

報告

パネル討論会

新しい計算機アーキテクチャ——非ノイマン機能

昭和 52 年度第 18 回全国大会*報告

パネリスト

市川 忠男(KDD), 後藤 英一(東大), 坂村 健(慶大)
 箱崎 勝也(日電), 淵 一博(電総研),
 村岡 洋一(横須賀通研), 司会 相磯 秀夫(慶大)

はじめに

本稿は第 18 回情報処理学会全国大会において行われたパネル討論会の概要をまとめたものである。まとめに当っては、討論形式をとらずに、パネリストの主張または見解を概要としてそれぞれまとめ、パネル討論参加者の意見の主旨を最後に一括して載せている。

討論の主題は最近話題になっている「新しい計算機アーキテクチャ」であるが、「非ノイマン機能」という副題に総称されるものといえる。

非ノイマン機能

相磯 秀夫

「非ノイマン機能」を討論するに当っては、いわゆる「ノイマン機能」について定義しておく必要がある。「ノイマン機能」についての定義は明確なものはないが、古典的な計算機を眺めたときどんな特徴があるかを基準にすべきと考えている。

ノイマン型計算機の特徴とそれらに対する改良の試み(非ノイマン機能に対応)を簡単に列挙すれば表-1 のようになる。これからも明らかなように、今までにない新しい機能を称して「非ノイマン機能」と広義に定義したい。

非ノイマン機能が要求される理由は、(i)新しい応用分野の展開、(ii)価格性能比の改善、(iii)ソフトウェアの生産性改善、(iv)ハードウェアの急速な進歩、(v)本質的な問題解決の模索、などに深い関係があるように思われる。

表-1 ノイマン型計算機の特徴とその改良例

ノイマン機能	非ノイマン機能
(1) プログラム記憶方式	(将来とも変わらず)
(2) 逐次処理	並列処理・連想処理
(3) 線型アドレス記憶	ディスクリプタ方式・スタック・連想記憶・ハッシュ機構
(4) 命令とデータ	タグ・ゼロアドレス方式
(5) ソフトウェア	高レベル言語直接実行計算機・特殊専用計算機
(6) 決定性論理およびプログラミング	非決定性論理 (Stochastic computers)・バックトラック機能・問題適応型計算機 (adaptive computers)・学習機構 (learning machines)

一方、現状はどうかと考えると、新しい計算機アーキテクチャが容易に取り入れられる程単純なものではない。特に、商用計算機ではソフトウェアの開発が大変で、新しい計算機アーキテクチャ導入に伴うソフトウェア手直しのわずらわしさを考えると、導入の効果が分かっても手が出ないというのが実状であろう。それよりも、アーキテクチャを変えずに徹底的に LSI 化した方が価格性能比の改善が容易に得られることは明らかである。しかしながら、新しい応用分野の展開に伴って、新しい機能がアーキテクチャ・レベルで組込まれないと、本質的に、あるいは実用の面から考えて問題が解決できなくなる場合が出てくることも予想される。また、最近話題になっているソフトウェア開発のための生産性を抜本的に改善する上でも計算機アーキテクチャの研究は無視することはできなくなっている。現在のところ、商用計算機の多くの場合については、「非ノイマン機能」を積極的に受入れることにはならないが、これからの小型計算機や特殊計算機にはその価値が評価されるものと考えられる。いずれにしても、以下に実例を通して述べられるように、

* 日時 昭和 52 年 10 月 5 日, 12:30~14:30
 場所 東京工業大学

「非ノイマン機能」は目下のところ重要な研究課題になりつつあるといえよう。

参 考 文 献

- 1) A. W. Burks, H. H. Goldstein, and J. Von Neumann: Preliminary Discussion of the Logical Design of An Electronic Computing Instrument, IAS, Princeton University, 1946.

スタックマシン・タグマシン・ハッシュマシン

後藤 英一

非フォン・ノイマン・マシンの定義は人によって必ずしも一定していないが、司会者はタグマシンやスタックマシンもこれに入るとされている。

筆者たちのグループはここ数年、ハッシュ符号法を利用するデータ構造を付加した LISP (HLISP) システムとこの HLISP をホスト言語とする数式処理システム、HLISP-REDUCE の純ソフトウェアのインプリメンテーションを行ってきた。ソフトウェアのポータビリティを重視した結果、HLISP-REDUCE システムは現在アーキテクチャが全く違う機種、F230-75, H-8800, M-190, CDC 6600, ACOS 700 などの上で働いている。しかしこのようなソフトウェアシステムを作成してみると次の2点が痛感される。

1) 現在の大型機のアーキテクチャは数式処理を始めとする記号処理を高速に実行するには不適當である。

2) 現在の計算機のアーキテクチャは互換性(ポータビリティ)が高く、かつ実行能率が良好なソフトウェアを書くのに適していない。

これに関連してまずスタックについてのべる。スタックは帰納的(recursive)なプログラムには不可欠な機構である。数値計算指向の言語には FORTRAN のように帰納 (recursion) を許さないものもあるが、記号処理には帰納は絶体が必要である。マイクロコンやミニコンにはハードウェア化されたスタック機構を持つものが多いが、IBM 系の大型機とわが国の大型機には合理的なスタック・ハードウェアは組込まれていない(米国のバロース社、英国の ICL 社の大型・中型機には組込まれている)。ソフトウェアでスタックを構成する場合特に余分の時間がかかるのは、スタック領域の溢れの検査のステップである。従ってスタックのハードウェアとしてはスタック・ポインタの増減機構のみならず、スタックの溢れの検出割出し機構を備えているものでないとあまり有効ではない点をここに強調しておきたい。またサブルーチン・リンク専用

のスタックならば、純 LIFO (Last In First Out) 利用となるが、サブルーチンへのパラメータの受け渡しと局所変数の記憶領域にスタックを使用する場合にはランダム・アクセスも要求され、その実現法がスタック・マシン設計の中心課題といってもよい。最近のようにメモリの価格が低廉化してくると、複雑な凝った機構よりも、あっさりと高速大容量の RAM をスタック専用利用の方が得策かも知れない。

FORTRAN や ALGOL 60 のような言語では、データ型はすべてコンパイル時に決っている。これに対して LISP ではデータ型はすべて実行時に検査して、型に応じて異なった演算処理を行う。そのために各データの内部表現は必然的に型の情報を含むことになり、これを(広義の)タグ(情報)と呼ぶことにする。タグというと狭い意味では、機械語中の特定のビット部位を意味することもあるが、データ型の表現にはアドレス空間の分割など別の方法もあるので、上記のように広義に解釈する方が概念としては統一的であろう。純ソフトウェア手法によってデータ型の実行時検査を行うと型判別のスラップが増大し、実行速度は著しく低下する。これを避けるために多くのプログラミング言語は、コンパイル時にデータ型が決定される言語設計になっている。しかし記号処理特に数式処理では、例えば同じプログラム中の X という変数も場合により数値を表わしたり、数式を表わしたりするなど、その型が実行中に変化することを許すことが望ましい。このようなデータ型の実行時検査を高速化するには、タグの検査をハードウェア化したタグマシンが必要になる。タグは記号処理のみならず、数値計算やソフト作成上にも多大な効用があることをここに注意しておきたい。FORTRAN など書かれた数値計算プログラムの計算機の互換性を著しく損う要因として、数値データの内部表現の差異がある。とくに悪いのは整数型データで計算機の語長に応じ 16 ビットから 60 ビットまで多種多様の機種があり、整数のオーバーフローに関する処理も $\text{MOD } 2^M$ (M は語中のビット数)で計算するもの、エラーとして割出すものなど一定していない。浮動小数点数の場合にも、指数部の範囲は数学的根拠のない 16^{-64} から 16^{+63} とか 2^{-1024} から 2^{+1023} となっている。今日の多くのプログラム言語では指数部のオーバーフローに関しては全くお手上げの状態にあり、指数部 12 ビット 2 進基数の機種で開発デバッグされたプログラムを指数部 7 ビット 16 進基数の機種に移設する場合などには、多大な困難が

伴う。またアルゴリズムの設計上も指数部は大きな制約となっている。仮数部の有効桁数に関しても、多くのシステムでは1, 2, 4 倍長を選択利用できるようになっているが、4 倍長を超える精度に関してはこれもお手あげである。タグ付データで数値を表現すれば、任意多倍長整数演算、大指数超多倍長浮動小数点数演算などは容易に実現され、今日の進んだ数式処理システムではこれが実現されている（因みに筆者たちの数式処理システム HLISP-REDUCE を任意多倍長整数演算に便利だからという理由だけで使っているユーザもいる。）。この種のシステムのコンパイラ（又はインタプリタ）は数値の型を取扱う部分が全部省略できるのでそれだけ簡単になる。しかしこの種のシステムは、タグの判定を純ソフト的に実行時に行うので実行速度は、コンパイル時型決め法の場合と比較して著しく低下する。タグマシンでは1語長（例えば 32 ビット）のデータのそれぞれには少なくとも次の3種類の型を識別できるタグが付いているものとしよう。その3種とは短整数、短（1倍長）浮動小数点数、長い数（多倍長整数、大指数多倍長浮動小数点数など）を表わすデータ構造へのポインタである。このためには一見各語に2ビットのタグを取付つける必要があり、このようなビット損失はしばしばタグマシンの欠点と見なされてきた。しかしタグの表現法を工夫すればビット損失は大幅に減らすことができる。例えば 16^p を表わす7ビットの指数 $-64 \leq p \leq 63$ の中で $p=63$ の場合の仮数部はデータ構造へのポインタ、 $p=-64$ の仮数部は短整数を表わすものとし、その他の場合は短浮動小数点数の指数というタグの表現法を使えば、 $27=128$ 種中の2種だけが特別なタグなのでビット損失は $\log_2 \frac{128}{126} = 0.028$ ビットにしか相当しない。

筆者のグループではスタック、タグとハッシュ機構をハードウェア化した FLATS と呼ぶマシンを設計試作中である。ハッシュ・マシンについては文献 3) を参照願うこととするが、ハッシュ法はノイマン以降の技法であるから非ノイマン型マシンといえよう。

参 考 文 献

- 1) M. Sassa, E. Goto: A Hashing Method for Fast Set Operations, Info. Proc. Letters, Vol. 5, pp. 31~34 (1976).
- 2) E. Goto, Y. Kanada: Hashing Lemmas on Time Complexities with Application to Formula Manipulation, ACM-SYMSAC 76 York Town Heights NY (Aug. 1976).
- 3) 井田哲雄, 後藤英一: ハッシング・プロセッサ, 情報処理, Vol. 18, No. 4 (1977).
- 4) 後藤英一, 佐々木建昭: 計算機による数式処理の現状, 情報処理, Vol. 18, No. 9, pp. 830~837 (1977).

人工知能とアーキテクチャ

淵 一博

前にお話があったように、エレクトロニクスの技術の進歩によって、アーキテクチャ問題にも新しい時代が来ようとしている。そこで私は、「道楽としてのアーキテクチャ研究」を提唱したい。

古き良き時代には、個人や小グループの創意工夫が許された。しばらく、それが困難な時代が続いたが、今度は、より大きな規模でそのような時代が復活することを期待したいのである。

（追記：ここで「道楽」といったのは、商業ベースでない、理想追求的な、真剣な、非妥協的な試みという積りである。楽しい工夫がなくて、創造ということがあるのだろうか。）

従来の商業機路線から大きく離れた試みが必要なのは、計算機というものの性格による。小手先の改良がソフトウェア・コンパティビリティを損って、かえってマイナスになったことは多い。それならばむしろ、 $\times \times \times$ コンパチでいいわけで、それでも済むのが計算機の原理的な良さでもある。私自身、十数年前には、コンパチ路線を主張したこともある。

我が国のメーカがコンパチ路線に固まった今では、逆に新しい試みが望まれるが、それは個人や小グループの創意工夫に期待するよりないだろう。

新しいアーキテクチャの試みの一つのポイントは、この20年ほどのソフトウェア技術の経験を活かすことであろう。「高水準言語マシン」はその線に沿うものと考えられる。ところで、本日私に与えられたテーマは、人工知能 (AI) 用言語とアーキテクチャということである。

70年代の前半、AI研究グループの中で新しいプログラミング言語が生れ、PLANNER (MIT), QA4/QLISP (SRI), CONNIVER (MIT) などが有名になった。これらは、それまでの普通のプログラミング言語より高水準な、いくつかの特徴を持っている。

(1) パターン・マッチングの利用

- a それによるリスト処理
- b それによる手続き呼出し
- c それによるデータベースの検索

(2) 非決定性の導入

- a バックトラッキングによる実現
- b コルーチンによる実現

(3) コンテキスト構造などの導入

非決定性の問題は、その実現法を含めて多くの議論を呼んだ。この非決定性とは若干異なるだろうが、最近 Dijkstra など非決定性を積極的に評価していることは注意すべきであろう。

パターン・マッチングは、強力なマクロな機能である。これによって、プログラミングのスタイルも大きく変わってくる。

これらの特徴をマシン・アーキテクチャに関連させるとどうであろうか。マクロな機能を直接的にマシンに組込むことには異論があるだろう。複雑になりすぎる危険がある。それはソフトで実現するというのが正統的な行き方であろう。

一方、高水準な機能の効率化の方がねらいやすい面がある。FORTRAN マシンでは、変ったアーキテクチャを考えてもあまりしようがないであろうが、もっと高水準の言語では、むしろ、アーキテクチャ上の工夫が大きな効果をもつ可能性がある。

たとえば、制御のためのスタック構造は、基本アーキテクチャとして実現されることが望ましい。これはすでに試みられていることである。上述の AI 用言語の制御には、「拡張された」スタック構造が基本となる。

また、パターン・マッチングは一般には複合操作であるが、これをハードあるいはハードに近いレベルで基本操作に組込めば、効率が大きく向上するだろう。

ところで、AI の分野は、ソフト的な面でやらなければならないことが山積している。AI 用言語というのも、実のところ、今でも確立しているわけではない。それなのにハード（のアーキテクチャ）をいうのは何故か。困難な課題には優秀な武器が欲しいのである。

いま自然言語を扱うとする。1文の解釈に1時間かかるのと、数秒で済むのでは人の頭の働きも大いに変ってくる。ソフト的な工夫が本質的に必要なのは当然であるが、それを支えるのはハードである。

話を元に戻すと、先ほど述べたような AI 用言語はほとんどの場合、LISP の上で実現されている。LISP は確立された言語と見てよいし、水準からすればマシンに近い。そこで LISP マシンの方が AI 用としても現実性があるといえる。この LISP マシンの試みは、すでに各所で行われている。電総研でも1グループが従事している。

LISP マシンで良いのであるが、私としては「道楽」ついでに、もう少し別の道を探ってみたいと思っている。それは「述語論理的プログラミング」の考えを取り入れることである。

PLANNER などは「手続き的定理証明システム」と呼ばれたことがある。これには歴史的事情がある。かつての純定理証明システムの「非効率さ」のアンチテーゼとして、「手続き主義」が復活し、それを具体化しようとしたものである。それは進歩の一つではあったのだが、逆に、プログラミング言語——形式言語としての性格がはっきりしなくなった面がある。

また LISP についても、その純 LISP 部分は論理的にすっきりしているが、PROG フィーチャは妥協の産物と見られる。PLANNER の後続の ACTOR/PL-ASMA では、それに対する試みがある。

一方、述語論理の方では、その部分系であるホーン集合について効率的な証明手続きが発見された。しかも、これは「手続き的解釈」ともびったり適合するものである。

この考えをベースにして PROLOG と呼ばれるシステムがフランスで作られている。PROLOG は純 LISP に相当するものであるが、この考えを進めると PROG フィーチャに相当するものもカバーできる。これを含めた拡張を私は EPILOG と名付けた。

このところ、goto less プログラミングが言われているが、述語論理的プログラミングは、さらに、assignment less プログラミングとも特徴づけられる。また、これをベースにするプログラミング言語は、前に挙げた AI 用言語の特徴をもっている。一方、それらの言語と異なって、述語論理の形を保存しているために構成が単純であり、セマンティクスも明確である。これは、プログラムの検証や合成に有利と考えられる。

言語プロセッサの立場から見ると、そのエレメントは、実は、LISP とほとんど共通である。速度効率の面でも、LISP と同程度を実現できる見通しがある。マシン・アーキテクチャを考えても LISP マシンと似たことになるだろう。

実際、LISP のベースであるラムダ計算と述語計算には、理論的に自然なつながりがある。両者の表現と計算にはそれぞれの有用性がある。そこで、両者を融合したものを効率的に実行するようなマシンのアーキテクチャを考えてみたい。と、こういうことを今思っている。

パターン認識と連想処理

市川 忠男

米国テキサス州ダラス市で開催された NCC-77 で Architecture for Pattern Recognition Applica-

tions と題するパネル討論を企画し、パターン認識の立場から見たアーキテクチャの諸問題を討議した⁵。もちろん単純に合意点に到達できる性質のものではないのだが、具体的な物に裏づけされた話をみると、今回のパネルの言葉を借りるならば、“非ノイマン化”の傾向にあると云える。

非ノイマン化の論拠の一つとして処理スピードの向上という点がとり上げられる。これについて CDC の G. R. Allen は、リモートセンシングデータの処理の立場から 1 画素当り $10^2 \sim 10^3$ 回の操作が必要で、また毎秒 10^6 画素を処理する必要がある。したがって、毎秒 $10^8 \sim 10^9$ 命令の処理能力が要求されるといっている。カーネギーメロン大学の D. Raj Reddy も音声認識の立場から毎秒 $10^7 \sim 10^8$ 命令を目標に置いている。そこでなにがしか非ノイマン的な仕組みを考えるわけであるが、これが経済的に見合うかどうかが問題である。

CDC では 40 個のプロセッサをリング状につないだ SYBER-IKON¹¹ というシステムを開発した結果、従来 CDC 6600 でほぼ 300 ドルかかっていた 1 フレーム当りの処理が 3 セントで済むようになったといっている。また、カーネギーメロン大学の Reddy は、DEC 1080 で 1~5 ドルかかっていた連続音声の 1 区切り当りの認識コストがマルチプロセッサタイプのシステムでは 10 セントで済むだろう、……10 セントで済めば、この種のデータ取得を人間が行った場合の給料などから算定される費用に照らしてみても十分効果が上る、として現に C. mmp²⁾ や Cm^{*3)} といった新しいアーキテクチャを開発した。

こういうわけで、パターン認識そのものが社会的ニーズに応えるべくクラシックな認識問題から、さらにその土俵を広げつつあり、その結果、パターン認識という一応用分野からも高性能な新しいアーキテクチャの出現が強く望まれるわけである。一方、観点を変え、逆にアーキテクチャ技術の発展がパターン認識の対象と意識を変えつつあるととらえることもできる。対象と意識がどのように変りつつあるかということで先のカーネギーメロン大学の場合を例にとると、単語音声を実際に認識しようとする立場から、連続音声といった、認識対象に対する制約をやわらげたより実際的な場で、目的に沿った好ましい理解を示すようなシステムを具体的に作り上げていこうという方向に

動いている。

Reddy はまた、パネル討論に 8 ミリフィルムを持ち込み、“I saw the statue of liberty flying to New York” という声を聞かせて、ニューヨークへ飛んで行く飛行機の窓から自由の女神を見ている情景と、自由の女神が空飛ぶじゅうたんにまたがってニューヨークめざして飛んで行くのを見たという情景とを見せていた。このように、文の意味を正しく把握するには文脈に関する知識や情況の判断が必要だということから HEARSAY¹² というシステムを開発している。ここでは、物理的観測値のレベルから概念レベルにいたるまでの数多くのレベルに対応した knowledge source を用意し、それぞれのレベルで現在たてている推定が妥当なものであるかどうかをテストし、テストの結果、必要に応じてこれらのレベルを登り降りして推定をたてなおしながら上位の概念レベルへと進んで行く手法をとっている。Knowledge source は、その過程で互いに交信できるように同一構造のデータベースにまとめられていて、先に述べた C. mmp などのマルチプロセッサで並列に処理される。

もう一つ、マサチューセッツ大学の VISIONS¹³ というシステムを紹介しよう。これは E. M. Riseman らが屋外風景の解析をテーマに開発しているシステムで、アレイ状の論理でイメージの細かな特徴から粗い特徴までを階層的に選りだし、この階層構造を登り降りして視覚的な特徴と概念的な知識とのインタフェースをとりながら主だった対象を把握するという仕組みになっている。

以上の 2 つの例で分かるように、1 つは音声であり、1 つはイメージであるが、手法的に見て共通していることは、認識あるいは理解に必要な基本的な知識をデータベースとして整理しておいて、アソシアティブなサーチによって可能性の高い方向を見つけ、演繹的な手法でデータベースを設定、かつ更新してゆくという点にある。

ここで、このような手法をハード的にサポートするものとして連想プロセッサが浮んでくる。連想プロセッサの並列性が処理の高効率化に貢献するわけであるが、連想とはいっても現在のところ直接原情報に内容アドレッシングを適用しているにすぎない。もう少し人間の連想に近い高度な機能が望まれる。

筆者はかねてより符号理論を用いた新しい連想方式を提案していた⁶⁾。この連想方式は、データ間の関連の度合いが符号間距離で表わされるように符号化され

* T. Ichikawa (moderator), K. S. Fu, C. C. Foster, G. R. Allen, D. Raj Reddy, J. Sklansky, M. J. Castelhberg (panelists).

た原データに、誤り訂正処理を行って個々のデータ固有の性質から離れた基本的な構造を求め、基本的な構造の上で一致度の検定を行って関連の強いデータを指定するという考えに基づいている。原データに誤り訂正を適用する操作を blurring と呼び、そうして得られたデータ空間を blurred data space と呼ぶ。このようにして、関連度の評価を直接原空間で行う場合に要求される複雑な計算処理は blurred data space での一致検定という単純な操作に置き換えられる。すなわち、誤り訂正と内容アドレッシングの技法を結びつけることによって論理機械にある種の直感性に似た効果を付与したと見ることができる。

手書き文字の認識を想定したシミュレーションによってその動作を確認したのち、慶大相模研の協力を得てアーキテクチャ設計を完了し ARES⁷⁾と名づけた。ひきつづき音声やイメージ理解のためのセマンティックデータベースにおけるアソシアティブサーチに主眼を置いて、そのマルチマイクロプロセッサ構成の検討を進めている⁸⁾が、ここではデータの深層構造を求める目的で blurring の階層化を試みている。

このようなマルチマイクロプロセッサシステムでは、拡張性・汎用性のある組織的な構成を持たせることができるし、またマイクロプログラミングによって機能にも柔軟性がでてくる。したがって、アーキテクチャが非ノイマンであるということは必ずしも融通性に乏しい専用機であることを意味しない。

高性能化から一歩進んで高機能化を意識したい。

参 考 文 献

- 1) CDC: SYBER-IKON, Image processing system design concepts, Control Data (January 1977).
- 2) W. A. Wulf et al.: C. mmp-A multi-mini-processor, Proc., FJCC 1972, pp. 765~777 (December 1972).
- 3) R. J. Swan et al.: Cm* A multi-microprocessor, Proc., NCC-77, pp. 637~644 (June 1977).
- 4) V. R. Lesser et al.: Organization of the Hearsay II speech understanding system, IEEE Trans. ASSP-23, 1, pp. 11~24 (February 1975).
- 5) A. R. Hanson et al.: The design of a semantically directed vision processor (Revised and Updated), COINS Tech. Report 75C-1, The Univ. of Massachusetts, Amherst (February 1975).
- 6) T. Ichikawa et al.: Recognition of handprinted characters with associative read-out of patterns in a memory, Proc., Second Int'l Joint Conf. on Pattern Recognition, pp. 206~207 (August 1974).
- 7) T. Ichikawa et al.: ARES-A memory, capable of associating stored information through relevancy estimation, Proc., NCC-77, pp. 947~954 (June 1977).
- 8) T. Ichikawa et al.: A multi-microprocessor ARES with associative processing capability on semantic data bases, Proc., NCC-78, pp. 1033~1039 (June 1978).

高級言語マシンのアーキテクチャ

箱崎 勝也

非ノイマン機能を語る上で高級言語マシン（以下、HLLM と略す）は欠かせないものであるが、新しい計算機アーキテクチャとして HLLM を語るのには多少抵抗がある。アイデアとしての HLLM は決して新しくはない。しかしながら、最近のハードウェア技術の進歩は単なるアイデアであったアーキテクチャを実現性の高いものにした。その意味では高級言語マシンは新しい計算機アーキテクチャである。まず、定義を述べよう。

HLLM の定義はあまり明確ではないが、ここでは「高級言語で書かれたプログラムの翻訳/実行に適した計算機」と広く定義しておく。これは従来のノイマン形計算機では必ずしも高級言語の翻訳/実行に適さない場合があることを暗に含んでいる。これに関連して、言語の性質、性能に対する要求、使用環境と HLLM の関係を表-1 に示す。

高級言語マシンで用いられる非ノイマン機能には次のようなものがある。

(1) タグ付アーキテクチャとデータ記述子

ノイマン形計算機では記憶装置上のデータとプロシージャは区別がないのに対し、タグ付アーキテクチャではプロシージャとデータ、更にデータの属性が各々に付せられたタグによって明確に区別される。タグ付アーキテクチャにはデータの属性を記述するデータ記述子がデータ側に用意されるものもある。ノイマン形では命令がデータを制御するのにに対し、データ記述子はデータ側が命令機能を制御することになる。これは HLLM に要求される複雑なデータ構造を記述するためにも有効である。

表-1 言語、使用環境と高級言語マシン

	HLLM 向	ノイマン型向
言語の性質	• 言語の持つ機能レベルが高い言語 • 動的（実行時）に決定される要素が多い言語 APL, LISP, SNOBOL, GPSS	• 機能レベルが低く機械語に近い言語 • 静的（翻訳時）に決定しうる要素が多い言語 COBOL, FORTRAN
	ALGOL PL/I	
性能要求	プログラム入力・結果迄の応答時間	オブジェクトの実行時間を重視
使用環境	• 会話的使用 • プログラムの再使用回数小 • 変更・修正多	• 一括処理 • 同一プログラムの繰返し使用 • 変更・修正少

(2) スタック

スタックには逆ポーリッシュ・ストリングで表現された算術式を効率良く処理するために用いられる演算用スタックとサブルーチンなどの CALL/RETURN に用いられる制御用スタックとがある。

いずれも多くの HLLM で効果的に用いられている非ノイマン機能である。

(3) 高レベルインタプリタ

高級言語の持つ高度な処理機能を効率良く実行するために、ハードウェア、ファームウェアあるいはソフトウェアで実行される高機能の命令とそのインタプリタが用意される。高級言語マシンではノイマン形計算機に比べ命令の持つ機能レベル、インタプリタの機能と性能が高いのが普通である。

高級言語によるプログラミングが普及するにつれて、商用機においても多かれ少なかれ高級言語指向性を持ったアーキテクチャが採用されている。いくつかの商用機の高級言語指向性を概観してみよう。紙面が限られているため、ごく一部についてしかふれられないこととお断りしておく。

(1) B 6500/7500 とその後継機種

バロース社は古くから高級言語指向を強くうち出しているが、B 6500/7500 のアーキテクチャには主として Algol を対象として、次のような機能が採用されている。

- データ記述子 (descriptor) の導入
- スタックを利用した逆ポーリッシュ表現の命令
- 制御用スタックによるブロック構造の制御

(2) B1700

マイクロプログラムによるインタプリタを各種高級言語ごとに用意することによって、HLLM の問題点の一つである複数の高級言語を実行することを可能にしている。ビット単位でアドレス可能な構造がインタプリタの作成に効果的に活かされている。

(3) IBM 370 の APL 用付加機構

IBM 370/145 などではノイマン形計算機では処理効率が悪い APL 言語の効率を向上させるために、マイクロプログラムによる APL エミュレータが付加されている (APL Assist Feature)。APL エミュレータへ制御を移すために特別な命令 (APLEC) が用意されている。これもマイクロプログラムによるインタプリタの導入の例である。

(4) ACOS シリーズ 77

日本電気一東芝の ACOS シリーズ 77 にはノイマ

ン形アーキテクチャをベースに、高級言語指向のいろいろな機能が用意されている。主なものに次のような機能がある。

- 制御用スタックを用いた CALL/RETURN 命令
- 添字のアドレス計算用命令 (Compute Subscript)
- 部分的なデータ記述子の採用
- 文字列サーチや移送のための高機能命令

(5) パーソナル・コンピュータ

中・大型機の高級言語指向性は多くの言語を処理しうる汎用性を持たせた上で、いくつかの言語に対する指向性を強化したアーキテクチャが多いが、個人用コンピュータでは特定の言語だけを対象にした HLLM がある。APL と BASIC を対象とした IBM 5100、BASIC 用の Wang 2200 などがその例である。

高級言語マシンのアーキテクチャを商用機を中心に述べたが、HLLM が多くの商用機の中に本格的に採用されるようになるにはまだ多くの課題が残されている。

(1) コスト/性能

非ノイマン形アーキテクチャの HLLM がノイマン形にとって替るためには、コスト/性能が優れたものでなければならない。ハードウェア技術の進歩がその可能性を高めている。ノイマン形に比べて複雑なアーキテクチャを持つ HLLM においてはハードウェア/ファームウェア/ソフトウェアのトレードオフが重要である。

(2) 多重言語の処理

大規模なソフトウェアシステムを作成するとき、複数の高級言語を使用することが多い。異なる言語を同時にサポートし、異なる言語プログラムを結合しうるようなアーキテクチャであることが必要である。多くの実験的 HLLM でもこの問題を解決したものは少ない。

(3) 互換性/移行性

莫大なプログラム遺産にはアセンブラで書かれたものや、高級言語で作成されてもオブジェクトプログラムしか残っていないものが数多くある。これらのプログラムと HLLM との互換性、移行性の問題がある。機械語プログラムを高級言語プログラムに変換するようなアプローチも研究に値する。

(4) 大形・超大形機に対する適性

大形・超大形機においては、パイプライン制御のよ

うに高度な並列処理が行われており、HLLM のように複雑なアーキテクチャよりも、個々の命令は単純な機能をはたすアーキテクチャの方が望ましいとされている。複雑なアーキテクチャを高性能に実現する構成法や超大形機に適した HLLM アーキテクチャも課題の一つである。

以上の他にもシステム記述言語の HLLM, システムアーキテクチャから見た HLLM の位置づけ、更には超高級言語 (VHL) との関係等さまざまな課題がある。

種々の課題はあるものの、ほとんどの使用者が高級言語を介してコンピュータを使用している状況を考えて、これからのアーキテクチャは徐々に高級言語マシンの要素の強いものになると思う。

並列処理アーキテクチャ

村岡 洋一

並列処理技術は期待はずれであったといわれている。しかし、Illiac IV の例等をふり返って見ると、その失敗の理由の多くはマネジメントに起因するものであり、CRAY マシンや TI の ASC 等によっても分かるように技術的には必ずしも不成功ではなかったといえよう。むしろ LSI メモリ技術動向等に助けられて、並列処理に必要な多ポートメモリが経済的に有利に実現できるようになって来ている。

並列処理技術を、メモリ技術と対応づけると次のようになる。

LSI メモリの特徴:

(1) 性能向上

メモリアクセスタイムの CPU サイクルタイムへの接近。

→プロセッサの多重化 (複数命令のストリームでの並列処理)。

(2) 経済的なモジュラ化の可能性

ポート数増加が経済的に実現可能。

→データの並列処理 (多数データへの同一演算の並列適用)。

商用コンピュータのアーキテクチャの設計には、だんだん新規なアイデアが入られる余地が小さくなって来ているといわれる。しかし、応用分野を限ればまだまだ冒険的な試みが可能である。

その第一の例は、Flynn によって提案されたリソース共用マルチプロセッサである¹⁾。これは上に述べた LSI メモリの第一の特徴を利用しようとするものである。本プロセッサでは、複数のプロセッサが同一のメ

モリを共用しているが、共用の対象は単にメモリに止まらず、演算装置 (いわゆる E ユニット) にまで及んでいる。このような構成が効率良く動作するためには、プログラムからの E ユニットやメモリへの使用要求が適当なものでなければならない。E ユニットを使う命令 (加減算命令等) の出現する比率は、科学計算で例えば 30% 程度、商用計算になると 5% 以下程度であるから、数台から数十台程度のプロセッサが単一 E ユニットの共用できる可能性がある。リソース共用マルチプロセッサは、複数の異なるプログラムの命令ストリームを並行して処理することを主目的としているが、一つのプログラムを複数の独立なセグメントに分解して並列処理しても勿論差つかえない。その場合には並列処理可能な複数セグメントへの分解が問題になるが、これについては次の例で説明する。

LSI メモリの第 2 の特徴を利用したデータ並列処理システムの例は、Illiac IV を始めとして多々見られるが、このようなアレイタイプの計算機の成功例は皆無に近い。理由は色々あろうが、そのうちのいくつかの技術的な問題を、全くユニークな考え方で解決しようという新しいアレイタイプ並列処理システムについて説明する²⁾。このシステムの主要な特徴は次のとおりである。

(1) コンパイラによる並列処理性解析

従来この種のシステムでは、並列処理の指定を利用者に委ねていた。これに対して、このシステムでは既存言語でのプログラミングを許し、並列処理性の解析は全くコンパイラが責任をとる。

(2) ユニークなメモリマッピング

アレイタイプ並列処理システムの効率を決める重要な鍵の一つはメモリマッピングにある。データに対しては並列に演算できても、そのためのデータを sequential にメモリから読み出すのでは何にもならない。従来とも効率的な並列データアクセスのできるメモリマッピングについては色々と考えられて来ている。本システムではこれらをさらに一歩進めて、マッピングを旨く行うため余分なメモリユニットを設けることまで考えている。

このようなユニークなアプローチの背景には次のような割り切った考え方がある。まず第一に応用分野を極く限定している。次にその分野内での問題の性格 (具体的にはプログラム) が徹底して分析され、共通項が見つけ出されている。上に述べた並列性解析アルゴリズムやデータマッピングは、この共通項を基にしてい

る。すなわち、すべての分野のすべてのケースに対する汎用解ではなく、一部の応用分野の代表的なケースに的をしぼり、その範囲で最適な方法を採用することを目的としている。

メモリ素子に加えた論理素子の LSI 化に有利な方式技術は、多数プロセッサ構成であるといわれている。多数プロセッサ構成の代表例は、いわゆる平等マルチプロセッサと専用プロセッサ（ポリプロセッサとも呼ばれる）である。汎用な多数プロセッサを構成する場合に問題となるのは、制御表等いわゆるソフトウェア資源への競合である。現在の汎用オペレーティングシステムを基に考えれば、例えば 10 台以上の CPU から成る平等マルチプロセッサを効率良く稼働させるのは困難と思われる。ポリプロセッサはこの問題を解決するのに有利といわれているが、各専用プロセッサの負荷バランスを保つのが難かしいという欠点がある。このために、やはりプロセッサ台数に比例した性能向上は期待できない³⁾。これらの観察を基にすると次のようなことがいえよう。

LSI 技術によるハードウェア価格の低下を享受する一つの方法は、問題向きシステムを off-the-shelf パーツ（例えばマイクロプロセッサ）を使って組み立てることである。FFT 等科学技術計算や、データベース処理等が、問題の例である。分野を限れば、アーキテクチャのチューンも容易である。狭い応用分野に特化したシステム（というより専用プロセッサ）を構成することは、ハードウェア価格が十分に低下すれば経済的にひきあう。このような専用プロセッサは汎用プロセッサ又はネットワークに接続されて、汎用システムの一資源として使われるであろう。マイクロプロセッサ等を使った専用プロセッサの開発例^{4), 5)}もある。

専用プロセッサをサポートするプログラミング言語は、できるならば FORTRAN 等既存のものを使いたい。そのために、コンパイラにより多くの処理をさせるアプローチを考える必要がある。専用プロセッサ用ではないが、コンパイラの機能分担を増やしてハードウェアの最適設計を狙うアプローチが見直されているようである⁶⁾。

参 考 文 献

- 1) Flynn: Shared Resource Multi-processing, Vol. 5, No. 4, pp. 20~28 (1972).
- 2) 村岡: 最近の並列処理研究動向, 情報処理アーキテクチャ研究会資料 19 (1976).
- 3) 池原: 待ち行列モデルによる機能分散型分散処理方式の性能評価, 情報処理, Vol. 18, No. 11, pp. 1102~1109 (1977).
- 4) Aiso, et al.: An Approach to Parallel Processing for Continuous Dynamic System Simulation with Microprocess-

os, UJCC, pp. 172~177 (1975).

5) Aiso, et al.: A Very High Speed Microprogrammable Pipeline Signal Processor, IFIP, pp. 60~64 (1974).

6) Electronics, pp. 30~31 (Dec. 23/1976).

問題適応型計算機アーキテクチャ

坂村 健

情報処理システムにおける動的なオブティマイズ手法が注目されている。動的なオブティマイズ手法とは計算機の実行動作をもとにして実行時負荷の大きな所のみをオブティマイズするもので、明らかに現在一般に行われている全リソースに対して行われる静的なオブティマイズ手法とは異なる。このような手法の理論的裏づけとしては Knuth, Darden, Heller などが報告している¹⁾コードの 3~4% 以下の部分で実行時間の 60% 以上が費やされているという事実や, Saal などが指摘している²⁾命令セットの使用分布の偏りに基づいているのであるが, CMU の Hansen の言うように³⁾, 現在の計算機システムはこの事実をほとんど利用していないので, コンパイラを例にとるならオブティマイゼーションは一様となり, その結果コンパイラのデザインを困難にし, またユーザにも目的に応じてコンパイラの選択を要求することになる (たとえば IBM は FORTRAN, G レベル, H レベルなどオブティマイゼーションの異なるいくつかのバージョンを持つ)。すなわちオブティマイゼーションはプログラムの動的な振る舞いに基づき行われるべきである。そこで彼はそのプログラムの実行回数と動的な統計情報によりオブティマイゼーションの度合いを変え, コンパイル時間+実行時間が最適化される Adaptive FORTRAN システムを提唱した。そしてその結果も従来の FORTRAN と比べると優れているという。また, Texas 大の P.R. Blevins らも動的な振る舞いを考慮してオペレーティングシステム (OS) のオブティマイズを行うべきであると主張している⁴⁾。これは OS に対してのリソース要求を OS 内部のモニターで測定しそれを統計モデルと対比させ解析することによってどのようにリソース要求に答えるかの決定をする。そして実験的にこのような動的オペレーティングシステムを製作したところ従来のラウンドロビン方式に比べ効率が高かったという。

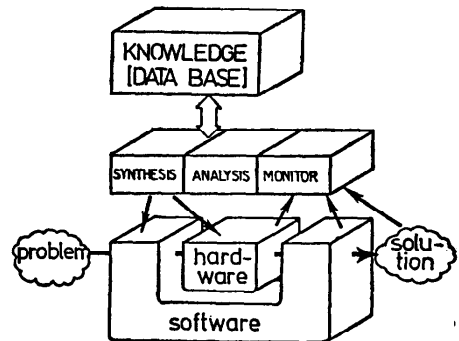
このように動的な情報をもとにしたオブティマイゼーションの効果は大きい, 以上紹介した例は二つともソフトウェアの問題であって, 計算機の構造 (アーキテクチャ) に対してのオブティマイゼーションでは

ない。明らかにアーキテクチャレベル（狭義に考えるとハードウェア）での動的なオプティマイゼーションはやっかいである。これは、ソフトウェアには概念そのものに可変性があるのに比べ、ハードウェアにはそのようなものはないからである。

ハードウェアレベルでの動的オプティマイズの考え方は、仮想記憶構造をもった計算機でのページ入れ換えアルゴリズムの1つであるLRUや、ページングマシンにおけるワーキングセットの概念にわずかに見られる程度であった。それが、1970年代になるに従い動的なオプティマイズがより有効に行える可能性が強まってきた。すなわち高速半導体メモリの発展とともに登場してきたダイナミックマイクロプログラミングが十分実用に耐えるようになってきた。マイクロプログラムの概念は古くからあったが、計算機の製造技術として使うのではなく問題に計算機を適応させるための道具として使うのである⁵⁾。制御記憶の内容をダイナミックに、問題に応じて入れ換えてやる。問題に応じて最適な命令セットを用意してやる。これは、書き換え可能な制御記憶の発展とともに初めて可能になった。しかし制御記憶が入れ換え可能となったのは適応型計算機の手段が提供されただけで、問題の本質はどのように制御記憶の内容を入れ換えるか、どのようにして問題に最適な命令セットを用意するかである。そこで前述した動的なオプティマイズ手法とこのダイナミックマイクロプログラムを結びつけるような概念が生れてきた。具体例を示そう。

A. M. Abd-Alla らは⁶⁾、プログラム中にあらわれるループに注目しループの動特性（ループ中で参照されるメモリの参照回数）を検出しそれを解析することによって最も頻度の大きいループと、その上で最もよく使われるメモリのリージョンを探し出し、それをマイクロプログラム専用レジスタに置き換えることによって、アーキテクチャのオプティマイズ（これをアーキテクチャチューニングと言う）を行う手法を提案している。ループ中のメモリ参照命令をレジスタ参照命令に置き換えループを出る際に再びもとに戻す機能をダイナミックマイクロプログラムで実現する。

また筆者は、命令の実行時遷移過程（インストラクションパターン）に注目する⁷⁾。これは、用意されている命令がどれ位の頻度と、どのような遷移で実行されるかの関係に注目するもので遷移頻度の大きな命令の組を新しい命令として、ダイナミックマイクロプログラムを使い中間言語レベルで再構成する。そして、これ



feedback on information processing

図-1

を新しく合成された命令として制御記憶に登録するという方法でアーキテクチャの動的オプティマイズを行う。このようにアーキテクチャレベルでは応用問題を汎用命令セットの上で実行させた結果を動作の局所性に注目して解析することによって応用問題向きの命令セットが合成でき、ダイナミックマイクロプログラムにより制御記憶の内容を入れ換え動的なオプティマイズが行われる。この効果は両者とも HP 2100 などを使った実験によると問題によるがチューニングの効果は平均して 2~6 倍程度の速度向上とされている。

計算機の誕生以来現在まで、性能評価は地道に行われ、その結果は人手によりフィードバックがなされてきた。しかし現在では計算機の構造が複雑になりすぎて、その解析が困難になりつつある。このような背景のもとで、現在のところははまだ研究の域を出ず、その可能性が示されたにすぎないが、動的なオプティマイズのための機構を備えた計算機（これを Adaptive Machine という（図-1 参照））、すなわち、「性能評価を計算機自身が行い、計算機自身で性能改善を行うことのできるマシン（計算機）」の意義は大きく、またそれに対する期待も高い⁸⁾。

参 考 文 献

- 1) D. E. Knuth: "An Empirical Study of FORTRAN Programs", Software—Practice and Experience, vol. 1, p. 105 (1971).
- 2) H. J. Saal and L. J. Shustek: "Microprogrammed Implementation of Computer Measurement Technique", MICRO 5 (1972).
- 3) G. J. Hansen: "Adaptive Systems for the Dynamic Run-Time Optimization of Programs", CMU internal report, (March, 1974).
- 4) P. R. Blevins and C. V. Ramamoorthy: "Aspect of a Dynamically Adaptive Operating System", IEEE Trans. on Computers, C-25, No. 7, pp. 713~723, (July 1976).
- 5) K. Sakamura, et al.: "A Debugging Machine—An Ap-

- proach to an Adaptive Computer", Proceeding of IFIP Congress 77 pp. 23~28.
- 6) A.M. Abd-Alla and D.C. Karlgaard: "Heuristic Synthesis of Microprogrammed Computer Architecture", IEEE Trans. on Computers C-23, No. 8, (1974).
- 7) 坂村, 相磯: 「計算機アーキテクチャの自動最適化に関する考察」 電子通信学会論文誌 Vol. 60.-D No.11/1970, p. 929.
- 8) 坂村, 相磯: "A Learning Machine" 情報処理学会計算機アーキテクチャ研究資料 29-3, (1977).

おわりに

以上のように, 最近の研究成果のいくつかを通して, 「新しい計算機アーキテクチャ」へのアプローチが紹介されたが, これらに対して, 「非ノイマン機能」としての保護機構の意味(京大 池田氏), 大型計算機と高レ

ベル言語直接実行との関係(東大 元岡氏), 並列処理における安価なマイクロプロセッサの意義(日立 吉村氏), 並列処理適用の限界(横須賀通研 村岡氏), ソフトウェア開発とアーキテクチャとの関係(東大 後藤氏) などについて熱心な意見の交換があった。

この討論からも明らかなように, 「非ノイマン機能」に関する明確な定義はなく, 今までに見られない新しい機能を対象にしているといえる。そのような意味からも, より広い応用分野を通して, 将来の計算機アーキテクチャのあり方を論ずる必要性があるように思われる。