトル処理に小規模並列の技術をつけ加えるというマイナチェンジで過去の遺産を重視する形でまだもうしばらくは頑張れるんじゃないかと考えています．また，頑張りたいとも思っています．性能，コストパフォーマンス共にすぐれた超並列機が登場したからといってベクトル計算機が急になくなるというわけではありません．以上説明いたしましたように，世の中に急速に受け入れられていったベクトル計算機でございますが，じゃ並列計算機のほうはどうかということを少し述べてみたいと思います．

2.5 超並列計算機の問題点

私は悪玉でございますので若干独断の入った，超並列には厳しい話も入ってくるかもしれませんが，その点お含みおきのうえ，お聞きいただければ幸いです．超並列計算機の良い点は以前にも述べたとおりです．一方，超並列計算機の普及を妨げる要因ですが，一つは，ベクトル型スーパコンと市場が競合することがあげられると思います．超並列計算機がただちに新たな市場を作り出すわけではないということです．事業という側面を考えた場合，何かすみ分けの戦略が要るということになります．

次に，ベクトル計算機が世に出たときに比べまして，主流のアーキテクチャや並列モデルがみえないという問題があります．ユーザとしては，新しいアーキテクチャをもつ超並列計算機向きにソフトウェアを開発してよいものか不安が残り，その利用に際し決断がにぶる場合が多いのではないでしょうか．実際，超並列計算機への移行のビジョンが明確に示されていないのが現状で商品として位置づけがしにくいという状況にあり超並列

計算機の性能を十分に引き出す利用技術につきましてもまだまだ未熟なところにあるといわざるをえないという気がいたします．いつごろ成熟するかも明言できないのが現状ではないでしょうか．そういった状況ですから，第3者ソフトの開発もまだまだ消極的で，普及させようにもそう簡単にはいかないというのが現実です．そうはいっても超並列計算機は非常にいい特性をもっておりますので，全然だめだとは思っておりません．有望なアーキテクチャだと考えております．

2.6 ユーザ層の動向

高性能コンピュータのユーザを大まかに分類してみますと，だいたいこの図（**図-3**）のようになるのではないかと思います．

まず先端的な研究をされている先進ユーザの方々ですが，ソフトウェアの開発力も含めまして非常に高い技術をももっておられます．資金力もおありで，とにかく高性能のコンピュータを必要とされていますから，超並列計算機が最も必要とされ，かつすんなり普及していくユーザ層だと考えています．

第2番目の，性能重視のユーザ層ですが，高精度の解析が主要な業務であり，やはり超高速のコンピュータを必要とする領域です．

現在はベクトル計算機が広く使われている領域です．

このユーザ層の方々は，自らプログラムを開発されますが，一方，信頼性が高く使い込まれている流通ソフトを非常に重視されます．

信頼性の高いソフトウェアパッケージを作ることはなかなかむずかしいことであり，コストパフォーマンスの良い計算機が登場しても，それがまったく違うアーキテクチャをもつ場合はそう簡単に移行できるものではありません．

ただし，非常に高い性能が必要とされる領域ですから，いずれは超並列計算機が使われていくことになるでしょう．3番目のユーザ層は，一般ユーザ層であり，応用分野も多種多様で，かつ流通ソフトを重視する領域です．このユーザ層は，高性能のワークステーションやサーバなど新しいタイプのマシンが利用できるようになり，今後ますます拡大していくことでしょう．

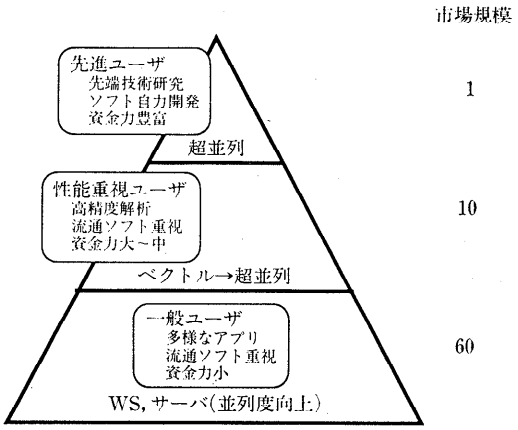


図-3 高性能コンピュータユーザの分類

ユーザの数を市場規模という指標で大まかにとらえるならば，この図のようになるのではないかと思います．

2.7 超並列計算機普及のシナリオ

次の図（**図-4**）は，高性能コンピュータ普及のシナリオを私の独断をまじえまして示したものです．

当初は一部の先進ユーザのためのマシンとして使われはじめますが，OS，コンパイラなどの基本ソフトウェアが整備され，利用技術が蓄積されるにしたがって性能重視ユーザへと広まってまいります．

有用な第3者ソフトが充実してくれば，一般ユーザへも普及することになり，世の中に浸透してまいります．私の考えでは，超並列計算機の現在のフェーズは，システムソフトが充実しつつあ

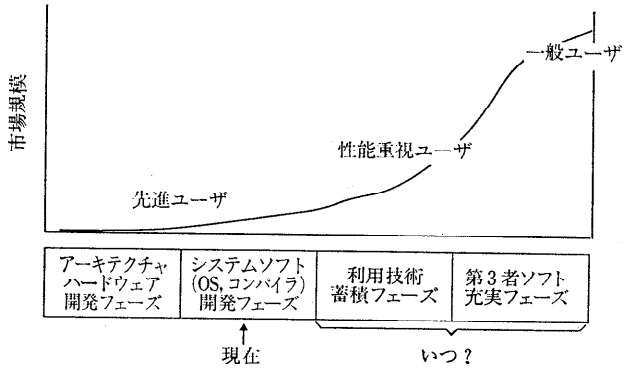


図-4 高性能コンピュータ普及のシナリオ

る段階だと思います．現在一番問題なのは，いつ性能重視のユーザ層に普及し，市場が立ち上がるかということです．

既存のスーパコンベンダはこのポイントがいつになるのか，あらゆる方面から検討を加え，市場への本格参入時期を図っている状況ではないかと考えています．

最後にまとめでございますが，超並列計算機の高い性能，良好な性能価格比，スケーラビリティの良さは十分に認識いたしております．しかしながら，性能を引き出す利用技術がまだまだ未熟であるなど，多くの問題をかかえていることも確かです．

これまでベクトル計算機を製品化していた各ベンダは，超並列計算機関連技術の進展をにらみながら市場に本格参入する時期とその戦略を練っている状態だといえます．

超並列計算機は，ベクトル計算機と市場が競合する領域がありますし，ソフトウェア遺産の継承を考えますとベクトル計算機もまだまだ必要と思われます．

システムを提供する側としては，新しいアーキテクチャをもつ超並列計算機への移行ビジョンを明確に示す必要がありますし，また，市場規模拡大の大きな鍵である第3者ソフトの充実に向けた利用技術の蓄積も精力的に取り組むべき課題です．

以上でございます．

3. 文献から探るスーパコンピューティングの研究動向

司会　どうもありがとうございました．引き続きまして島崎先生．

島崎　ご紹介いただきました九州大学の島崎でございます．

スーパコンピューティングの研究は，コンピューティングのほかに，並列コンピュータ，特にここでいっているのは先ほども出てきましたように，共有メモリ型の少数並列方式ではなくて，マッシィブリパラレル方式を主な対象にしているのだと考えます．少数並列のベクトルスーパコンピュータですと実際に製品も出てきて，財政的な余裕があれば入れられる状況になっているし，日本の大学の中にも今年度あるいは来年度になれば，入っていくのではないかと考えられる状況にあります．ここではマッシィブリパラレルの問題を少し考えてみたいと思います．どうして入れられないかということなのですが，そのことを歴史的に考えてみたいと思います．

3.1 文献データベース INSPEC

いまではベクトル型のスーパコンピュータというのは科学技術の分野であたりまえのことになっています．しかしたとえば 1974 年，5年ごろ，といいますと CRAY-1 がつくられる直前になりますが，そうではありませんでした．そのころちょうどたまたま1年間外国に留学していましたが，そのときに現在と同じような状況といいますか．当時ベクトルスーパコンピュータが日本ではどうして実用化されないんだろうかともどかしいような思いをしたことがございました．その事情が現在と少し似ているように思ったわけです．

それについてはいろいろ感じているのですが，感覚的なことでお話ししても説得力という点で問題があるかと思います．少し考えてみまして，文献データベースを調べて実証的にそういった事柄が出てきはしないかということで，INSPEC という，イギリスの電気学会が出している文献データベースがありまして，INSPEC A というのが物理学，INSPEC B が電気工学，INSPEC C が制御及び情報工学ですが，その中に含まれている文献を分析して何か出てこないかということで調べてみたわけです．

INSPEC のデータベースはAですと年間5万件入っているわけで，1970 年から 1992 年まで合計すると相当膨大な文献となります．件数的にみると一番上がAで物理学，それからB，Cが電気工学と制御工学・情報工学でして，制御工学・情報工学関係は昔は少ないけれども，伸び率はほかのものよりも多くなっているという状況です（**図-5**）．

もう一つここで述べておかなければならないことは，INSPEC は文献データベースということで，参考文献でしたら実際に論文が出てから収録されるまでに年月がかかるということです．実際に 1991 年の文献数は必ずしも十分ではなくて 1992 年はもちろん少ないわけです．ですから 1990 年までで見るのが正当ではないかというこ

とで，少し現在とはへだたりがあるかと思いますがお話をしてみたいと思います．

3.2 キーワード「スーパコンピュータ」で調べた文献

図-6 はスーパコンピュータというキーワードをもつ文献の数を調べたものです．

これを見ると分かりますように，文献は 1976 年ぐらいから出始め，INSPEC C の文献がかなり出てきているわけですが，非常に増えだしたのはやはり 1983 年，84 年，日本のスーパコンピュータが出始めたころで，それ以後非常に増えています．そして若干参入が遅れるようなかっこうで，INSPEC A, INSPEC B すなわち，物理学，電気工学の文献もふえてきています．実際に計算機工学が進んで，システムとして実用化され，実際に使われて応用分野の論文が書かれるようになるということがよくあらわれていると考えられます．

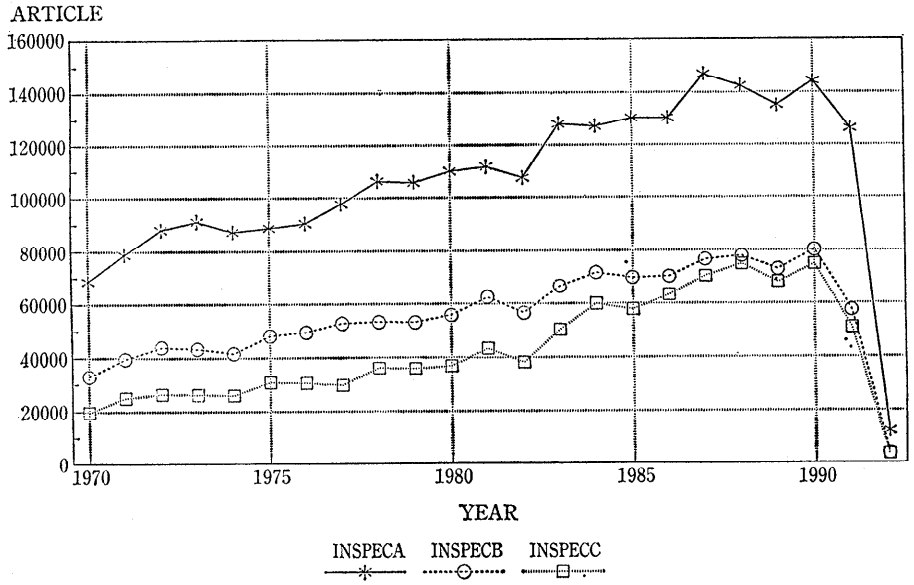


図-5 INSPEC の論文数の推移

Supercomputer(s)

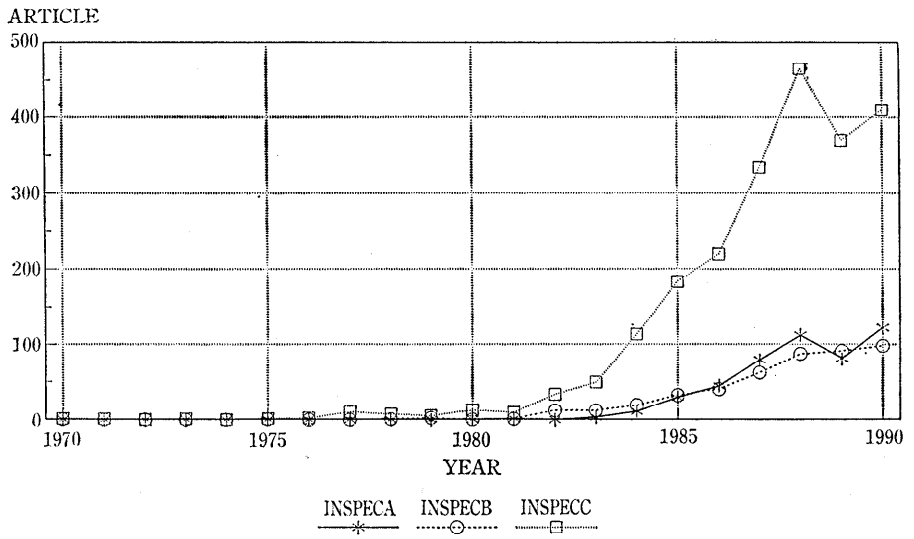


図-6 Supercomputer(s) をキーワードにもつ論文

3.3 キーワード「超並列」ではどうか？

その次にマッシイブリパラレルをキーワードとして文献を探してみますと，やはり同じような傾向になります(図-7)．ところがスーパコンピュータと比べると何年かの遅れがあります．計算機工学，INSPEC の分野では 1986 年，87 年ぐらいから急速に立ち上がっている．絶対数は必ずしもまだスーパコンピュータに比べて大きくはありませんが，急速に立ち上がっているということがいえます．ただし，INSPEC A，INSPEC B についても立ち上がっておりますが，まだその数は必ずしも多くない状況です．

3.4 キーワード「ベクトライジングコンパイラ」に対する論文

いまの状況から，まず INSPEC C で論文が増えて，そのあと INSPEC A，INSPEC B で論文が増えているという傾向が出ています．スーパコンピュータが普及するときに自動ベクトル化コン

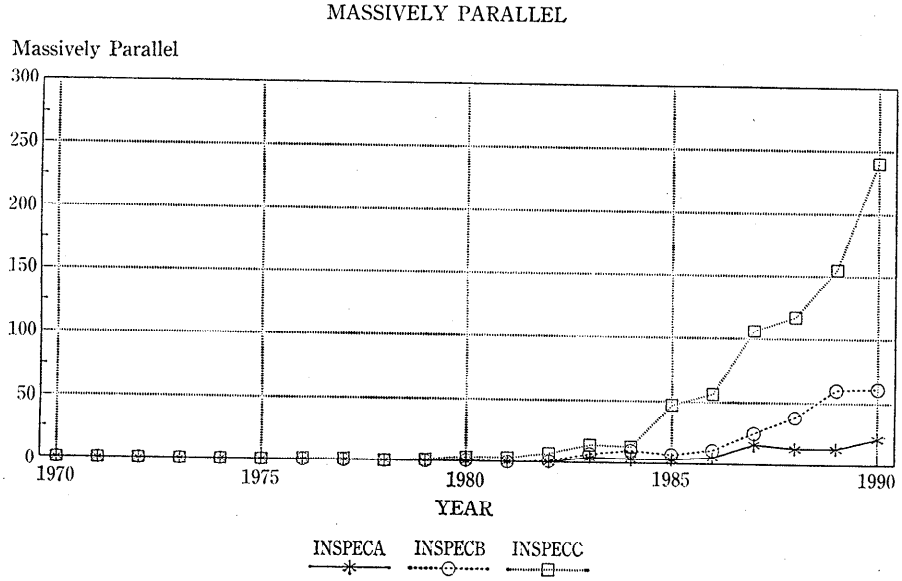


図-7 Massively Parallel をキーワードにもつ論文数

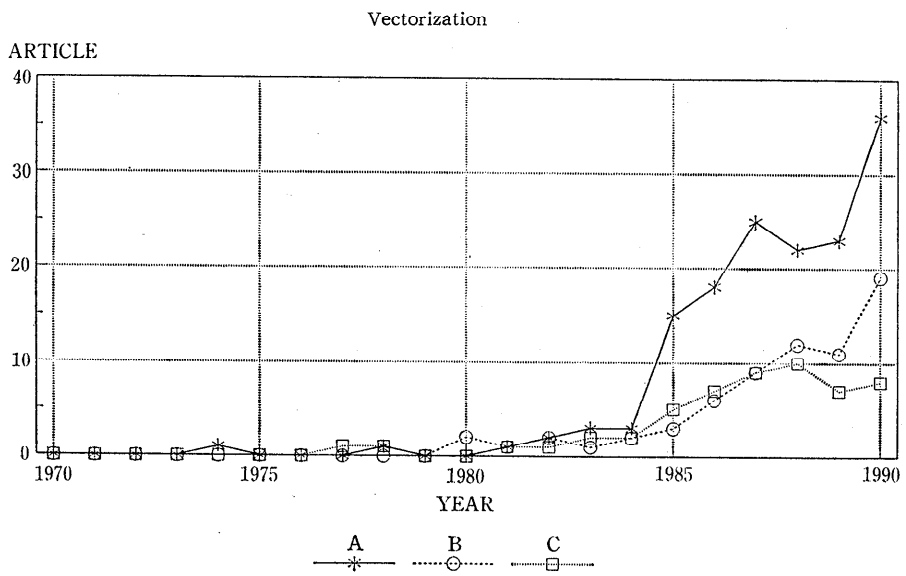


図-8 Vectorization または Vectorizing Compiler をキーワードにもつ論文数

パイラが非常に貢献したということがよくいわれるわけですが，それを裏づけるデータがあるのではないかということで調べてみたのが次のグラフです（図-8）．INSPEC C では「ベクトライジングコンパイラ」をキーワードにもつ論文の数を，INSPEC A, INSPEC B については「ベクトルライゼーション」をキーワードにもつ文献の数を示しています．

ベクトライジングコンパイラの論文は1984年，85年ぐらいからずっと増えていまして，1989年ぐらいでむしろ飽和してきているという感じです．1988年ぐらいが一番多く，技術的に相当の水準に達したと考えられます．ベクトライゼーションという応用の文献はずっと伸びているという状況です．やはり INSPEC C で先に論文が出て，その後 INSPEC A, INSPEC B で論文がふえています．

3.5 キーワード「並列化コンパイラ」はどうか？

それに対してパラレライジングコンパイラあるいはパラレライゼーションというキーワードをもつ論文を調べてみました（図-9）．パラレライジングコンパイラですが，1988年ぐらいから立ち上がりかけていてベクトライジングコンパイラに比べて時期的にかなり遅れている，または研究が最近始まったと考えられます．INSPEC A, INSPEC B でも論文が増えつつありますが，まだそれほど絶対数が多くないという状況だと思います．

現象論的な話ですが，ベクトライジングコンパイラの論文を見ますと，アメリカの研究者によって書かれたものが 55% ぐらい，日本の研究者によって書かれたものが約 35% あります．これに比べてパラレライジングコンパイラのほうでは 70% がアメリカの研究者によって書かれ，日本の研究者によるものは 6.4% で，その他のスイスとかドイツとかとあまり比率は変わらない（図-10）．まだ日本でもパラレライジングコンパイラの研究は必ずしも十分ではないと思われる状況です．

3.6 計算センタの使命

スーパコンピュータのセンタですが，スーパコンピュータのセンタは(1)応用センタたとえば，アメリカのイリノイ大学の NCSA* と，(2)スーパコンピューティング研究開発センタ，たとえばアメリカのイリノイ大学の CSRD あるいは EC の CERFACS の二つに分けられます．ところが日本では大型計算機センタなどは1番に位置づけられ，2番のように実験的なスーパコンピュータを入れるには，ちょっと目的ないし使命が違っている感じをもたれています．2番のようなセンタがつくられる，あるいは情報工学研究者のための

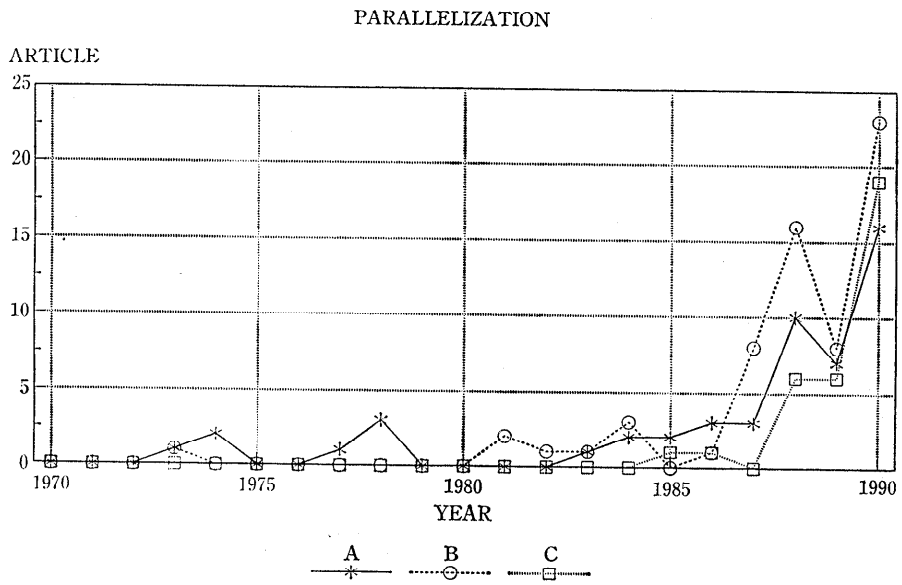
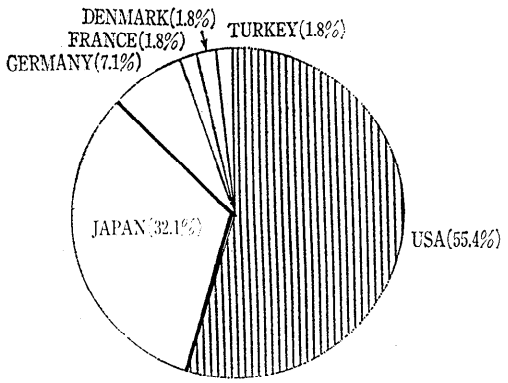
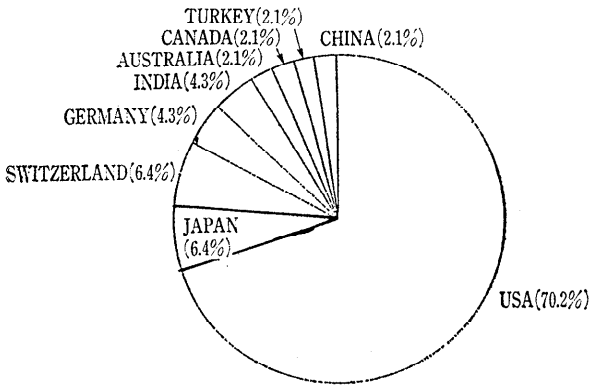


図-9 Parallelization または Parallelizing Compiler をキーワードにもつ論文数

* National Center for Supercomputer Application



VECTORIZING COMPILER



PARALLELIZING COMPILER

図-10 Vectorizing Compilers(s), Parallelizing Compiler(s) をキーワードにもつ論文の著者の国別の調査

スーパコンピューティングセンタができることが望ましいと思うのです.

1番に分類されるセンタは，利用者が多いわけですから，(1)システムの最新性の維持 (2)既存ソフトウェア資産の継承，(3)システムの将来性の見極め，これらが非常に大事になります．特に2番目が重要です．こういうところに入れるためには，絶対性能がたとえば TFLOPS ぐらいが望ましいです．しかし，たとえば 0.1 TFLOPS 級の機能が現在実用的にあって，多くの分野でそれだけの性能が得られれば，これはもうすぐにでも導入できる．予算の問題さえ片づけば入れられるんじゃないかと思います.

先ほどのセンタの性格もありまして，共用システムの利用という形態で，利用者としての範囲で使うことになり情報工学の研究者が実験的研究に使いにくいということがあります．しかしそのような情報工学の実験的研究を行うユーザが集団として多数おられるようなら，実験的システムも入れられるのではないか．そのような利用者集団が成立していないのが導入しにくい原因の一つではないかと思われます.

3.7 ベンチマークの重要性

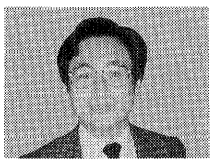
あともう1点だけ指摘して私の時間を終わろうと思います．スーパコンピュータの世界ではベンチマーク，たとえばイリノイ大学の CSRD でつくっている Perfect ベンチマークがあります．Perfect ベンチマークに対していろんなスーパコンピュータの性能が出ています．(1)ソースプログラムに手を加えないで得られるベースラインと呼ばれる結果と (2)オプティマイズというプログラムにチューニングを加えた結果とがあります．普通のスーパコンピュータの場合は，Perfect ベンチマークの 13 のプログラム—これらは非常に大きいものですが，それに対してその結果がだいたいあるわけです.

1990 年のデータですが，マッシィブリパラレルの場合に 13 個のうちの3個のプログラムに対する結果があります．13 個のプログラムに対してさっと，かつそれなりにいい結果が出るという状況であればいいのですが，残念ながら 1990 年の時点ではそうではないということが問題です．膨大なソフトウェアに対してそれなりの結果をすぐに出せない状況であるというのが，計算機センタとしてマッシィブリパラレル機を導入するのを躊躇させる原因であろうかと思っております.

以上で私のお話を終わりたいと思います.

司会　どうもありがとうございました．小柳先生よろしく.

4. なぜユーザはベクトル計算機から離れられないのか



小柳　ご紹介にあずかりました小柳でございます．私も昔はベクトル計算機ユーザだったんですが，その後改悛しまして(笑)並列になってしまいましたので少し昔のことを思い出しながら発題をさせていただきたいと思います.

先ほど稲上さんの話にもありましたが，ベクトル計算機の出始めのころというのがある意味でいまの並列計算機の出始めと対応するんではないか

と思いまして，そのころのことを，お話をしたいと思います．

4.1 ベクトル計算機導入時の障害

東大大型計算機センターとか京都大学センターに入ったのは 83 年ころですが，東京センターの例では，それまでにいわゆる IAP という内蔵型プロセッサで三，四年の経験を積んでおり，しかもその研究会などがあっていろいろ情報交換をしておりました．だからセンタのユーザレベルとしてはかなりいい準備をしていたと思います．しかし 810 のモデル 20 がインストールされたときはじめはだれも使わなかったのです．

要はユーザというものは保守的なんです．

まずだいたい普通のユーザは動いているプログラムは触らない．これがまず第一原則です．自分でつくったのならまだいいんですが，先輩からもらったとか，ましてや買ったものだとまず手を入れられない．これはもう皆さんも気持ちとしては分かると思います．

それから，当時のコンパイラはあまり賢くなかった．

もう一つ．スーパコンは速いという宣伝のために，高速化，ベクトル化の技法という講習会がいろいろと行われまして，それが逆に何か非常に難しい計算機だなという悪い印象を与えてしまったということがあります．

また，先ほどの保守性と同じことですが，だれか使ってみてうまくいったという話を聞いてから使おうという気持もあったでしょう．結局多くのユーザの研究者は，非常に計算機を駆使している人でも，コンピュータというのは必要悪．だからそのための努力はなるべくしないでおこうというのが基本にあるんではないかと思います．

4.2 いかに沈滞を破ったか

ではこの沈滞をどのように破ったか，これが問題です．これはベクトルの話ですが．

まず第1，先ほど稲上さんの話にもありましたが，パワーです．ベクトルについて何人かの先駆者と呼ばれる人がいて，私もその1人と思っているんですが，いろいろと苦労して実用計算で 10 倍から 30 倍，それ以上という性能向上を達成して，「速かった，速かった」と言い始めたわけです．その1例を簡単にお話ししたいと思います．

東大にスーパコンが入った翌年，私の先輩である筑波大学の物理学のI先生とその学生3人を東大センターに連れてきました．持ってきた格子ゲージシミュレーションのプログラムを入れましてコンパイルした，これは汎用機上で最適化したプログラムでしたが，多少ベクトルのことも考えてありました．でもコンパイルすると，スカラレジスタが足りない．そこでスカラの1，2，3をI1，I2，I3 に置き換えたが，こんどはベクトルレジスタが足りない．じゃしようがない．I3 だけは3のままにして(笑)．そうしたら見事コンパイル成功．当時高エネルギー研で使っていた M 200 H の 25 倍の速度が出たというわけです．たちまちこのI先生はスーパ中毒になってしまいました．

4.3 スーパコンピュータは3日使えばやめられない

1分の計算が3秒になってもあまりありがたくない，でも 100 時間が5時間になったら革命的な進歩だ．この仕事は当時 M 200 H で 500 時間使ってやろうと思っていた仕事だったわけですが，これが 20 時間で終わってしまった．当時はすいていたんで2週間で 20 時間使えました．500 時間の計算というと，いくら高エネルギー研のように大型計算機を自由に使える環境といっても数カ月の仕事になります．

スーパ中毒はたちまち伝染しました(笑)．多くのユーザは見よう見まねでコードを修正して，おまじないのようなディレクティブを挿入して…．そうすると驚くほど高速化するわけです．「スーパコンピュータは3日使うとやめられない」というような調子でございまして，ユーザがあっというまにふえて繁忙期にはターンアラウンドが二，三週間ということになりました．810 のときは最初はスカスカだったのですが，820 にリプレースしたときには，4倍ぐらい性能が上がったにもかかわらず最初からもうフル稼働でした．

4.4 それでは高並列計算機は？

高並列計算機も同じ経過をたどるのかなと思うんですが，アーキテクチャの変化がはるかに大きいということで，当然障壁も高いわけです．この障壁を崩すにはいうまでもなく，ベクトル型に比べてかなりの性能向上，これは絶対性能もしくは価格比という意味での相対性能の向上が必要ではないかと思われます．まず最初はセンタに高並列あるいは超並列のものを積極的に導入することが

必要です.

たとえば，ベクトル計算機で100万円だったけれども超並列ならその 10 の1ですむとなれば，プログラムの書換えなどという苦労もいとわないユーザが出てくるでしょう．やがてそれが全員に伝染して，並列計算機を3日使ったらやめられないという時代に早くなっていただきたいと思います.

4.5 新しい酒は新しい革袋へ

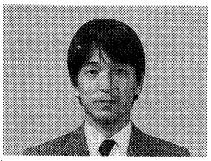
そのためにはやはり利用環境が問題であります．先進的ユーザというのはプログラムの書換えなどいとわないわけで，そういう人が最初に使うんでしょうけれども，やはりそれを普通のユーザに使っていただけるようにするために，つまりプログラムをいじりたがらないし，自分で書いたこともないユーザですね．ただ，自動並列化コンパイラがどこまでいけるかということは問題ですが.

私自身の主義は「新しい酒は新しい革袋へ」ということです．つまり一度逐次的なプログラムを書いて，それを並列化するのは遠まわりではないかということでございます．これは歴史的な問題がありましてそのとおりいくかどうか知りませんが，基本的には新しいものは新しい戦略で攻略するというのが基本的な方法ではないかと思います.

ですから結論というほどではありませんが，まず何よりも重大なことは，コンピュータの利用そのものを各ユーザレベルで自分の問題として見直そう．決して必要悪というような感じではなくて必要善であるというような雰囲気を世の中にもたらすことが必要ではないかと思います.

この根はかなり深いんでございまして，昔，ある化学の大御所の先生は弟子にコンピュータを使わせなかった．化学ですよ．私自身も物理の大学院生だったとき，先生などから「一流の物理屋は計算機を使わない」などと揶揄されたものでございます．物理や化学のようなコンピュータなしではやっていけないような分野でもこのような状態がつい二，三十年前にあるわけですから，あとは推して知るべしでございまして，我が国の学術全体がそういうような後進性を脱却して新しい境地に達しなければ，並列計算機はなかなか普及しないんではないかと思っております.

5. 並列計算機に対する不満と希望



吉岡　東京大学大型計算機センターの吉岡です．始めにざっと自己紹介をさせていただきたいと思います．私は，もともと破壊力学など機械関係の研究をしておりました．その研究の過程で，有限要素法を用いた構造解析をする必要に迫られたわけですが，その際既存の汎用有限要素法コードの機能，使い勝手の悪さに不満を抱きました．「こんなもの使っていられるか」ということで，自分でなんとかしようと思ったわけですが，ただコードを作るのでは面白くないので，有限要素法の並列化に関する研究を始めました．それが，だいたい今から3年くらい前のことで，私は大学院生でした.

5.1 高すぎて買ってくれとも頼めない

そろそろ CM，nCube などの初代のものが出始めていたころですが，正直言いましてとてもそんな高価なものを先生に買ってくださいなどと頼めない状況でした．しかし，どうにかして並列計算機を使わないことには研究ができませんので，周りを見回してなんとかならないのかと考えてみたのが，ワークステーションのネットワークです．通信が遅いなど，いろいろと問題があるけれども，考えようによっては並列計算機とみなせなくはないということで，いろいろとプログラムを書きまして，自前の並列有限要素解析システムを作ってみました．これが，思いのほか良い性能が得られまして，喜んでいるところに偶然アメリカのクレイ社のスーパコンピュータを無料で使わせていただける機会を得ました．「WS でできるのだから，スーパコンピュータでも同じはずだ」などと調子にのりまして，スーパコンピュータを何台かつないで並列に解析させる，ということをしておりました.

次に私のスタンスですが，こちらにいらっしゃる多くの方々とは異なりまして，数値解析を中心としたコンピュータ・ユーザです．ただ，先ほど小柳先生からご指摘がありました「保守的なユーザ」というのとはちょっと異なりまして，プログラミングにも興味をもち，つまり「プログラムを書き換えるのは嫌いじゃない．使いにくいプログ

ラムを我慢して使うよりは自分で作ってしまおう」というスタンスです.

次にきょうの発表内容ですが,正直言いましてとても「発表」などと呼べる内容ではありませんで,並列計算機を作っていらっしゃる方に対する嘆願という感じで,とにかくお願いにきました.

きょう,ここにパネラとしていらっしゃった方方は,大学の方も企業の方ももう長いこと研究をなされてきた方々ばかりですが,私は1年前には大学院生だった,今でも助手という,研究者の中で最も下っ端にいる人間です(笑).そういう下っ端からみますと,今の並列計算機は,上司に「買ってください」と,とても言えるものではありません.そういうわけで,「買ってください」と言えるようなものを早く作ってください,ということをお願いしたいと思います.

5.2 ユーザの指向,利用形態

ただ,とにかく「速くしろ」とか「安くしろ」とか言うのでは中身がないですので,まずコンピュータの使われ方について考えてみました.大きく分けて,その用途として,いわゆる研究開発に使われるもの,他方業務に使うものに分類できると思います.必ずしもはっきりと分けることができるわけではありませんが.

同じようにユーザ数について考えてみますと次のように分類できると思います.まず,非常に多人数で使う場合,大学の計算機センタなどがその最たるものでしょう.一方,1人ないしごく小人数で使う場合が考えられます.また,ユーザの指向について考えますと,特定の分野に限って使われる場合と多分野/多指向で使われる場合があります.多くの場合,1人あるいは少人数で使えば当然指向も特定分野になり,多人数で使えば多分野になるでしょう.

次に,ユーザの利用形態を考えますと,まず,A.アプリケーションを利用する人,それから,B.アプリケーションを開発して自分でそれを使う人,C.アプリケーションの開発のみを行う人に分類できると思います.

続いて,どのようにコンピュータが使われているかを考えた場合,たとえば計算機センタのようなところでは,用途は研究開発,ユーザは多人数,指向は多分野,利用形態はA,B,C全てを含んでいる,ということになります.一方,企業の場では業務として使われていることが多く,その場合,ユーザ数の多少には関係なく,指向は特定分野,利用形態はアプリケーションの実行だけ,という場合もかなり多いと聞いております.

5.3 ユーザの要求

次に,ユーザ側からみて,先ほどのA,B,Cのタイプのユーザが,コンピュータに対してどういう要求をもっているかを考えてみたいと思います.

まず,Aのアプリケーションを使用するだけのユーザですが,これはさらに2種類に分類することができると思います.一つは,市販のアプリケーションをバイナリで購入して利用するユーザです.このタイプのユーザは基本的にアプリケーションが高速に実行されさえすれば,使うコンピュータがどんなアキテクチャであろうと関係ありません.したがって,自分の使用するアプリケーションが動作することが,そのコンピュータを使うかどうかの基準となります.次に,アプリケーションをソースで保有しているユーザです.この中の多くが,先ほど小柳先生がお話しされた「困った存在」になるわけですが,このタイプのユーザの要求も実は単純で,そのソースを変更せずにコンパイルできて動作すればOK,ただしソースの手直しは絶対に不可,ということになります.

2番目に,Bのアプリケーションを開発し,自ら使用するユーザがいます.これらの方の多くが,先ほど小柳先生がおっしゃっていたように,「必要悪としてコンピュータを使う」人たちです.つまり,何か得たいものがあるのだが,それを求めるには計算機を使わなければならないので,仕方なくプログラムを作り,それを走らせて,結果を入手する,という場合です.このタイプのユーザの要求は,短時間に,そしてなるべく手間をかけずに結果を入手することです.したがって,ソースの変更はなるべくならやりたくないが,ほんの少しの変更で非常に高速化できるのだったらやってもいい,ということになります.先ほどのベクトル化の話が,まさにこの場合に相当すると思います.

ただし,このタイプのユーザは,将来もっと高速なコンピュータができればそちらに乗り換えますので,ある会社のある特定の機能などは使いたくありません.したがって,アーキテクチャは現

在あるいは近い将来に主流となるものがよく，また高速性を引き出すのにベンダ依存なプログラムを書くのは困る，というのが，要求事項になると思います．

また，多くの場合「1回走らせればいい」というわけにはいかないので，開発環境も問題となっていきます．特に Compile & Run が短い周期でできることが要求されます．

5.4 アプリケーションを開発するユーザは？

3番目に，おそらくこういう人はごく少数だと思いますが，アプリケーションの開発のみを行うユーザについて考えたいと思います．このタイプのユーザは，高性能なアプリケーションを開発することが目的ですから，とにかく高速なコンピュータであれば文句は言わないでしょう．ソースコードの変更は，たとえそれが大規模であっても高速性が得られるならば良しとします．ただし，Bのアプリケーションを開発して自分で使う場合と同様に，将来より高速なコンピュータが実現すれば，そちらに乗り換えますので，極端にベンダ依存な部分は使いたくない，ということになります．

もっと具体的に言いますと，今までなかったようなまったく新しい言語での書き直しなどでも，それを書き直すことによって非常に高速化できる，そして，その言語が今後いろいろな計算機上で動作するならば，書き換えもします．しかし，ある1ベンダのコンピュータでしか動作しない言語に書き直すようなことはしたくない，ということです．

アーキテクチャ，開発環境についての要求は，だいたいBのユーザと同一です．しかし，Bのユーザよりもいっそうデバッグに労力を割くことになりますので，デバッグ環境がしっかりしていることが必須だと思います．

5.5 普及するかどうかはコスト次第

最後に，本日の議題でもある，普及するか/しないかの問題ですが，これはコストとの兼ね合いだろうと考えます．現在の並列計算機の価格は大変高価ですから，大きな組織でなければ購入できません．そうしますと結果的に，多人数で使用することになり，多くの場合それは多分野/多指向のユーザを相手にすることを意味し，A，B，Cの全てのユーザの要求を満たさなければ普及しないと思います．この全ての要求を満たす，というのは，正直言いまして大変困難なことなのではないかと想像しています．むしろ，特定分野向けのほうが実現しやすいのではないかと思います．この場合，AならAだけを，あるいは，CならCだけを満足していると想定します．そうすると，特定分野/特定指向で使うことになりますが，これは多くの場合1人ないし少人数で使うことを意味すると思います．その場合には，現在のような価格ではとうていだめで，1人ないし数人の予算で買える価格にして欲しい，というのがお願いです．

あまりベンダの方にばかり要求するのも悪いので，少しはユーザ側でできることを述べたいと思います．先ほどユーザのタイプにA，B，Cの3種類の分類を導入しました．計算機が専門ではないが計算機を使う研究者と呼ばれる人たちの多くは，このBに分類されると思います．最近では，多少Aのタイプの方も増えてきたような気がしますが．そして，このBの方の要求が一番難しいのではないかと思います．工業製品，たとえば自動車を考えてみますと，昔は一人の設計者が全部作ったと思いますが，今では多くの設計者が分業して作っていると思います．そのことを考えると，このBのタイプの研究者も一人で何もかもやろうとしないで，そろそろ分業しましょう．「コンピュータなんか嫌いだ，触る時間はなるべく短くしたい」という方はAに移っていただいて，アプリケーションを使う側になる，そのかわり，アプリケーションに対する不満をどんどん出してもらいます．一方，「プログラムを作るのも結構好き」という方はCに移って，自分の分野のアプリケーションを精力的に開発する，というのが，良いのではないかと思います．ここにいらっしゃる方々にはあまり関係ないと思いますので，会社の同僚の方，大学の友人などアプリケーション分野で活動している方で，Bのタイプの方がいましたら，「Bみたいにどっちつかずになるのではなくて，AかCに徹したほうがいいよ」とお奨めいただければ幸いです．

これで終わります．

6. 並列化コンパイラの現状と将来

司会　どうもありがとうございます．以上で悪玉のほうは終わりまして，次に善玉のほうにいきまして，まずトップバッタは早稲田大学の笠原先生お願いします．

笠原　早稲田大学の笠原でございます．私はソフトウェア，特に並列化コンパイラの観点から並列計算機の実用化・商用化を逡巡させる要因は何かということについて考えてみたいと思います．現在の並列計算機，これは特にマルチプロセッサが中心なんですが，それを使用するときにどういう方式をとっているかということを考えてみたいと思います．

6.1 現在の並列計算機の利用形態

まず1番目の方式は，先ほどからユーザの方のお話がございましたけれども，逐次型言語でプログラミングをする，あるいは既存の逐次型プログラムをそのまま自動並列化コンパイラを用いて並列化するというアプローチであります．現在，この自動並列化コンパイラでやれることはループレベルの並列化，特に Doall あるいは Doacross を用いた並列化が行われています．

2番目の使用法は，並列性記述機能を付加した拡張言語でプログラミングし，自動並列化コンパイラを用いて並列化するというアプローチです．このときにユーザが指定する並列性というのはループレベルの並列性，すなわちベクトル処理とか，先ほどの Doall などとなります．また，それ以外の並列性としてはサブルーチン間の並列性の指定が可能になります．

3番目のアプローチは，並列化言語を用いまして並列処理の単位あるいはタスク間の並列性，メッセージ通信，タスクのプロセッサへの割当てなどを記述しまして，比較的データ依存解析，プログラムのリストラクチャリングなどの機能の低いコンパイラで並列化を行うという方式であります．

これらの三つのアプローチに関しましてそれぞれ問題点などを考えてみたいと思います．

6.2 逐次プログラムの自動並列化

まず一番目の逐次型プログラムの自動並列化なんですが，このアプローチが好きなのはプログラムの並列化にはまったく興味がない，あるいは並列化の知識経験がないユーザです．実際にはこのようなユーザが大半を占めているというのが現実だと思います．現在の市販のコンパイラはそのようなユーザの要求を満たすだけの機能をもっていない．これが現在並列計算機の実用化・商用化を遅らせている最大の原因ではないかと思います(**図-11**)．

なぜ現在の自動並列化コンパイラの能力が低いかといいますと，ループ並列化には限界があるということだと思います．これはアムダールの法則ですけれども，100% のシーケンシャル処理時間のうち 90% が Doall 処理できるプログラムがあるとします．90% の部分が並列処理できるということは非常に並列化しやすいプログラムだと思いますが，これを 1,000 台のプロセッサで Doall 処理しますと 90% の部分の処理時間が 1/1000 になって，結果として 1,000 台使って 10 倍の処理時間が得られる．これにより Doall による並列化ではどんなにうまくやっても限界があることが分かります．このように実効性能が実際には高くならないところにユーザが並列処理計算機を使いたくない一つの原因があるのだと思います．

Doall 処理だけでは限界があることが分かり，今後何をしなければいけないかといいますと，現在シーケンシャルループとか，ループ以外の部分のコンパイラを用いた並列化だと思います．

6.3 拡張言語を使ったユーザによる並列化

次に2番目の拡張言語を用いたユーザによる並列化なんですが，ここでいくつかの問題点があります．まず，一つは本当の意味で高い並列性をひきだしているのはごく一部の優秀なユーザだけで

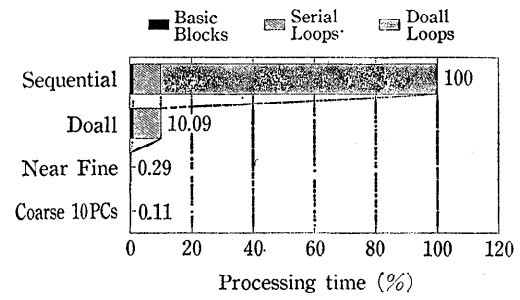


図-11　マルチプロセッサにおける粗粒度/細粒度並列処理によるスピードアップ

ある．それらのユーザを対象にマシンを作っても売れる台数に限りがある．だからあまりメーカのほうも力を入れない．1万台，100万台規模のプロセッサをすべて動かせるようなプログラムを書けるかどうかということにも疑問があります．もちろんループだけの並列性でしたら記述できると思うのですが，ループ以外の並列性を記述できるかどうかは疑問です．

これはすべて自動並列化コンパイラがうまく並列化していけばいいんですが，自動並列化コンパイラだけでは十分な並列化が行えない場合には，チューニングツールなどを用いてユーザが簡単に並列化できるようにしていかなければいけないと思います．

6.4 並列言語

3番目の並列言語を用いたユーザによる並列化なんですが，このアプローチを望んでいるのは並列処理研究者，あるいは本当に並列処理が好きなユーザだけだと思います．人口的にはきわめて少ないと思います．

このアプローチでどういうことが問題かといいますと，先ほどからプログラムの書換えの話が何回かありましたが既存のプログラムを並列言語を用いて書き換えるということは，ユーザにとっては無駄な時間を費すことになる．すなわち，大多数のユーザはこういうことはあまりやりたくないというのが実状です．

早期に並列計算機を実用化・商用化するためには，やはり先ほどから何回かお話が出ましたが，自動並列化コンパイラの高機能化が重要ではないかと思います．

6.5 粗粒度，細粒度の両面からの並列化

先ほど申しましたように，従来のループ並列化に加えてこれからやっていかなければいけないのは，粗粒度タスクの自動並列化，細粒度タスクの並列処理データの分割，配置，データ転送の最適化などでこの辺の技術をしっかり開発していく必要があると思います．

6.6 粗粒度タスクの並列処理

まず粗粒度タスクの並列処理に関してですが，現在の自動並列化コンパイラではサブルーチンとかループの間の粗粒度の並列性がほとんど検出できていません．先ほどのようにユーザが指定することはできるわけですが，並列処理をしたときのオーバヘッドが大きいためうまくいかないという問題点があります．この二つの問題を解決することが必要となります．

まず，1番目の粗粒度タスク間の並列性抽出のために，まずやらなければいけないのは使用領域解析とかインタプロシージャ解析などを含めたデータ依存解析の高度化だと思います．

またもう一つは，制御依存とデータ依存を考慮してマクロタスク間の並列性を引き出す最早実行開始条件解析が重要となります．これはすでに研究所レベルでは使用されていますので，これを早く実用化していくことが重要だと思います．アメリカではこの技術を実用化しようとするプロジェクトがすでにありますが，日本のメーカの方々はまだやっていないと思いますのでこの辺に対する研究・開発を早くやっていただきたいと思います．

6.7 細粒度タスクの並列化

次に低オーバヘッドの粗粒度タスクスケジューリングに関しましては，従来のようにOSコードを利用すると数百，数千クロック程度の大オーバヘッドが生じてしまいますので，これを避けるためにコンパイラが対象プログラム用のダイナミックなスケジューリングルーチンを自動生成し，マクロタスク間のスケジューリングを低オーバヘッドで可能にするべきと思います．

粗粒度タスクの並列処理，それと従来のループ並列化がうまくいったあとさらに並列性を上げよう，実効性能を上げようとすると，残ったシーケンシャルループとか，ループ以外の基本ブロック内の並列性をどうにかして引き出さなければなりません．このような並列性を使用するためには細粒度の並列処理が必要となりますが，プロセッサ間で命令レベルの細粒度の並列処理を行うことはほぼ不可能ですから，たとえばステートメントレベルでの並列性利用を考えるべきと考えます．

このステートメントレベルの並列処理に関しましては，すでにコンパイラ技術としては開発されています．あとはアーキテクチャサポートを前提としてそのような並列処理を実現できるようなマルチプロセッサシステムをつくって，実際に製品として生かしていくということだと思います．

6.8 マルチプロセッサのアーキテクチャは？

今後のマルチプロセッサのアーキテクチャは，従来の主記憶共有型からローカルメモリ＋集中共

有メモリ型，ローカルメモリ＋分散共有メモリ＋集中共有メモリ型，ローカルメモリ＋分散共有メモリ型＋分散メモリ型へ，すなわちコンパイラの開発が難しくなる方向に移行してくると思います．そのときに最も大事なのは，ローカルメモリをいかに有効に使うかということです．ローカルメモリ有効利用ということになりますと，データをいかに分割して配置するか，これがきわめて問題だと思っております．

このデータ分割配置の研究が現在世界中で注目されており，活発な研究が行われていますが，現在の段階ではコンパイラの実現が難しいということで，ツールをつくりましょうとか，言語をつくりましょうという方向に力が向けられています．

この自動データ分割配置を可能とするコンパイラを開発するためにやらなければいけない研究はグローバルなデータ依存解析，あるいは先ほど申しましたマクロタスク間の最早実行開始条件解析だろうと思います．

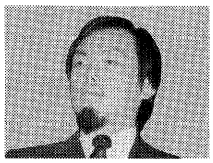
また，ローカルメモリを使ったときに，いかにうまくデータを分割・配置をしてもどうしても残ってしまうデータ転送が存在する．このデータ転送を隠す方法としては処理とデータベースのオーバラップが考えられますが，このようなオーバラップを積極的に使用できるスケジューリング手法の開発が今後の重要な課題ではないかと考えます．

6.9 もっと並列化コンパイラの研究を

最後にまとめといたしまして，私の意見としましては並列処理計算機の早期商用化，普及を目指すためにはユーザの望む方法で利用できる並列計算機を開発すべきである．現在のユーザが望んでいるのは，並列処理を意識せずに簡単に使えるマシンであると思います．またこのような計算機を実現するためには自動並列化コンパイラが最も重要であると思われます．したがいまして，従来多くの研究者の方がハードウェアとかアーキテクチャを中心に研究されていたと思いますが，もう少し並列化コンパイラの研究にも力を入れて，先ほど島崎先生がいわれましたように，この分野の論文を多く出していただきたいと思います．

以上でございます．

7. ICOT の並列アプリケーションのエッセンス



瀧　ICOT の瀧でございます．

アプリケーションの話をしなさいということですが，きょうは時間の関係であまり詳しい話はできません．第5世代コンピュータプロジェクトを 10 年もやって，500 億円の金を使っていろんなことをやったわけですけれども，その中のアプリケーションに近いところでエッセンスといえるようなところをお話ししたいと思います．

7.1 並列処理の性質

これはかなり独断ですが，並列処理の計算の性質のカテゴリ分けをやってみました（**図-12**）．まずデータストラクチャがスタティックかダイナミックか．データが均質か不均質か．計算が同期的か非同期的か．これがみんなそろっているのが狭い意味でのデータ並列で，スタティックでユニフォームでシンクロナスという3拍子そろった性質があります．この種の計算は，データ分割によって効率のよい並列処理が可能な扱いやすいタイプといえますが，下のほうにいくに従ってこの3拍子そろった性質がなくなってきます．

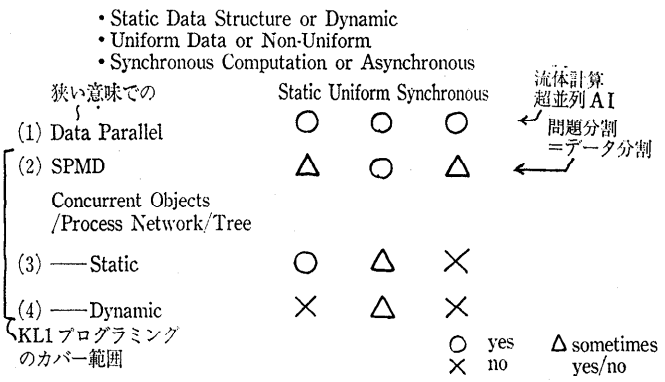


図-12　並列処理の分類

7.2 ダイナミックでノンユニフォームな並列計算とは

ICOT でやってきた並列処理というのは，主にスタティックではなく，ユニフォームではなく，シンクロナスではなくという範囲がカバーできるようなそういう並列処理ということだったと思っております．

簡単にいいますと，ダイナミックでノンユニフォームな並列計算ということができます．たとえば探索木の展開みたいなものは，それぞれの評価関数，評価値によってどの枝を伸ばしていくかというのがダイナミックに変化してたいへん動的な性質をもっています（**図-13**）．またコンカレント・オブジェクトモデルにもとづくプログラムでは，個々のオブジェクトが非同期的に異なる計算を行い，計算の性質が非常にノンユニフォームだといえます．

そのようなところを ICOT で相手にしてきました．これをまとめますと，従来あまり手がつけられていなかった並列のアプリケーションとして，非常に大規模な計算なんだけれども，動的で非均質的な性質をもったそういう計算，いいかえるとデータ並列的でない計算，そういう領域の計算に対して効率のいいシステムというものをかなり実現できたんじゃないかと考えています．対象としては知識処理なんかが，その代表的なものです．知識を並列計算機上にマッピングしたときに，入力によってどの知識が利用されるか分からない．非常にダイナミックな，ノンユニフォームな計算を発生します．ほかに知識処理に限らなくても，記号処理計算でそういうものがたくさんありそうです．それから数値計算でも必ずしもデータ並列的でないものもあるでしょう．こういうふうなところは，いってみれば非常にやさしくない問題ということができます．それは，強いシーズ指向の国家プロジェクトだから，といえるかもしれません．

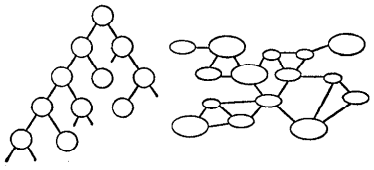


図-13 ダイナミックでノンユニフォームな計算

7.3 ICOT のアプリケーションのいろいろ

こういう難しいところからやって，そんないい結果が出るのかというお疑いもあるでしょうし，世界中探してもこれまで出ていなかったところがほとんどなわけです．そこで，ICOT でやってみた結果を簡単にまとめさせてもらいます．つい先ほど，6月1日から5日間国際会議がありまして，そのときデモンストレーションでもやりました．結構な数の応用プログラムが並列処理マシン，PIM とか，その1世代前のマシンのマルチ PSI とか，プロセッサ数でいうと最大で256ぐらいですけれども，この上で**図-14** のようなアプリケーションがいろいろ動きました．

7.4 論理シミュレーション

この中の主だったものを簡単にいくつか紹介させていただきます．

これは論理シミュレーション（**図-15**）．やったのはイベントシミュレーションに基づくタイプでタイムワープという方法を使います．詳細は省略させていただきますが，計算のモデルは，基本は並列オブジェクトモデルなんですけれども，実現したのはほぼ SPMD です．シングルプログラム・マルチプルデータで，各プロセッサにシミュレーションエンジンを置き，ゲートデータを分割配置します．ただし，ここで「ほぼ」とついているのは，SPMD 的でない複雑な処理も含まれるからです．タイムワープメカニズムの中ではどこか一つのプロセッサでの集中管理を必要とし，通信や同期の構造も単純ではなくなります．

SPMD 的なんだけど，細部をみると，より複雑なプログラムになっているわけで，通信や同期の複雑な部分の記述に，KL1 言語の利点が生かされています．性能は256プロセッサで1プロセッサの166倍です（**図-16**）．

- Major parallel application systems (18 systems)
  - —VLSI CAD (7 systems)
  - —Diagnostic expert systems (2 systems)
  - —Natural language processing systems (3 systems)
  - —Biological analysis (2 systems)
  - —Biological data base application for Kappa
  - —Parallel software generation system, MENDELS ZONE
  - —Go-game playing system
  - —Legal reasoning system, HELIC-II

図-14 ICOT で開発された並列アプリケーション

イベント・シミュレーション＋タイムワープ
基本は並列オブジェクトモデル

⇩

実現は　ほぼSPMD，静的データ割付

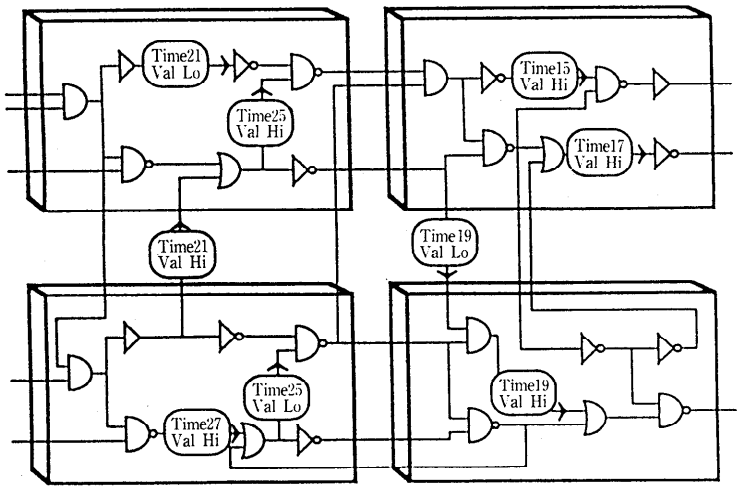


図-15　並列論理シミュレーション

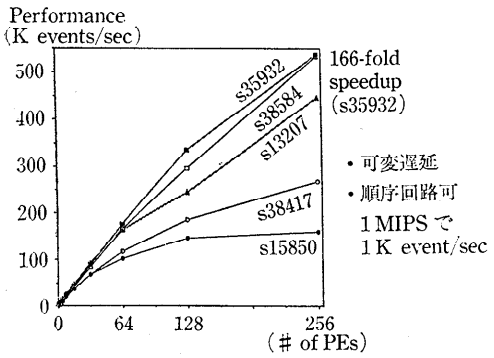


図-16　並列論理シミュレータの台数効果

● シミュレーション性能は，500 K イベント/sec です．この種のソフトウェアシミュレータは 1 MIPS に対して 1 K イベント/sec といわれており，試作システムは KL1 という論理型言語で記述しているにもかかわらず，十分高い性能を実現したといえます．

7.5　並列配線プログラム

次は配線です(**図-17**)．予測線分探索法という，保障した線分探索アルゴリズムを基本にしています．それを並列オブジェクトモデルにもとづいてプログラム化しました．それぞれの配線セグメントがすべてオブジェクトとして実現され，配線セグメント同士が通信し合って，短い配線経路を決めてゆきます．

予測線分探索法はまったくの分散アルゴリズムに設計しなおしています．

静的な初期割付と，オブジェクトが動的に生成消滅をくり返すという特徴をもっています．かなり不均質な計算を発生します．この場合は 256 プロセッサを用いて，1 プロセッサの性能のだいたい 92 倍 (**図-18**)．問題が小さいと少し性

先読線分探索法
並列オブジェクトモデル　静的初期割付
分散アルゴリズム　　オブジェクトの動的生成消滅

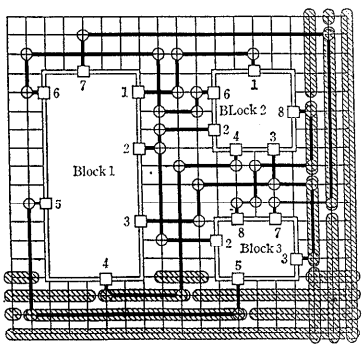


図-17　並列配線プログラム

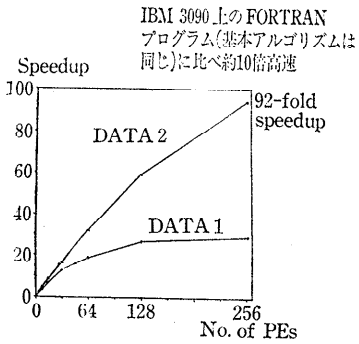


図-18 並列配線プログラムの台数効果

能が頭打ちになります．幸いにして，基本的に同じアルゴリズムを FORTRAN で記述したものが先に NTT でつくられていて，それを IBM 3090 で実行した場合とわれわれのシステムの絶対性能を比較したところ，IBM に比べて約 10 倍高速でした．

7.6 遺伝子解析，法律家のエキスパートシステム，定理証明

あとまだまだあるわけですが，次は遺伝子を構成するアミノ酸の類似性解析（アラインメント）です（図-19）．これに関しては 256 プロセッサで 223 倍の性能向上を実現しました．アルゴリズムは，ダイナミックプログラミングの手法を用いたタンパク質のアミノ酸配列の最良マッチングを用いています．

その下は法律家のエキスパートシステムで，これまで判決で出てきた事例というものをケースベースとしてもって，一方法律の条文をルールベースとしてもっています．ケースベースとルールベースを組み合わせて，新しい事件に対してそれは有罪になるのか無罪になるのかというあらゆる解釈の可能性をすべて計算するというようなシステムをつくりました．

これは 64 プロセッサで 56 倍ぐらいでてますがこのパラダイムは，ケースベースのほうは問題分割，SPMD です．ケースルールを分割してプロセッサに割り付けます．

ルールベースのほうはダイナミックな負荷分散です．実行時に探索しなければいけないことを割り付ける動的負荷分散を使用しました．

最後は MGTP です．これは定理証明機です．もともとはプログラム合成に使うことを意図したものですが，いまはこれを一種の推論エンジンとしていろいろな用途に使っています．たとえば前の法律エキスパートの中ではルールベースの検索に使っています．この定理証明機は 256 プロセッサで，230 倍の性能向上を示しています．このパラダイムは基本的にはジェネレートアンドテストですが，一部データベースをもっており，動的な負荷分散も行っています．

ということで，これまでお話しした応用はだいたいどれもが非常に動的な性質をもっている，あるいは計算の均質さが低い問題で，従来皆さんが難しそうだからやめておこう，もうちょっと見込みのあるところからやりましょうというものでした．そのような応用について，結構な性能がでてきたわけです．

Program	Speedup/procsssors
Logic Simulator	166/256
Routing	92/256
Alignment	223/256
CBR of Legal Reasoning	56/64
MGTP (Theorem Prover)	230/256

図-19 各種アプリケーションでのスピードアップ

7.7 KL 1 の 力

まとめですが，動的で処理の均質さの低い問題でも結構高い並列処理効率が得られるようになってきた，正確にいうと効率を上げられるものが出始めたという段階かもしれませんが，そういうことになってきました．なぜそういうことができるようになってきたかというと，KL 1 という言語は，この動的で不均質な問題というものを扱うためのだいたい 5 つぐらいの特徴のある機能をもっています．このために，従来難しかったプログラムの記述が楽になってきたこと．合わせて効率のよい実効環境，デバッグと性能解析ツール，この辺が充実したために，実用レベルのプログラム作りが可能になってきたことがあげられます．

同時に，多様な並列プログラミングのパラダイムを蓄積するために，われわれシーズ指向の種まき屋さんが努力して，アプリケーションの人をむりやり引き込んできて，応用システムを作りながらその中で工夫を重ねてきたことがあげられます．並列処理のためのプログラミングパラダイムは，蓄積され始めているわけですが，これはまだまだ継続してやっていかなければいけないでしょう．

結局こういう言語実効環境，こういう複合パラダイムを支えられるようなシステムがあって，それに多様なパラダイムの開発，蓄積がそこに刺激を与えて，それで並列応用の開発が加速されて，それによって本当に魅力的な性能というのがある分野で実現できたら，これは本当のマーケットをつくっていくというところにくるのでしょう，それがシステム作りにフィードバックされて，システムの改良が行われると，並列処理が順調に発展してゆく良い循環が生まれることになるのでしょう．ここまで達せられればいいわけですけれども，恐らくいまは，いわゆるテスト的なシステムができて，それからこの並列応用の開発の事始めぐらいのところができてきたというふうに思います．

きわめてシーズ指向のアプローチなので，マーケットの拡大まで突き抜けるまでには，種まき屋さんはまだ努力を続けなければいけない時期かなと思います．

司会　どうもありがとうございました．山田さんお願いします．

8. コネクションマシンの話

8.1 超並列コンピュータの市場

山田　われわれはメーカですので超並列コンピュータの市場のサイズをみてみますと，いま1992年ということで，私どもいま今年は全世界で400億円の超並列コンピュータが売れるというふうにみております．私どもことしの目標が200億円足らずですのでだいたい半分近いシェアと思っています．日本ではこういう最先端のコンピュータはアメリカから普通2年おくれです．ですからことしは日本では25億円ぐらいの市場であるとわれわれは思っています．実際販売していてトータルしますとだいたい25億円ぐらいになります．この一，二年の間にアメリカの市場に追いつくだろうと．そうしますとだいたい世界の市場の20%ぐらいが日本になるというふうに私どもは思っています．

8.2 データ並列処理

ダニエル・ヒリスはコネクション・マシンをつくってどういうふうにアプリケーション処理をしようと思ったかと言いますと，それはデータ並列処理ということです．いまの瀧さんの話にもありましたけれども，データ並列処理，つまりデータというのはこういう流体現象を解析する場合でも，もしこんなメッシュがありましたらノードならノード，エレメントならエレメントに対する計算式は全部一緒です．100万点の節点があっても，それに対するアルゴリズムはたとえばナビア・ストークだとか，またはフリーラグランジなど同じ計算式をすべての点に対して計算を繰り返し繰り返し当てはめていく．

逆にいいますと，100万点の節点がありましたら，その100万点に100万分の仮想プロセッサを全部割り付けまして，全員に同時に解析のための式を当てはめてやって同じ計算をすれば，一挙に処理が済んじゃう．

そういうアイデアです．それをデータ並列処理というふうに呼びまして，このデータ並列処理方式に一番適したハードウェアをつくるということでつくったのがコネクションマシン，CM-2という機械でございます．

8.3 CM-2のアーキテクチャ

CM-2という機械は大ざっぱに次のような感じになっています．つまりプロセッサとメモリがペアになっていて，それがずらっと，このCM-2の場合は6万4,000個ほどのペアが並んでいる．先ほどのようなデータをフロートエンド経由でメモリに移行する．またはディスク・アレイのデータを下から移行するということで，データをとにかくいっぱい詰めます．そして上からフロントエンドでもってコンパイルしたオブジェクトコードをシーケンサというプロセッサの親分みたいなところにどんどん転送します．そうしますとシーケンサは全プロセッサに対して同じ命令をロードします．ですからこれはハード的にも典型的にSIMD型の機械なんです．

命令は全プロセッサが同じものを実行する，もちろんデータはおのおのちがうけれども，ということでこういう方式を作った．

8.4 CM-2の使用例

この機械を使った最近のCFDの例ですね，FEM構造解析に使用したCMSSL．これは私どもの科学計算用ライブラリを使った解析例で，レポートがスタンフォード大から出ています．このケースでは，32K個のCM-2で，4面体要素を87万8,544個，それからトライアングルが1万

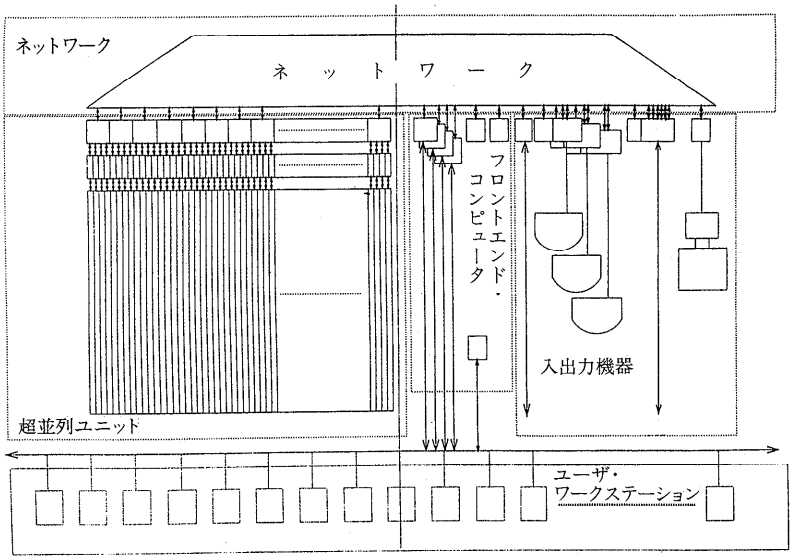


図-20　CM-5 の構成

5,000 個ぐらいです．合計 90 万個ぐらいのエレメントに対してだいたい計算時間が 40 分，ギャザリングが 20 分ぐらい，スキャタリングがだいたい 20 分ぐらいということで，だいたい一，二時間で 90 万要素ぐらいの 3 次元の FEM 処理ができるようになったというレポートです．これは純粋に SIMD 方式で処理したものです．

同じように，三十数万の四面体を使った，これは地球規模の大気解析をした例が最近出てきております．これも純粋に SIMD 処理をしたものです．

8.5　CM-5 の登場

私ども去年の 11 月に CM-2 の上の機械を発表しまして，CM-5 と呼んでいますけれども，CM-5 はこういう構成です（**図-20**）．上のほうにありますのは超並列ユニットで，一番上に SUN のスパーク・チップが 1 個ずつ入っています．スパーク・チップの下に 4 つのベクトルユニット．一つ一つの下にメモリがありまして，一つの短冊が 8 メガバイトのメモリです．周辺機器類がこちら，そして真ん中のユニットがフロントエンド・コンピュータということでこれもいまは SUN のモデルです．

ということで，CM-5 のアーキテクチャは CM-2 のアーキテクチャと比べましてがらっと変わりました．ハードウェア側からみますと MIMD 型に完璧に変わったんです．ただ，処理形態はデータ並列ということで全然変えていません．

これは一つのプロセッサ，ノード・プロセッサの中身なんですけれども，RISC プロセッサとベクトル・プロセッサ二つありまして，下に 4-パイプのベクトルユニットが入っています（**図-21**）．

この CM-5 になりますと，CM-2 と違いまして，いくつかの処理領域がある場合，処理領域ごとに違うアルゴリズムを処理しなければいけない場合に，それぞれのプロセッサ群を処理領域に割り付ければいい．領域ごとに同時にデータ並列処

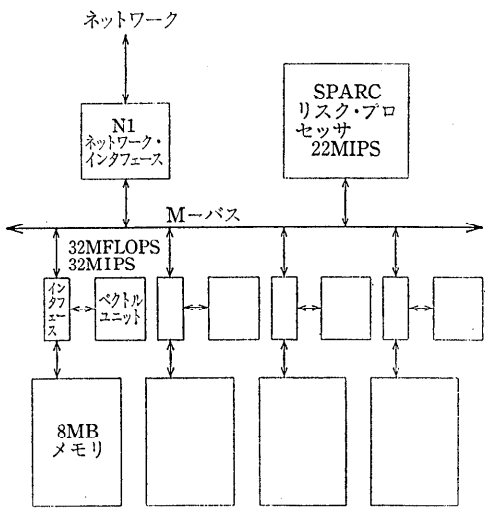


図-21　ノード・プロセッサ構成図

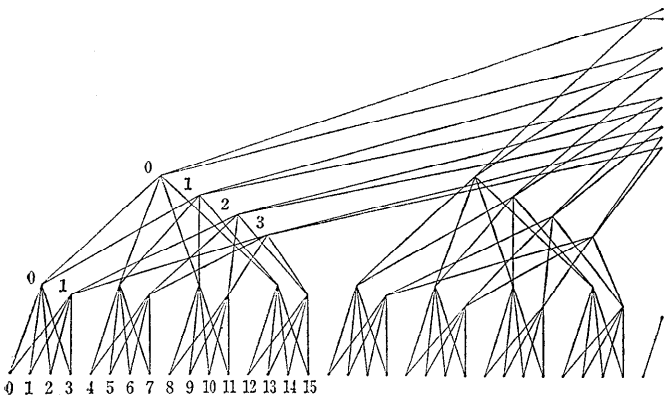


図-22 データ・ネットワーク

理を多重に実行します．もちろんそれほど単純じゃないけれども，データ並列処理を並列にといいますか，多重に実行することができます．ですからこの辺は MIMD の要素が採り入れられているんですね．

8.6 キーポイントはネットワーク

並列処理コンピュータというのは，ネットワークがキーポイントでございまして，ノード・プロセッサを数百，数千つなぎ合わせてネットワークして，そのネットワークの中にどういうファンクションをもたせるかということがキーポイントでございまして，ネットワークの中には通信ということだけではなくて，各プロセッサの結果を階層的に積み上げながら一つの論理演算結果になる．また大きな数値から順番に長いデータ連糸にしたり，または小さい順に並べたり，または合計をとったりということをフロントエンドから自由にコントロールができます（図-22）．

8.7 ユーザは同期を気にする必要がない

また逆に，コントロール・プロセッサから，ユーザプログラムのロード，ブロードキャスト，シンクロナイゼーション，コンテキストスイッチ，などをネットワークに指示したり，TSS 処理の途中結果をコンテキスト・スイッチしたり，パーティション間でデータのソケット転送なりしなければなりません．こういう制御を OS がやります．コンパイラはコンパイル時に必要に応じて同期命令を挿入します．ユーザがシンクロナイゼーションをすることは一切必要ありません．もちろん，いや自分が全部したいんだという人にはアセンブラレベルまで公開していますので，自分で同期まで考えれば全部それはできますけれども，ユーザからみた場合に超並列処理といわれる難しい部分を全部隠しちゃおうということです．同期はユーザが気にする必要はございません．コンパイラがやります．そういうやり方をとったわけです．

いろいろな処理の形式はあっても中心はデータ並列処理です．それからプロセッサ部分をつないでいるネットワークの中に，その同期，メカニズム，または集合演算のファンクション，または論理演算のファンクション，こういうものを折り込んであります．これは全部システムの責任です．

それからなるべく使いやすいように超並列用のライブラリを充実させてそれから望む人には何でも公開しますのでプロセッサを自分でもってすべてコントロールしたい人はどうぞ．

ということで，先ほどの地球規模の大気解析の例で最近 1024 ノードで処理したときの実行結果が入りまして……．

ちょっと数字は公表できませんですけれども，その結果数十ギガフロップスで処理がされている，ということです．

9. 並列計算機並列処理に対する教育

司会　どうもありがとうございました．最後になりましたが高橋先生よろしく．

高橋　本日は並列計算機の実用化を逡巡させる諸要因についてというテーマで，私の分担は教育ということでございますが，全体的にみますと，なぜ並列計算機が実用にならないかということはいろんなことが考えられるわけです．

計算機の研究開発は，高速化と記憶容量の拡大と使いやすさ，新しい応用，そういうことを目指して長い間一生懸命やってきたわけですが，高速化については，従来の方式では限界があるということで，超並列というのはブレークスルーがあるのではないかと期待されているわけです．メモリ容量については一応超 LSI メモリ，磁気記録，光ディスクの発展により，かなり見直しがあると

いえる．使いやすさ，新しい応用については劇的に効果があるものから，それほどでもないものまで千差万別である．しかし，使いやすさの追求はCPU power を相当必要とすることも分かってきたというのが現状です．

並列処理計算機が普及しない要因はメーカ側，ユーザ側，大学側のそれぞれにあるんですが，私は大学側の問題を述べてみたいと思います．

9.1 並列プログラムの教育

まず，大学側の問題としては，プログラミング教育の問題があります．先生は，並列プログラムというと，多くの場合学生に対していつもトイプログラムだけで話をする．特定の応用に対して適切なアルゴリズムの開発とそれに対する教育がおくれているというようなことがあるかと思います．だいたい実用規模のプログラムとトイプログラムの間には大きなバリアがあるわけです．このバリアを何回か飛び越えるようなことを考えないといけない．しかし，現実は，このバリアを埋めるようなことがあんまり行われていないんじゃないかというのが一つの問題点です．

それから，先ほど並列計算機を対象とした言語やそのコンパイラのフィードバックがおくれているという話がありましたが，ユーザの立場で考えてみますと確かに既存プログラムを自動的に超並列計算機に変換してほしいというユーザから，処理速度をあげるためには，少々の苦労はいとわないというユーザまで，いろいろな対応の仕方があるんですけれども，残念ながら日本では並列計算機自身が少ないもんですから，どうしても研究者の数というものも少ないというのが現状ではないかと思います．

9.2 十分でないプログラム開発環境

次に，並列処理を対象としたプログラムの開発環境，デバック環境の開発が十分でないということがあります．たとえば実行状態のモニタリングが可能であってほしい．システムの構成の面から考えてみると，ブラックボックスがなるべく少なくなるようなことを考えなければいけないんですね．ですからOSの部分を軽くしていき，極限はゼロにもっていきたい．OS なんか要らないというぐらいにもっていくぐらいのことを考えていきませんとうまくいかない．そのためには並列処理に必要なOSの機能に対するハードウェア支援の徹底とか，いまは低水準言語でやっているんですけれども，並列処理を前提とした高水準言語をつくる必要があります．比喩的にいうとOSの世界はMultics から始まって，その反省で UNIX をつくったんで，そろそろ並列用のOSとして，軽くてシンプルな PARAX ぐらいのものをつくらないと世の中に普及させるのは難しいんじゃないかと．

次に，多くの大学で並列計算機について，本格的な教育をしていないというのはたしかであります．OS の講義の中で，仕掛の話で終わってしまう．コンピュータサイエンスのカリキュラムでさえ並列処理についてはあまり突っ込んでいないのが現状であります．

9.3 並列計算機が使えない

最後に日本の場合，ほとんどの大学の情報工学科で学生たちが並列計算機を自由に使える環境にないという点です．国立大学の情報工学系学科の例では，教育用計算機について，買い取り方式からレンタル方式に 15 年かかってやっと切り換わってバッチ，TSS を経て，現在ワークステーションの分散ネットワークをつくってやれやれというところが大多数であります，現状は，これは今後，“ワークステーション＋並列計算機”という環境が普通にならなければと思います．アメリカではすでにそうなっていますけれども．私の学科でも現在官報にそういうシステムを公示しておりますので，ぜひ日本のメーカ，外国のメーカ問わず受注に応募していただければありがたいと思っております．

9.4 カリキュラムと並列処理の関係

この辺で普及しない理由をおしまいにして，次に，コンピュータサイエンスのカリキュラムと並列処理の関係について話を移しましょう．コンピュータサイエンスのカリキュラムは ACM の提案に基づくもので9つの副領域と各副領域が三つのパラダイム，すなわち，理論，抽象化，設計からできています（**図-23** 参照）．この 9×3 のマトリックスをつくってどういう講義をやったらいいかと決めております．

その9つの副領域でございますが，並列処理に関係する分野で特に大きいのは，アーキテクチャとOSというところであります．この内容にしても必ずしも十分ではないんですが，しかし ACM

	理論	抽象化	設計
1. アルゴリズムとデータ構造			
2. プログラミング言語			
3. アーキテクチャ			
4. 数値的および記号的計算			
5. オペレーティングシステム			
6. ソフトウェア方法論とソフトウェア工学			
7. データベースと情報検索			
8. 人工知能とロボティックス			
9. 人とコンピュータのコミュニケーション			

図-23 ACM 提案のコンピュータサイエンスの学問領域とパラダイム

モジュール番号	学期	入門科目系列でのテーマ	講義回数(週)
1	C1	アルゴリズムの基本概念	(6)
2	C1	計算機の構成	(5)
3	C1	数学的プログラミング	(3)
4	C2	データ構造とデータ抽象	(5)
5	C2	計算可能性の限界	(3)
6	C2	オペレーティングシステムと保全性	(3)
7	C2	分散処理とネットワーク	(3)
8	C3	人工知能のモデル	(5)
9	C3	ファイルとデータベースシステム	(3)
10	C3	並列計算	(3)
11	C3	ヒューマンインタフェース	(3)

図-24 入門科目系列（C1，C2，C3）で取りあげるテーマのモジュール分け

カリキュラム 88 を見ますと，入門科目の系列が例示されています．期間は 1.5 年，3 学期をかけて 9×3 のマトリックスの主なところを抽出して入門レベルで広く浅く全体をカバーしています．

ただし，ここでは一般的なプログラミング教育は既習であることを前提としております．この授業科目の内容を表す実験テーマだけをご紹介いたしますと，**図-24** のようにモジュール 1 とかモジュール 11 まであるんですけれども，モジュール 1 の「アルゴリズム」の基本概念から始まってモジュール 6 の「OS と保全性」とか，モジュール 7 の「分散処理とネットワーク」，モジュール 10 の「並列処理」，これをだいたい 3 週間から 6 週間の実験でやれというわけですから，そう大したことはできないだろうとご想像になるかもしれません．

たとえば，モジュール 1 アルゴリズムの基本概念を 6 週間でやるのですが，そのテーマは「どうしたら信頼できて効率のよいプログラムを書くことができるか」というものです．実験の課題として，①サンプルプログラムを編集し，実行し，計測してプログラムの振舞(プロファイル)を作成するとか，②プログラムを与えて，道具を使って機能のちょっと違うプログラムをつくるとか，③スペックを与えてプログラムをつくるとか，④複数のアルゴリズムの比較，計測とか，⑤再帰的プログラムと反復的プログラムとを相互変換させる，などというのを最初にやっていくわけです．

次にモジュール 2 の「計算機の構成」を見ますと，割当時間は 5 週間ですが，テーマは「アルゴリズムを実行する機械装置をどうやってつくるか．そうするのはなぜか」というものです．これはノイマンコンピュータのモデルを学ぶということが一つの目標でございます．

モジュール 6 の「OS と保全性」というのがあるんですが，割り当て時間が 3 週間ありまして，テーマは「コンピュータシステムの資源を管理するソフトウェアを設計し構築し，そのソフトウェアの保全性を保証するにはどうしたらいいか」というものです．ここでの実験課題は，既存の OS が提供するさまざまなサービスに触れさせること．すなわち，並列処理機能が計算を記述しやすくしたり，効率よくしたりする例を直接の経験を通じて理解させることにあります．この中には，協調動作をする複数のプロセスを使って，何か簡単な仕事をやってみる．そして，性能を逐次的解法と比較する，という課題が三つの中の一つとして含まれています．

さて，本パネルの主題であるモジュール 10 の「並列計算機」では 3 週間で，テーマは「複数のアクティビティが並列性に生じ得る計算機アーキテクチャでアルゴリズムを実行するとき，制御や効率に関してどのような問題が新たに生じるか」というものです．実験課題では，トイプログラムの代表である「哲学者の食事問題」が出てくるわけです．すなわち，①「哲学者の食事問題」の解がどう動くかを，さまざまな事象系列について追いかける，というものです．

次に今度は②コネクションマシンなど機械のシミュレータを使って実行してみるわけです．

9.5 日本のカリキュラムはどうか？

これが現実の入門コースの科目の内容です．**図-25** は日本のカリキュラム IPSJ CS カリキュ

↑	JCS 1	プログラミング序論
	JCS 2	プログラムの設計と実現
コア	JCS 3	計算機システム序論
	JCS 4	計算機ハードウェア基礎
	JCS 5	情報構造とアルゴリズム解析
	JCS 6	オペレーティングシステムとアーキテクチャI
↓	JCS 7	プログラミング言語の構造
	JCS 8	オペレーティングシステムとアーキテクチャII
	JCS 9	ファイルとデータベースシステム
	JCS 10	人工知能
	JCS 11	ヒューマンインタフェース
	JCS 12	計算のモデルとアルゴリズム
	JCS 13	ソフトウェアの設計と開発
	JCS 14	プログラミング言語の理論と実際
	JCS 15	数値計算の理論と実際

図-25 情報処理学会の提案になる理工系情報学科（情報工学科，情報科学科など）のカリキュラム（IPSJ CS カリキュラム J90）

ラム J90 です．日本の情報工学科というのは情報工学科という名前の中身は何だか分からない学科がたくさんあるという現状では，JCS 1 から JCS 15 まで全部を深くやれというのは無理があります．したがって，JCS 1 から JCS 7 までがコアであって，せめて，まずコアだけはやってほしいという段階です．並列処理をこの中に加えるほどの数熟度が残念ながらまだない．並列計算のキーワードはしたがって JCS 6 の中でデッドロック防止だとか，相互排除だとか，同期だとか，こういうような OS 絡みのことになっているのが現状です．

さて，私の学科の実例を申しますと，私の研究室では，OS/omicron という名前の OS を 10 年前ぐらいからつくっています．この OS の開発目標の一つに，マトリックス・スイッチ方式の並列計算機用というのがあったわけです．したがって並列計算機でもタスクを並列にでも動かすことができるんで，学生実験で OS/omicron 上で並列処理の実験をしてみたりしています．

たとえば迷路の探索問題とか，ソートの問題，これはデモプログラムですが，こんなものをやってみて，どんなことが分かるかというと，並列の教育には実はビジアリゼーションというのが特に重要だと．また，おもしろいと．たとえば迷路の問題などは，本当にワァーッと並列に動くことがよく分かるんです．そうすると学生はそれをみるとおもしろい．ところが並列プログラムをつくるよりは，そのプレゼンテーション，とか，ビジアリゼーションとか，おもしろく見せるというほうにはるかに大きな労力がかかるというのが現実であります．実験の課題という単純な経験ですが，並列処理の振舞を見せるというビジアリゼーションについても，今後やっていかなければならないと思います．

9.6 人が原動力

最後に解決へ向けてでございますが，これは時間がないからやめましょう．結論として並列処理に限らず新しいことをやるには，第一は人が原動力だと．我が国の並列処理計算機の開発でも富田さんみたいな原動力のある人に，周りが協力してやる以外にないんだと．

第二に，コンピュータ産業に私は 1957 年からずっとかかわっているわけですから 35 年になりますけれども，最近，私は「コンピュータ産業は単なる技術産業じゃなくて一種の啓蒙産業だ」と思わなければいかんと思っています．啓蒙産業だとすると，使える環境づくりと，ユーザをどう説得するかということを本格的に考えないといけない．ユーザは常に王様ですから，先ほど昔の名前で出ていますとおっしゃった小柳先生の中にも出ていますけれども，ユーザというのは王様ですから，その王様のご機嫌を損じないような啓蒙をしていかないといけない．王様に対しては「超並列でやらないと時代おくれよ」というような風潮にしていかないとなかなか難しいんじゃないかと．

そのためには，問題点とすれば，日本の大学というのはプロジェクトを組んでハードウェア，アーキテクチャ，OS，開発環境，応用まで一貫してやる体力に乏しいですね．だいたい通産省は金持ち国なんですけれども文部省は貧乏国ですから(笑)，ICOT 対文部省の並列処理に関する重点領域研究の研究費を比較してみると，500 億円対 6 億円ですから……．

そういう意味で産学共同というものを上手にやっていく必要があるのではなかろうかと思っております．

以上でございます．

司会 どうもありがとうございました．