2.4 パネルディスカッション
「並列処理と新世代コンピュータ」

座　　長　相磯　秀夫　　慶応義塾大学

パネリスト　雨宮　真人　　日本電信電話公社
古川　康一　　ICOT
D. May　　Inmos（英国）
E. Shapiro　　Weizmann Institute of Science（イスラエル）
S. J. Stolfo　　Columbia Univ.（米国）

座長：このパネルディスカッションは「並列処理と新世代コンピュータ」と題して私（慶応義塾大学理工学部教授）が，本セッションの座長を勤め，また，プログラム委員会の委員長として参加できることは，この上ない名誉である。

本セッションでは，並列処理という極めて重要で，かつ最も興味深いテーマを，新世代コンピュータの革新的可能性に関わる観点から考察したい。

VLSI，革新的アーキテクチャ，または新規なプログラミング言語といった進んだ技術を駆使する並列処理は，高まりつつある処理能力を得る手段として誰もが認める点であろう。しかしながら，この目標を達成するには，極めて重要な，そして深刻な問題が残されている。したがって，本セッションの主旨は，論理型プログラミングおよび革新的コンピュータ・アーキテクチャにおける問題点に焦点をあてて，並列処理の技術面を検討することである。

まず，並列処理の研究および論理型プログラミング言語や革新的コンピュータ・アーキテクチャといった研究分野で活躍中の５人のパネリストを紹介する。

最初のパネリストは雨宮真人博士で，九州大学で電子工学の博士号を取得され，日本電信電話公社武蔵野電気通信研究所に勤務されている。雨宮博士には，データフロー・アーキテクチャからの意見を発表していただく。

２番目のパネリストは Dr. David May，ケンブリッジ大学を卒業後，英インモス社に勤務されている。VLSI システムのアーキテクチャおよび並列処理言語の観点から意見を述べられる。

３番目のパネリストは Dr. Ehud Shapiro，エール大学でコンピュータ・サイエンスの博士号を授与され，現在，イスラエルのワイツマン科学研究所で研究を続けられている。

同博士は，逐次処理と並列処理の関連性について発表される。

４番目のパネリストは，古川康一氏，東京大学より情報科学の博士号を取得され電子総合研究所に入所，現在は，ICOT の第五世代プロジェクトの研究リーダーの一人で，新しいプログラミング言語という観点から話をされる。

最後のパネリストは，Prof. Salvator. J. Stolfo. ニューヨーク大学 Courant 研究所コンピュータ科学の博士号を取得，現在，コロンビア大学で教鞭をとっておられる。エキスパートシステム用の大型並列処理という観点から意見を発表される。

パネル・ディスカッションに移る前に，簡単に本セッションの進行について説明したい。まず，各パネリストが，約15分間，それぞれ専門の立場から意見を発表した後，会場からの質問に応じることとしたい。その後，30～40分間の一般討議を行う。では，雨宮博士から発表をどうぞ。

雨宮：最初に，データフロー概念の観点から，並列処理とそのマシン・アーキテクチャについて述べたい。しかし，具体的なマシン・アーキテクチャというよりも，むしろ抽象的な問題を取りあげることになる。まず，並列マシン・アーキテクチャについて２種類のアプローチが考えられる。１つはタスク指向またはアルゴリズム指向のアプローチ，もう１つは包括的なアプローチである。タスク指向のアプローチとしては，例えば，イメ

ージ処理や偏微分方程式の計算などがある。この方法ではアレイ構造とかメッシュ構造といったレギュラーな構造をもったアルゴリズムを用いている。これは，どちらかといえば，実用工学の側面であると思う。しかし，ここでは包括的アプローチである第2のアプローチについて述べる。これは，コンピュータ・サイエンス的観点からのアプローチである。即ち，まだ十分に整されていない不定の構造または柔軟で動的な構造を対象として考える。例えば，AIプログラムや社会現象のような自然，人工的現象のシミュレーションが挙げられる。

第2のアプローチの，難しいが，しかし，重要な問題は，いかにしてプログラム構造をハードウェア構造へマッピングするかということである。これは，並列処理マシンのアーキテクチャへのアプローチ上の基本的な問題である。

プログラム構造ということを考えてみると各プログラム領域には多種に及ぶパラレリズムが存在すると思う。プログラム領域におけるパラレリズムの特長は2種類に分類でき，実用的なプログラムはこの2種を混合したものである。

我々は関数型あるいは論理型プログラミングの概念に基づいてプログラミングを行うのが最もよいと考えるが，関数型あるいは論理型プログラムにおけるパラレリズムについて考えてみよう。

第1種目の型は，並列演算である。それは，分割統治・アルゴリズムによるものである。

例えば，身近なアルゴリズムとしてクイックソートやマージソート・アルゴリズムがある。我々は，並列処理によってリストデータ構造を処理するプログラムの実行時間をリニアタイムにすることができる。論理的推論を行うプログラムの領域においては，OR並列がこの種の並列である。第2のタイプとは，ストリーム処理またはパイプライン処理における並列性である。この種のパラレリズムはデータ構造に対する線形再帰によって成し遂げられる。典型的な例としては，シーブ法による素数計算や集合演算がある。そして論理型プログラムにおいては，ANDパラレリズムがこのタイプである。一般のプログラムは，以上2種類のパラレリズムの混合物である。

次に，情報処理の特長を考えてみよう。3種類の特長が挙げられる。第1として，インプット・データからアウトプット・データへの写像を意味するデータ上の操作である。これは，関数型プログラムの特質であり，ここではデータの流れが予め決定されている。即ち，固定されたデータフローである。第2のタイプとは，リレーショナル・データベース等のようなデータ検索である。これはデータ間の関係によって特長づけられる。このタイプでは，データフローは前もって決定されず，双方向性を有している。これは論理型プログラムの特質でもある。

第3のタイプは時間依存型のプロセスである。関数型プログラムも論理型プログラムも，そのままではこの時間依存型プロセスには適合しない。しかし，この履歴依存性の概念が実際の情報処理においては重要である。

すなわち，実際的な情報処理では，状態及び逐時オペレーションの処理を欠かせない。コンカレントプログラミングの概念は，実際において，時間依存型プロセスに対処する上で必要である。

結局，マシン，及びプログラミング言語は以上3つの情報処理の特性をうまく実現しなければならない。それは関数型プログラミングと論理型プログラミングがモジュール化の概念を用いて統合されなければならないということである。

このためには，各プログラムモジュールがメッセージを介して協調してひとつの問題を解く協調型プログラミング，つまりメッセージフローシステムが重要な概念だと思われる。

もう一つのパラレリズムに関して重要な点は集合に関する概念である。AIプログラムにおいて基本的概念は生成とテストということである。まず最初に与えられた述語に適合するデータの集合

が生成プロセスによって作り出され，次に，この生成データが別の述語に合うかどうかがテストされる。これらのプロセスにおいては，述語は生成あるいはテストされるデータの間の関係を定めている。次に重要な点は，協調型のプログラム構

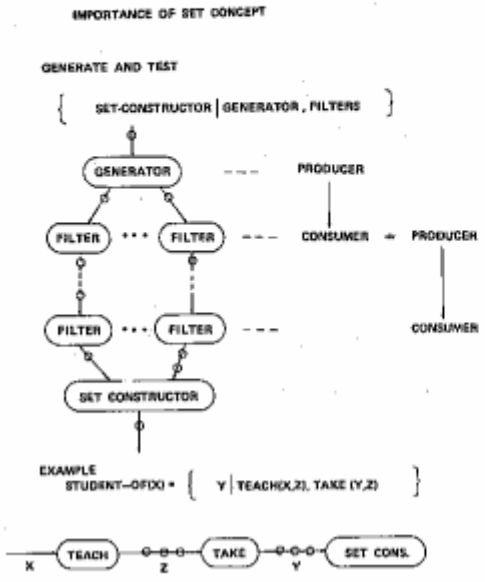

図 1

造がプログラムの階層構造化構成とモジュール化構成を可能とするということである。これはモジュール間の並行実行をもたらす。

では，集合概念の重要性を示す簡単な例を挙げてみよう。これは集合を与える式である。

まず初めに，ジェネレータがあるデータ集合を作りだし，フィルターにこの生成データを通過させる。そして最後に，セットコンストラクタで，データの集合を組みたてる。この処理法の特性は，ストリーム指向型計算である。簡単な例として，Xを教師とするとき，X生徒Yという概念について考えてみよう。Xがカリキュラム Z を教えているとすると生徒Yはカリキュラム Z を受けている者の集合として定義される。

では，こうしたパラレリズムの実現方法を考えてみよう。私の観点では，データフロー概念は，並列演算を制御するための基本概念である。第2に，構造データ操作もまた，重要な問題である。例えば，リスト構造は，複雑なデータ構造を表現するのに非常に便利な方法である。しかし一方，複雑なメモリ・オペレーションを必要とするため，こうした構造データ操作には多くの処理時間を費す。したがって，パターン・マッチングのような高度なメモリ・オペレーションは，ハードウエアによって支えられなければならない。データフロー・アーキテクチャは，パケット・コミュニケーションをベースとするので，メモリーユニット上のメモリ・オペレーションとプロセッサ上の実行制御との間で，パイプライン実行が可能となる。このようにデータフロー・アーキテクチャによってパイプライン制御の効果を十分引き出すことができ，メモリー操作における処理時間問題を解決することができる。第3に，プロセッサ間のコミュニケーションの問題がある。並列処理マシンにおいては大量のコンピューティングモジュールがネットワークで結合される。したがって，コミュニケーションオーバーヘッドは深刻な問題でありパケットスイッチングネットワーク構成法は不可欠な技術となる。パケットスイッチングネットワーク及びパケット・コミニュニケーション技術は，こうしたコミュニケーション・オーバーヘッドを解決するであろう。なぜなら内部の演算処理と外界との通信制御の間の処理のパイプライン化をパケットコミュニケーション技術により実現することができるからである。このようにして，データフロー・アーキテクチャは，プロセッサー間コミュニケーションとプロセッサー内処理との間のギャップを解消することができる。

前にも指摘したとおりデータフロー・アーキテクチャは，並列マシン・アーキテクチャとして極めて優れた特質をもっている。しかしながら，実用的なデータフローマシンはまだ存在していない。解決すべき問題は何であろうか？パラレル・マシン・アーキテクチャに関する研究上の問題として次のものがあげられる。一つはデータフローにお

いて並列処理の対象となる粒の大きさ(granularity粒度)の問題である。適切な粒度を定義することが必要であるが，これは応用に依存する問題であろう。しかし，一般にはプログラミング言語の特性が，データフロー・アーキテクチャに対する最適な粒度を決定することになる。この最適粒度を定めるにはコンパイル段階での処理スケジュールが必然的となり，そのための技術としてプログラム変換技術が考えられる。

第2に，負荷バランスをとるためのタスクスケジュールの問題がある。つまり，多数のプロセッサ間において，システムへ負荷をいかに均等に配分するかということである。第3にハードウェアの実現化技術の問題がある。データ駆動プロセッサをいかに安く実現するか，高度なオペレーションを支える構造メモリをいかに実現するか及びパケットスイッチング・ネットワークをいかに実現するかである。ネットワークの実現化においておこる膨大な結線の問題も解決されなければならない。そしてVLSI技術はこれらハードウェアの実現化の問題を解決の方向へ導くであろう。第4は，並列推論メカニズムの実現に関する問題である。データフロー概念がAIプログラムにおける生成・テストスキームを支えるものであることは先程述べた。つまり並列推論マシンを支えるには，高度なデータフロー制御が必要だということである。最後に協調型問題解決システムの実現化であるが，これは，関数型及び論理型プログラミングパラダイムに基づくモジュール概念の現実化である。すなわち，我々は，データフロー概念をより高度化したメッセージフロー概念へと進むべきである。これが私の立場である。

座長：では次に，Dr.Mayにお願いする。

May：私は，まず言語及び高度なコンカレントシステムについて述べようと思う。私は，今まで扱ってきた多くの実際問題を述べるためには，逐次型とその状態，コンカレンシィとコミュニケーション，分類体系とローカリティという一連のものを含む言語が必要であると信じる。これらは，現実の世界において，観察できることである。これらは応用分野において必要不可決であり，またインプリメンテーションの構造及び行動を表すのに必要である。言いかえれば，このような特長を持つ言語は現実の問題を直接VLSIインプリメンテーションへ写像する手助けをする。

私は、これらの概念は，明快に言語に表現されるべきであり，ある意味では，言語における構造と直接対応していなければならないと思う。

OCCAMという,私が精通している言語についての例をあげてみよう。いったん私たちがこのような言語を有することになれば，プログラマは，プログラムを組むことのできるコンピュータの集合かまたはある目的用のVLSIハードウェアの構造や動きを記述する手段を持つことになる。

コンカレントシステムに用いられる言語にそなわる重要な特長は，一つにはフォーマルな性質を有しているかどうかである。シミュレーションと検証からその正しさを確立するのが大変困難であるので，コンカレントプログラムの性質を試験し証明する必要があると思う。また，プログラムの逐次翻訳と並列翻訳の間で移動ができ，また，知識にもこのような変換が行なえたらと考える機会が多い。このようなプログラムにプログラムの変換を行なえることができたらと思う。そしてそれはエラーを導かず，この2言語は共通の動きを保有している。私たちはこの動きを，プログラムを実行する機械を観察し記述としてみなすことで行える。この方法によって論理を，プログラムを意味付けすることのできる構造体系にすることが可能であり，論理型プログラミングなしで論理的なプログラミングを行えるようになる。

さて，問題になっている論理型プログラミング言語と記述言語は，記述言語を並列化することで行なえると語られてきている。しかし，私は並列化できないために，稼動できる全ての能力を使用できずにいると思っている。並列化までに及びも

つかないので，タスク処理の割り付けの問題が起きている。今の段階ではこれらの記述言語が実行できるものであるかどうか確信できない。

私はすでに，私にとっては重要課題とも思えるローカリティについての問題点をとりあげた。処理と通信の問題を克服しようと思うならば，私達は，操作ができるだけ局部に限られる処理プロセッサを作るよう試みるべきであろう。

これは，恐らくプロセッサを，非常に速く呼び出しのできるローカルメモリーと結合させることによってより完全に成し得るはずである。また同じシリコンチップ上にこの2つを備えることによって申し分なく達成されるであろう。

次に，私たちは通信のローカリティが必要であり，これを実現する最上の方法は，何組かのプロセッサの間でのみ通信を行うことである。

この方法でなら，特に，処理メモリ（Processes memories）や通信システムを共有しないですむ。

ローカリティの問題は，コンピュータ・アーキテクチャでは新しいものではなく，実際，過去多く，の研究成果がこれから生まれている。例えば，ベース・レジスタ，ポインターやキャッシュ等があるが，このレジスタは，いつもコンピュータ・システムの性能を上げるためにローカリティを使用している。

この種の，基本的には完全なローカル操作を使用したアーキテクチャは，世界的なバスクロックや行為者に同時性をもたらすことはなく，若干，面白い問題を提起している。私が言語の考案者に申しあげたいのは，現在，配置されているシステムにおいては，ほとんど実行不可能ではあるが，言語に便利なプログラミングの特徴を採り入れるのは非常に容易であるということである。というのは，言語には必要な世界的協力や同時性を内在的に必要としているからである。私のにがい経験から勧めるのだが，言語ははじめに，実行はかなり能率的であることを確信して考案されるべきだと思う。というのは，そうすると逐次処理で実行させることが非常に容易だからである。シーケンシャルプロセッサ・シミュレーションによりほぼ全ての事柄を実行することができる。

現在よく尋ねられるのは，処理メモリと通信間のシステムにおけるよい比率はどれくらいかということである。私たちの VLSI 技術の現状では1千万の命令が同じ量のシリコン領域で実行されるような高速のプロセッサは2キロバイトのメモリや，通信システムが残りのシステムに結合される，という状態にある。

これは，私たちはかなり面白い選択をすることができるという意味で，つまり4メガバイトメモリを用いて10 MIPS のプロセッサをつくるのにシリコンを消費できるということである。これは，非常に型にはまった種類のコンピュタになるか，または，1秒間にたった2メガバイトメモリで1千億個の命令を行えるコンピュタができるということである。この2つの装置はほぼ同様で，どちらも約1000 VLSI 装置を用いることになり

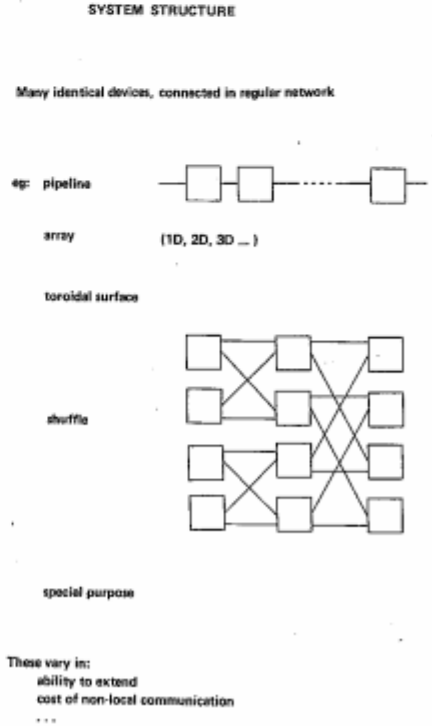

図 2

両方とも非常に小型のコンピュタになる。したがって，実際の問題に適用できる方法で，少量のメモリを分散させながら多くの処理要素を含むシステムを現実にどのようにして作成するかが問題になる訳である。

では，私たちはどのようにしてシステムを作るのだろうか。明らかに，このようなシステムは極めて規則的でなくてはならない。もし何千ものプロセッサを有することになれば混乱してプロセスを結合することはほとんどなくなるが現実には多くはない。

私たちは，1次元，2次元，3次元またはそれ以上のアレイでのパイプラインのような構造を作成できる。今いったように6次元以上になると配線が複雑になり作成はほとんど不可能であると言及した。

表面上は，もっと風変わりな構造もありえる。例えば，2次元的な配列を行なって，トーウスを作るためにそれを折り曲げる。

そして合成による変換のような構造のものがいくつかあり，このうち大部分はわずかに異なった構造をもって結合している。この配列は，当然，私がすでに示したより，かなり多くの段階を経て広がり，特定の目的に使用されるために作られた装置がある。これらの混合はかなり拡張が困難となり，そのせいで段階が増すにつれ配線もますます拡張が困難なものとなる。しかし，プロセッサの1000や2000は実行可能である。

構造は，他の要素でも変化する。つまり非ローカル通信のコストである。隣接していない別のプロセッサと通信しかけるプロセッサのことを意味している。当然，パイプラインは多くの中間点を通過しなくてはならない通信が含まれるので，特に好ましくない。一般の要求からは，ネットワーク内を通信する合成が最適である。従って，一般的な目的に使用する機械としてはかなりよい構造であると仮定される。このような機械では，写映における問題をとり除くことは非常に解決困難になる。というのも問題を取り除くことをよぎなくしてしまうからである。

本当のところ，取り得る解決方法はあまりない。これは将来の研究課題となるであろう。私たちはこのようなマシンのための現在のアルゴリズムにどう対処すべきなのか？若干は，特に信号処理，数値処理，そしてシミュレーション領域に存在している。これには規則的な配列のあるプロセッサに容易に写映される規則性のある問題である。このような場合での最適の結果は，普通，逐次型と並列アルゴリズムを結合することで得られる。このことが，逐次型と並列型のプログラミング言語能力の両方を併せもったプログラミング言語をもつのが特に重要であると感じている理由の1つなのである。処理と通信の間の適切な比率の選択が特に重要になる。これが，どれだけ内部のオペレーションが，時間毎に，メッセージを受けとるプロセッサによって実行されるのか，という理由である。スペクトラムの末端に，単一の操作でメッセージをインプットし，アウトプットする。データフローを効果的に有することになる。スペクトラムのもう一方には，巨大な量のデータをインプットする逐次型プログラミングがある。多数の逐次処理が存在し，アウトプットおよび結果がでてくるのである。しかしながら，実際には，一般のコミュニケーションが処理のコストとの関連において消滅してしまうこの比率を作り出すアルゴリズムに適応するのは可能であるように思われる。

したがって，結論としては，コンカレントプログラムはちがうものでも，困難なものではない，ということになる。

座長：ありがとう。では次に Dr. Shapiro にお願いする。

D Shapiro： Dr. May の話の続きになると思う。並列処理コンピュータの開発は，逐次型のコンピュータの開発よりも容易でないがそれは，また，逐次型コンピュータの発展よりは困難であってはならないということも意味する。私の話は，ある

人にとっては明白であり，ある人にとっては風変りであるかもしれないが，どのようなディスカッションの方向にもっていくかみてみよう。

しかし，少なくとも，逐次型コンピュータのように並列型コンピュータの開発に40年もかかるという意味ではない。しかし，少なくとも，逐次型コンピュータにおいて経験したような，大変革を我々も乗りこえねばならない。それは，まさに，誕生する前から，子供は人類としての進化をめまぐるしく辿りながら成長していくという昔からの理論と同じである。新生児である並列型コンピュータは，逐次型コンピュータと同じ変革を経ていくだろうが，そのタイムスケールとしては短くなろう。具体的には，まずコンピュータが完成し，それからマシン言語を得るということだ。コンピュータが誕生したての頃のように，専門家のみがそのマシン言語の使い方を知っているかもしれないが，こうした新言語で新しいコンピュータを利用しながら，経験を積んだ上で，ようやく，これらのコンピュータ用の高度な言語の設計方法を理解できよう。

こうしたマシン言語を使って，広範な経験を積み重ねていけば，高度な言語をマシン言語へと編集していく方法もわかるようになろう。いかにして高度な言語を逐次型コンピュータ言語に編集できるかを理解するのにいかに長い年月を要したかを銘記していただきたい。実際に秀れたコンピュータ言語コードを生み出すプログラムを知るまでにコンピュータ言語のプログラミングにおいてあらゆる経験を積まなければならなかった。並列型コンピュータに関して，従来のコンピュータと異なることが起きるとは思われない。またアリゴリズムにおいてもノイマン式コンピュータと同じような経験をするであろう。多くのアルゴリズムが飛躍的に成長しはじめている。

アリゴリズムにかかわらず，並列型コンピュータに取りかかれば，開発は急速に進むだろう。

それでは，開発の現段階で必要なものは何であろう。出発点として必須不可欠な3つの要素である。すぐれた抽象マシンがまず必要である。並列型コンピュータのプログラミングを行なっているという事実は無視できない。適切なプログラムができあがってこそ，並列型コンピュータを順調に運転できるということを忘れてはならない。並列型コンピュータのモデルを頭に浮かべなくてはならないが，低レベルすぎても，複雑すぎてもいけない。

十分に抽象化されているならば，我々は，明確な概念を描きだせる。それを同時に，十分に，具体的アーキテクチャに近くならなければならない。つまり，概念と実際の学習プログラムに大きな隔りがあってはならない。

次に必要なものはすぐれたマシン言語である。前にも述べたように，しばらくの間は，マシン言語で並列型コンピュータ・プログラムできなければならない。我々が高度な言語を編集するまでには十分な経験を積むために2～4年の歳月を要すると思われる。だからこそマシン言語が高度であることが重要なのだ。今，我々が実現しつつあるコンピュータは，まだ人間の手で操作できるものだ。但し，研究に従事している専門家だけが扱い得るのだが。第3に必要なものは，前にも述べたように，アルゴリズムである。

少なくとも，この開発の第1段階において無視してならないものは，抽象マシンにおいてのコミュニケーション・ローカリティの明確なコントロールである。Dr. May は実際のマシンのシステム設計におけるローカリティと規則性について，強調していた。そしてプログラムを作成しようとする抽象マシンにおいてもこの観点は無視できない。さもなくば，具体型マシンと抽象型マシンとの隔りは広がり，実際に抽象型マシンをシミュレートする時には，効率を失うことになる。したがって，抽象型マシンの定義において，ローカリティを無視できない。マシン言語では，明白なパラレリズムを無視できない。Dr. May が前に述べたように

並列性の研究のためにコンピュータについて何もしらずに，高性能のコンパイラやランタイムシステムなどに頼ってプログラミングできる程十分な知識を持っているとは思えない。

我々は，まだ，学習段階にあり，このことは無視できない問題である。そして，もう一つ重要かつあきらかなことは，アルゴリズムの設計におけるローカリティと並列性の両方を無視できないということである。

従って，われわれが抽象型マシンの制御とプログラミングに使おうとするマシン語は，コミュニケーション・ローカリティと並列性の両方を明示的に制御できるものでなければならない。
Dr. May は，このことを別の言葉で話されていた。この点で私は Dr. May と同じ意見だが，宣言型言語にはこの能力がない，というかれの結論には同意しかねる。具体的に述べると，例えば，明示的なマッピング記法（notation）で強化された Concurrent Prolog はローカリティとパラレリズムの両方を明示的に制御できると思う。

現在の段階が何年続くのか－1年か，2年か，3年か，5年か ── は分からないが，とにかく今の段階でパラレリズムを自動的に抽出しようとするのは，時機尚早だと思う。まだ十分に分っていないからで，また，これと同じ理由から，高級言語をマシン語にコンパイルすることも，同様に時機尚早だと思う。私の考えている高級言語とマシン語は，従来のものとはかなり違っていると思う。プログラミング言語のロジックの特定のサブセットやクラスも，マシン語とみなせる。そして，私のいう「高級言語」は，このマシン語よりも高級な言語を意味する。つまり，並列コンピュータ用のマシン語と高級言語の最も大きな違いは，パラレリズムとローカリティを明示的に制御できるかどうか，という点であると思う。マシン語の特徴はシンタックスでもなければ，関数型，論理型，ＣＳＰ等のタイプでもない。マシン語の特徴は，コミュニケーション・ローカリティとパラレリズムを明示的に制御するその能力にあると思う。この2つを明示的に制御できるのであれば，マシン語 ── 使用に耐えるマシン語 ── とみなすべきと思う。

疑問点がさらに絞られてきたと思う。そのようなマシン語はマシンの実際のパワーを利用するには高級すぎるだろうか？　システム・プログラマにとってあるいは低級すぎるのだろうか？　これらは，マシン語のチューニングに関する現時点の問題である。しかし，ローカリティとパラレリズムを明示的に制御する力をマシンが備えているのなら，この枠組みの中では，マシン語とみなすべきであろう。われわれはここから出発するべきである。並列コンピュータのアセンブラ言語やマシン語（例えば，Concurrent Prolog やそのサブセット）によるプログラミングのやり方についてかなり分ったのだから，今度は，これらの点について明示的な制御をしないで，より高級な言語を目指すべきだと思う。

これらの問題に関する私の見解は，いささか楽観的かも知れない。しかし，私は，プログラミング・スタイルや高級言語，そしてアルゴリズムは，このような素晴らしいマシン・モデルが利用可能になれば直ちに出現してくる，と確信している。

もう一つ私が確信していることは，この発展の過程で，従来のコンピュータ科学の重要な概念のほとんど，もしくはすべてが再発見されるに違いないということである。「次世代のコンピュータは，まったく異なったものとなるはずだから，アルゴリズムや構造化プログラミング等の従来のコンピュータ科学の重要概念は突然不要になる」とか，「論理型プログラミングによりコンピュータ科学はごみ箱に捨てられるはめになる」と考えるのは誤りである。このようなことは，発展の過程では起こらない。

発展の過程で私が予想しているのは，コンピュータ科学の重要な概念の中で特定のマシンに依存しない概念のほとんどすべては次世代のコンピュ

ータに引き継がれ，そして恐らくは別の枠組みの中で再発見される，ということである。例えば，これは並列プログラミングで経験したことであるが，データ構造とプロセス構造が似ているということである。データ構造は逐次型アルゴリズムのデザインとインプリメンテーションで重要な役割を果たす。私の考えでは，並列プログラミングにもこれと似た概念があり，それがプロセス構造である。プロセス構造は新しい概念だが，逐次プログラミングのデータ構造と非常に似ている。ほとんど同一といっても良いくらいである。

今お話しした例のように，古い概念が新しい枠組の中で再発見されていくと私は予想している。

座長：では，次に古川さんの意見をお願いしたい。

古川：私が述べたいことは，もう少しアプリケーション寄りの話と，知識情報処理一般におけるパラレリズムについてである。これは非常に難しい研究テーマだと思う。どんな方法をとったらこのテーマを達成できるかは，今のところまだ分かっていない。このテーマに取り組むためには，次の四点が必要だと思う。研究の対象とすべき重要な問題を見つけ出すこと，研究を遂行するための適切なツールを用意すること，系統的な研究方法を設定すること，そしてたくさんの人材を確保することの四つである。

これらを一つにまとめると，「重要な問題は，たくさんの研究者が適切なツールを使って展開する系統的な研究活動の過程で見つけ出される」となる。私が主張したいのは ― これは，これまで主張してきたことであるが ― 論理型プログラミングは統合のための接着剤として非常に重要な役割を果たす，ということである。この点については，後でまた触れることにする。

論理型プログラミングは，知識情報処理と超並列コンピュータ・アーキテクチャの橋渡しをするという点で非常に重要である。知識情報処理と論理型プログラミングの間では，並列プログラミングが非常に重要である。そして，論理型プログラミングと超並列アーキテクチャの間では，計算モデルが重要である。

系統的なアプローチで必要なのは，ハードウェアからアプリケーションまでを引っくるめた形の総合的な研究である。様々な研究活動をすべて展開することが必要である。場合によっては，低レベル側か，高レベル側を調べるだけで並列計算を実現できることもあるであろう。しかし，システム全体を扱うことは非常に困難である。たくさんのパラレリズムをアプリケーションから抽出する方法が必要となるし，また，そのようなアルゴリズムを表現する効率的な言語や，それらを効率良く実行できる並列型コンピュータも必要である。

現在の段階は，この非常に困難な問題の研究に取り組む体制が整った段階，といえると思う。

この問題の研究に必要な優れたツールが揃った。ハードウェアについては，VLSI が使えるし，並列推論マシンはまだ手に入っていないがこれも可能である。ソフトウェアについては，記号計算用の並列型言語が使える。また，アプリケーション分野については，分散問題解決用の知識プログラミング言語がいくつか揃っていてこれらは，使用可能なツールの１例である。ソフトウェアについては，並列言語として，Concurrent Prolog があり，またその拡張としてKL１を設計中である。アプリケーションについては，actor and contract net などの提案がすでになされているが，われわれは Mandala という名前の言語を提案中である。そして実際，この非常に困難な問題と取り組むには，たくさんの研究者が必要と思う。逐次型のコンピュータ文化がここまで発展を遂げるのに，一体どれだけの研究努力が払われたかを，比較の意味で調べてみるべきであろう。また，アルゴリズムの逐次型アルゴリズムのことだが，研究が，逐次型コンピュータの出現直後から急速に進展した点についても，注目してみる必要がある。つまり，適切なツールが揃っていて研究方法も確

立しているなら，並列型コンピュータの文化を短期間で築くのも不可能ではない。という点である。これが私の結論である。

座長：では，次にProf. Stolfoの意見をお願いしたい。

Stolfo：これまでのところは随分おとなしい論調だったので，議論が少し活気づくような話をしたい。大部分は天の邪鬼的立場から述べてみる。哲学的なこともしゃべるが，そのすべてを信じているわけでは必ずしもない。いずれにせよ，何らかの議論に発展すると思う。

まず最初に，過去3年間の私の活動について，かいつまんでお話ししよう。私はAT & Tのベル研究所の技術スタッフとともにエキスパート・システムを開発した。これは商品化され，現在，AT & Tにより販売されている。われわれはこのプロジェクトにおいて，調査・研究から製品開発に至るまでを担当した。(現在は，AT & T 関係の仕事はしていない。)私は，またアークテクトとして，エキスパート・システムを高速かつ経済的に実行する並列型コンピュータの設計に携っている。この実験機は1983年の4月から稼動している。まもなく，もっと大型の実験機が完成する予定である。

パネルというのは論戦を繰り広げる場であるべきだと考える。その意味で議論の火付け役になるような話をしたいと思う。「LISPやプロダクション・システムをベースとするこれまでの人工知能（AI）技術には，まだまだ未来がある。その応用はやっとその緒についたばかりだ」と今だに信じている世界からやってきた私のような者には，既存の技術を最大限に活用することもなくPrologに飛びつくのはいささか性急であると感じられる。そこで，私は，LISPの内部に埋め込まれたプロダクション・システム言語について述べる。先に触れたACEエキスパート・システムは，そのようなプログラムの1例だ。これらのプログラミング・フォーマリズムを使って研究中の米国のたくさんの研究者達と一緒に，私は是非ともこれらをスピードアップしたいと考えている。

このスピードアップには理由がある。第5世代コンピュータをスピードアップすべき理由については私はまだ知らないが，プロダクション・システムをスピードアップすべき理由は良く知っている。専門用語に通じていない人にとって，プロダクション・システムとは何だろうか？。皆さんはルール・ベース・システムについてお聞きになったことがあるかも知れないが私が頭に描いているフォーマリズムはもっとOPSに近いものである。

プロダクション・システムは，主に3つの要素から構成される。ファクトを格納するワーキング・メモリ，if-thenルールがたくさん集ったものであるプロダクション・メモリ，それにインタープリタの3つである。このインタープリタの処理は非常に簡単で，ワーキング・メモリの状態と一致するような左辺を持つルールを全部見つけ出し，その中から1つ選んでその右辺 ── これはワーキング・メモリへのファクトの追加か削除であることが多いのだが ── を操作することである。非常に簡単な問題解決フォーマリズムであるが，かなり有効である。

このフォーマリズムでは，非常に限定された形式のユニフィケーション・パターン・マッチングをデータ構造に対し行なう。この種のフォーマリズムには，カーネギーメロン大OPSスタンフォード大のEmycin等，たくさんの例がある。そして現在，米国では，いくつかのAI企業が，これらと似たプログラミングフォーマリズムを商品としてエキスパート・プログラムの開発用に販売している。ここで，われわれが検討してきた問題を1つ，数分間で話したいと思う。それは，「どうしたらプロダクションシステム・プログラムを高速化できるか？」という問題である。この問題について少々技術的なお話をしてから，もっと哲学的な問題に移りたいと思う。

まず，インタープリタが実行する処理を，ルールの発見，ルールの選択，ルールの右辺に対する

操作の3つのフェーズに分けて考えてみよう。発見のフェーズにおけるパラレリズムは何だろうか。

このフォーマリズムを見てまず思い浮かぶことは，ルールを並列に照合することだと思う。逐次型マシン上で逐次的に照合するのではなく。それぞれのルールについては，プロセッサへわりあて他のルールとは独立して照合させる。しかし，この照合フェーズには，他にもパラレリズムのソースがある。各ルールの左辺はパターン要素の集合になっている。

できれば単一パターン要素をワーキング・メモリと照合したい。ワーキング・メモリでは，一連のプロセッサに対しワーキング・メモリを分配する。

ルールがたくさんあり，パターンもワーキング・メモリもたくさんある場合には，当然ながらたくさんのプロセッサが必要になる。そして，たくさんのプロセッサが必要な場合には，プロセッサが効率良く働くようにインプリメントすることが必要である。照合フェーズにおけるパラレリズムの重要性は，？1クラスのプロダクション・システムについて調べてみたが OPS フォーマリズムによると90%の時間が照合フェーズで費されていた。であるから，このプロセスを高速化できれば，うまく行く。このプロセスにかかる時間は，ワーキング・メモリとプロダクション・メモリの大きさに大体において依存する。

予想される効果，？これは将来の長期目標として明言しておくべきことであるが，それはこの時間を一定時間まで短縮することである。ルールの照合にかかる時間は，プロダクション・メモリとワーキング・メモリの大きさとは関係がなくなることが目標である。

インプリメンテーションについては，これらのプログラムを実行するアルゴリズムを実現するために，われわれは今，DADO というマシンを作っている。

このマシンは一言でいうなら，中規模から小規模の処理要素をたくさん集めたものである。「中規模」とは，8ビットないしは16ビットないしは32ビットのプロセッサを意味し（各々に）数千バイト程度のメモリが付いていて，プロセッサ同士はバイナリ・ツリー構造の高速ネットワークで接続される。

今後の問題は，？ granularity の問題は，アプリケーション・プログラムを検討しない限り決定できない。処理要素の規模はどの程度にすべきであるか？

他には，？2番丁のフェーズ，既ち選択フェーズが考えられる。しかし，これは照合ほど重要でないことが分かる。これは対数時間特にバイナリ・ツリー構造上では解くことができる。しかし，実質的な高速化は達成されない。実質的な高速化が可能なのは，照合フェーズと，3番目のフェーズ，即ちルールの右辺に対する操作フェーズである。

操作フェーズでは，パラレリズムのソースは2つある。1つは複数のルールを並列に操作することで，これは複数選択（multiple selection）に多少似ている。しかし，単一のオペレーションとして実際には実行しないルールを選択するためである。難しい部分は，一群のルールがある場合は，並列に適用したくなるということで，扱い難い問題である。しかし，これがパラレリズムのソースの1つである。もう1つのソースは，ルールの右辺のワーキング・メモリを大幅に変更できるように，プロダクション・システムから離脱して一時的冗長性をもつプロダクションを作ることである。これを並列にインプリメントできれば，性能を大幅に変更できる。従って，これら2つの複数ルールfiringの問題も重要であり，またワーキング・メモリへの大幅の変化も重要である。

昨年のことだが，電電公社から石田亨氏が来訪され，この問題を OPS 式プロダクション・システムについてわれわれと一緒に検討した。石田氏は，プロダクション・システムのルールを記述する方法を適度に緩和させると6～10ルールが可

能なことを発見している。これは，平均リサイクルで6～10個のルールを fire 可能なことを意味する。悪くはない。

この DADO マシンのインプリメンテーションとその方法は，ルールを分配して同期を最小にすることである。今後の問題は，どのようにルール・ベースを再構成したら，フォーマリズムによる逐次性を取り除けるか，ということである。

最後にワーキング・メモリを並列に操作することを考えたい。ワーキング・メモリを大型のデータベースとして扱い大規模な変更を行なえば，かなりの性能の改善が可能で，この使用に成功したシステムの例には事欠がない。たとえば，ACE システムのような大規模のデータベースの応用に応用の場合などがある。ACEは，現在VAX 780 上で走っている。走っている際中は，誰も他の人間は780 に近づけない。タスクの完了までに何時間もかかる。

これは本格的な AI プログラムである。われわれの研究から予想すると，780 より小型の DADO Ⅱマシンでこれを実行した場合，数分ですむ。

インプリメンテーションについては，ワーキング・メモリの分配，ワーキング・メモリの並列処理，そして今後の問題は再び，PE の粒度(granularity) 問題である。 そしてまた，どうやってワーキング・メモリを分配するのであろうか？

これで私の話の技術的部分を終えて，次に哲学的部分に入ろう。過去5年間，プロダクション・システムからわれわれが学んだことは，現在，フォーマリズムは当然ながら逐次型プログラミング環境で考案されたものであり，このフォーマリズムを使うとかなりのプログラミングでの逐次化が避けられないということである。プロダクション・システムのような，かなりのパラレリズムを生得的に備えた純粋に宣言的なフォーマリズムを思いつくかも知れないが，これらのフォーマリズムは並列的な思考には向いていない。ここに教訓がある。実際，長年に渡って逐次型マシンでプログラミングしてきた人から渡されるフォーマリズムは，並列型で考えることができたであろう問題をシリアルに扱かっている。彼等は依然として逐次的に考え，逐次的にプログラミングしている。もっと表現力に優れたフォーマリズムを実験するために，できるだけ速く実用的な並列型プロセッサを完成しなければならない。次にわれわれの現状を手短かに報告しよう。

われわれは DADO というアーキテクチャを開発している。DADO Ⅰは15個の処理要素からなるマシンで，1983 年の4月から稼動している。まもなく完成する DADO Ⅱは 1023 個の プロセッサから構成されるマシンで，ACE をインプリメントする予定である。できれば DEC 社からのR1プログラムを使いたいと考えている。また，われわれは，OPS を凌ぐもっと表現力の高いフォーマリズムのプロダクション・システムをインプリメントしている。また，当然ながら，論理型プログラミングも相変わらず手がけている。これには，今週の始めに話題にのぼった LPS システムを使っている。そして最終的には，その後DADO Ⅱより PE の数を増やし，VHSIC か VLSI でインプリメントしたいと考えている。それではなぜこれはあなたにとって重要なのか？ 私が今いいたいことは哲学的な忠告であって非難では決してない。

この会場にお集まりになられた皆様の中で何人の方が次のような条件を備えた実世界のエキスパート・システムをインプリメントした経験を持っているであろうか？ 即ち，重要なタスクあるいは役に立つタスクを実行し，何らかの仕事を楽にして生産性を向上し，コスト削減が可能なことを立証し，利益を上げ，市場を発見するエキスパート・システムである。

われわれは，これらの制約に応えられるわけではない。別の言い方をするなら，第5世代コンピュータは，問題の解を探すのであろうか？ 問題は何であろうか？

第５世代コンピュータはなぜ高速でなければいけないか。今のように処理の遅いプログラムは役たたないと言えるのだろうか。遅いエキスパートシステムでは実用的でないのか。それはわからないと思う。第５世代コンピュータにむけて飛躍するためには，今のプログラマと研究者をまず再教育しなければならない。と同時にユーザーをも教育しなければならない。なぜならユーザーこそ第５世代と生きていかなければならないからである。かれらは AI がなんであるか，なにを意味しているか全然わかっていない。

まだ我々にも何ができるかわかっていないことを処理する高速コンピュータを作る前に，まずその市場を定義しなければならないが，いったいどうしたらできるのだろうか。我々は，まずアプリケーションを作り，市場を決めなければいけない。それからハードウエアを作り，特定のフォーマリズムの中で実際にプログラムを作成せずに，そのフォーマリズム用の高速プロセッサを考えるべきであろう。まず，多量に売れそうで，現実に使われるようなエキスパートシステムを作成する努力をすべきである。ここでいう現実的な使われ方とはIBMの大型コンピュータでデータベースを使っているデータ処理産業のユーザに使われるという意味である。

彼等は，どのようにエキスパート・システムを使うのであろうか？　エキスパート・システムは彼等にとってどんな役に立つのであろうか？　これらのプログラムをよくしスピードをあげ，使い方をよくするための必要条件を見い出し規定し，その後で第５世代コンピュータを発明する。必要は依然として発明の母であるが，必要が何であるかを私は知らない。であるから，必要を規定しなければならない。

座長：各パネリストの立場の発表が終わったので，次に質疑応答に移りたい。

質問：（Prof. Iann Barron, 英インモス社）

ずっと，話を聞いていて，やや総括的なコメントを述べたいと思う。誰もが痛感することの１つは，逐次型環境で Prolog をインプリメントするのがいかに困難であるか，という点だと思う。単純なプログラミング言語と比較して，Prolog のインプリメンテーションは極めて複雑でありまた極めて困難である。これらの問題にさらに並列性の問題が付け加わると，われわれが目指しているシステムの開発は何倍にも困難になると思われる。私がいいたいのは，第５世代コンピュータの目標達成に必要な知的努力には２つの側面がある。といえる。その１つは，話を少し前に戻して，これまでわれわれが使ってきた処理要素（computing element）が，１階（first order）の処理要素，即ち，１階の逐次型処理要素という風に特徴づけすることができると思う。そこで最初の側面であるが，それは「並列型の処理システムを作れるのか？」という問題である。第２の側面は，「高階（higher order）の処理要素を作れるのか？」という問題である。

ここで１階の処理要素とは引数としてデータ値をとるプログラム，高階の処理要素とは引数として他のプログラムをとるプログラムを意味する。この２つの問題は現在のところ区別なく扱われている傾向にあるが，明確に区別して別々の問題として検討すべきではないかと思う。

パネリストの方々がこの点でどんな立場に立っておられるのかに注目してみたのだが，雨宮氏はデータ・フローを高階のコンテクストの中で研究されている。しかし，並列性を利用する方法としてデータ・フローが妥当な方法かどうかという問題を，われわれは，まだ，１階のコンテクストの中ですらまともに取り組んだことがないと思われる。データ・フローではプログラムのローカリティを利用し難いと考えるのは，先ほど指摘されたように控え目すぎるかも知れない。

次に May 氏の意見であるが並列性の問題を１階のシステムの中で扱っている。

May 氏はシステムを製作されている。その性能

は，どの程度のものなのだろうか？　May 氏は引き続いて高階のプログラミングに取り組まれるはず，と思うが。

Shapiro 氏は，比較的簡単な方法を採用して高階の処理システムの並列性を利用されており，この方針は問題の単純化にある程度役立っているが，Prolog をベースとする言語を出発点としたことで Shapiro 氏の問題探求能力が制約を受けるのではないかと懸念している。第5世代システムの特色の1つは，出発点として Prolog を恣意的に選んだことだと思う。Prolog は非常に魅力的な特性を備えた言語で，そのインプリメンテーションには並列性の利用に成功しているものがいくつかある。しかし，偶然に利用できたに過ぎないと思えるのだが。

次は Stolfo 氏の発表に対する印象である。Stolfo 氏の業績については存じ上げないが，高階のマシンの問題を単純な並列型環境で扱われてこられた，という印象を受けた。高階のマシンの研究については，その並列性の問題に本格的に取り組めるほどにはまだ十分になされていない，という印象を受けた。

最後は古川氏の発表と ICOT についてであるが ICOT はどんな立場に立っているのだろうか？ ICOT は恣意的な言語環境を出発点とし，まだ十分な研究が済んでいないデータ・フロー環境で並列性を扱おうとしている。古川氏は，高階の処理の可能性を追求しようとされている。

ICOT は，これら3つの問題全部に一度に取り組んでいるが，一度に取り組むものとしては法外な問題に思えるのだが。

座長：では，この質問に答えていただくこととし，最初は，Shapiro氏からお答えいただきたい。

Shapiro：さきほど壇上に立った時は，私はどちらかというと控え目な見解を述べ，高い望みや大きな歩みに水を差そうと努めたが，あなたの発言は私より一層控え目であったと思われる。

控え目な立場をとるという点では，あなたと私は少なくとも同じ見解だが，では，あなたの望んでいる控え目な立場とはどの程度のものなのか？ 実際のインプリメンテーションの中で立証されると思うのだが。あなたの最初の立場は，Occamのような言語はインプリメント可能で並列性を利用できるということだろう。それでは，数週間か数カ月でそれを証明してくれないか？　いや，これは単なる冗談だ。Occam をインプリメントすることと Prolog 的言語をインプリメントすることのギャップは，あなたが考えているほど大きくはないように思える。(C 言語で書かれた?) Prolog のインプリメンテーションの数が私の予想より多いことを見ると，これは Prolog にとってはある程度実証済みのことではないかと思われる。月並な Prolog システムならいとも簡単に実現できることは，多少なりとも明らかである。あと2～3年もたてばこれは学部の学生の論文のテーマにもならないだろう。私のところの学部学生は，このテーマでは特別優等過程試験を受けることができなかった。だから，これは単なる技術の発展という問題に過ぎない。確かに高性能の Prolog を実現するのは依然として困難である。あなたの疑問に対しては，成果をお見せするという形でしか答えられない。実現がどの程度簡単かあるいは困難かを議論するより，やってみるべきだと思う。あなたが思うように難しいかそれともそうでないか，確かめてみていただきたい。

もう1つのコメントは，ICOT が出発点として Prolog を選んだことは恣意的ではなかったかということだが，これについては古川さんの方から説明があると思う。

古川：はい。この2つの目標を同時に追求することが非常に困難なことは，私も承知している。しかし，重要な点は，システム全体のモデルを少なくとも作成したいという点である。そして，情報処理を実現するこのようなモデルをコンピュータ・アーキテクチャと並行して得るためには，多くの側面を同時に研究することが必要である。何

等かのモデルができあがれば，それがかなりおおまかなものであっても，それをたたき台にして出発できる。このようなモデルがこの種の長期的研究には必要と考えるのだが。

雨宮：データ・フロー方式の利点の1つは，粒度の細かいデータ・フロー概念を使うと非常に高いパラレリズムを利用できることである。しかし一方，データフロー・アーキテクチャはローカリティを抽出するのが困難であるという問題を持っている。先ほど指摘したように，粒度の粗いあるいは中程度の粒度のデータ・フロー・レベルをデータ・フロー・アーキテクチャに設定して，ローカリティを利用しコミュニケーション・オーバヘッドを改善すべきだと思う。例えば，1つのファンクションあるいは1つのプロセスは，1つのプロセッサに対して割り付けるべきであって，複数のプロセッサに対し分配するべきではない。

1つのプロセッサの内部では，ハードウェアの構造上の制約から，処理は一度には1つのことしか実行されない。この点で，粒度の細かいデータ駆動制御を全てハードウェアで行なうのは適当とはいえない。従って，既に指摘したように，データ依存分析によって準静的なスケジュルをすることが必要である。粒度の粗いデータ・フロー制御機能の方は，ハードウェア的に実現すべきだと考える。一方，ソフトウェア概念の方は，データ・フローと関数的セマンティクスを完全に保存すべきだと思う。

質問(Prof. Feigenbaum 米スタンフォード大学) 1つだけコメントを述べてから，Stolfo教授のご質問にお答えしたいと思う。コメントというのは，この会場に集った全員が，ある意味ではこの戦いにおけるヒーローではないかということである。このパネルを録音している方があれば，ぜひとも10年後に再生していただきたい。粉塵がすべておさまり，混乱の中から真のパラダイムが出現した後で聞いてみれば，本日のディスカッションはすべて奇異でこっけいに感じられることであろう。当時はこんなにも困惑していたのかと，驚きを新たにするに違いない。

では，Stolfo 教授のご質問にお答えしよう。たぶん私は，この質問に答える資格のある数少ない人間の1人だと思う。私の確固たる信念からいうなら，デザインというものは何等かのアプリケーションから出発すべきものであって，誰かが ── 電気技術者でも論理学者でも構わないが ── 考案した面白そうな仕掛を単に寄せ集めたに過ぎないものは，デザインの名に値しないと思う。

実に明らかなことだが，ICOT の第2段階の研究開発の成果や他の研究者の並列処理に関する成果など家庭の主婦はまったく必要としていない。主婦にアドバイスを与える機能は単純素朴であるから，パソコンでさえその実行に要する時間はとるにたりない。しかし，こういった研究成果を必要とするアプリケーション分野も存在する。いろいろなものを生成し，テストし，結びつける通常の人工知能のルーチンでは，何桁もの改善が可能なはずである。

例えば，最初のエキスパート・システム DENDRAL を見てみよう。現在，化学業界で使われているこのエキスパート・システムの場合，LISP や BCPL を使って面白そうな問題を解こうとすると，DEC 2060 のような従来のコンピュータでは10時間から20時間かかる。また，信号を記号へ変換する問題，即ち信号理解と私が呼んでいるような問題の範疇に属する防衛問題のアプリケーションの場合は，膨大な処理時間が必要だ。アプリケーションについて直接的に語るつもりはないが，大量の信号データを理解するシステムのプロトタイプは LISP マシンで実現されており，その性能はリアル・タイムの3～4桁の時間を要する。リアル・タイムという点が不可欠だ。ちなみに，性能は XEROX－1108 をベースとして測定されている。

それでは，近い将来を展望してみることにしよう。スタンフォード大学では，アプリケーション

が1つ展開されている。それは私の研究室（ヒューリスティック・プログラミング・プロジェクト）と，結晶でなく溶解状態の蛋白質の構造を NMR のデータから解析している化学の研究室との共同研究という形で進行している。そして，ここでもまた，我々の現在のアイデアを実現するのには能力が数桁の性能改善が必要と思われる。

科学や工学の重要なアプリケーションにとっては，現在の処理スピードは十分ではない。ICOT は人工知能を VLSI CAD に応用したエキスパート・システムを発表したが我々がもっと以前に行なった時に失敗したのはアイデアが貧困だったせいではなく，その処理スピードが人間の使用に耐えるスピードの1/1000 でしかなかったためである。それでは，人間の使用に耐えるスピードはどうしたら得られるだろうか？ 経済的な必要性から，われわれは並列処理を目指さざるを得ない。逐次処理に改良を重ねても，飛躍的なスピードの向上は望めないからだ。15%か50%の改善では何にもならない。3桁の改善が必要である。

IC チップの発展は，われわれに新しい技術を与えてくれた。これは，中世でいうならそれまでの筆写に対する印刷機の登場に匹敵するだろう。つまり，シリコンの上にコンピュータを焼き付けてしまうことに比べると，従来の方法でコンピュータを作る作業は原稿を1枚1枚筆写することに等しいのである。ですから，3～4桁のスピードを改善する道は，並列処理の研究をおいてしかない。私や ICOT の方々を始めとして並列処理に取り組んでいる研究者全員は，このような動機から研究を始めている。

最後に本当の未来を展望してみたいと思う。この会場に来ている人の中には，そうした未来において，例えば，化学者を支援するような単なる問題解決マシンではなく，最高の知能を結集したシステム，即ち「知的」マシンの名に真に値するシステムを開発したいと考えている方がいると思う。そのようなシステムは，ある種の学習技術を必要とするだろう。例えば，Lenat による URISCO の実験を始めとする初期の実験が示しているように，知識ベースに蓄えられた知識を新しい知識構造と結合して別の知識構造をマシン中に構築するためには，膨大な量の処理能力が要求される。

Lenat を例にとると，彼は XEROX PARC で XEROX の LISP マシンのネットワークを使っている。夜間に利用可能な LISP マシン全部をつなぎ合わせて，一大ネットワークを構築している。はるか未来にターゲットを据えて本格的な知的マシンを目指すには，このような並列処理方式をマスターしなければならない。さもないと，非常に不経済なマシンに終わるか実現不能な結果になることだろう。

座長：Stolfo 博士，何か付け加えることはあるだろうか。

Stolfo: はい。まず最初に，大量の処理を要するタスクのことなら十分に知っていることを述べたいと思う。私が現在，パラレルプロセッサの製作に取り組んでいるのは，これが理由の1つである。この点については，まったく異存はない。私が問題にしたいのは，どうやって3～4桁のスピードアップを達成するのか，それだけのスピードアップが必要なことがどうして分かるのか，という点である。特定の表現力豊かなフォーマリズムを研究した結果として，スピードアップの必要性に気づくわけではない。様々な問題に取り組みアプリケーションを働かせていく中で，実際に必要なのはどんなデータ処理リソースなのかが分かってくるのだと思う。知識表現と一般に呼ばれているプログラムは，簡単には作成できない。単に，メモリの中を検索してパターンを照合すれば済むという問題ではない。

本質的な問題は，プログラムに入れるべき知識は何か，という問題である。そして，この種の問題は，実際に自分の指の爪を汚してそのようなプログラムをインプリメントしてみる以外，理解できない。以上が私が先ほど述べた忠告である。並

列処理はある意味で「ニワトリと卵」の問題に悩まされている，というのではない。応用を先にしないで特定のフォーマリズムの高速化に重点を置くと道を誤りやすいと思う。

質問（Feigenbaum）：最後に主婦にはそれは必要ないだろうということは，まったく無関係であるとはいえない。1992年のVLSI CADの設計技術者なら，チップ当たり500個のトランジスタを設計する時にそれを必要とするかも知れない。

質問（Dr. Krutar 米国 Naval Research Laboratory）：「必要は未だに発明の母である」と述べられた点について，Stolfo 教授に反論したいと思う。発明の母なのかも知れないが，現在，市場に出回っている発明のほとんどは，必要の産物とは思えない。消費者が必要としたからではなく，消費者が欲しがった結果として生み出されたものである。Stolfo 教授は，消費者は何を必要としているかという観点からではなく何を買うかという観点から，市場を分析しているのだと思う。どんなバーテンダでも知っているように。それから家庭の奥様方を対象としたエキスパート・システムについてだが，私には有力市場の1つに成長する可能性が高い分野に思える。それは私が，非常に活発な性格の3才の子の父親であるせいかも知れないが，子育て用のエキスパート・システムがあったらどんなに素晴らしいだろうと思う。そんなシステムはいつになったら，手頃な値段で商品化されるのだろうか？　最後に，そのようなシステムの開発を望む母親達の声があちこちで聞かれる点を指摘したいと思う。

Stolfo：どのように答えたら良いのか？

発明が必要の産物なのか欲求の産物なのかは，たいした違いには思えない。生まれるべくして生まれたのだと思う。第5世代（プロジェクト）は，次世代コンピュータのあり方やその機能について実に明解に発表しているが，私には次世代コンピュータが何を実際に行なうのかが今だに分からない。大衆の購買欲という観点だけでは説明しきれない。私には，そもそもエキスパート・システムのイメージをこの分野の研究者の多くがはっきり頭に描いているとは，どうしても思えない。これが実際の問題である。この（第5世代）計画は，繰り返し言っているように，まず既存のハードウェアで構築してから，評価すべきものだと思う。コンピュータ・サイエンスでは一般にいって，発明したものを完璧に開発することはないようである。彼等は常により優れたものを追い求めていて，最先端でなくなってしまったものに対しては特に興味を感じない。私が主張したいのは，その逆である。今日あるコンピュータ技術やコンピュータ言語は，並列処理機能がないといっても，まだまだかなりの活躍をすると思う。VAX780やACEがその例である。

May　今の発言に付け加えると，マイクロエレクトロニクス産業がアプリケーションの要請によって動いてきたことはほとんどないと思う。例えば，Intel 社の4004はキャッシュ・レジスタの制御用に設計されたものであって，汎用のマイクロコンピュータとして使うべく設計されてはいない。われわれの経験から話すと，われわれがトランスピュータの設計を始めた時，それが具体的にどんなアプリケーション分野で使われているのかをはっきり頭の中に描いていたかというと，そうは思わない。今では非常にはっきり分かったことで，われわれが予想しきれなかったことが，このデバイスに最適なアプリケーションは特にCADの分野に数多く見い出されている。例えば，有限要素解析，分子シミュレーション，基礎物理学や，現在，フライト・シミュレータに使われているようなグラフィック・アニメーションの分野である。最後は回路シミュレーションで，これは絶対に必要である。これなくして第6世代のコンピュータはどうやって作れるだろうか？

Shapiro：私もこの問題に答えたいと思う。まず，ICOTの発表に対するStolfo博士の批判に

ついてであるが，博士はアメリカ的誤信とでも呼べそうなものをベースとして批判していると思われる。アメリカ的誤信とは，第5世代コンピュータ・プロジェクトはエキスパート・システムの開発を目指している，ということであるが，実際はそうではない。ICOTの代わりに発言することはできないので，私の見解を述べてみたい。第5世代プロジェクトについてのイスラエル的誤信と呼ばれるかも知れないが。

第5世代プロジェクトは，新しい技術を構築しようとしている，と私はとらえている。では，なぜ新しいコンピュータ技術が必要なのか？　それは，既存のコンピュータ技術が非常にはっきりした問題を2つ抱えているためである。1つはコンピュータが遅すぎるという問題，もう1つはプログラマが遅すぎるという問題である。新しいコンピュータ技術は，この2つの問題を解決しようとしているのだ。第5世代プロジェクトの優れた点は，これはIann Barron氏の質問の答になると思うが，この2つの問題を別々に解くことは不可能と見通した点だと思う。一方を解かない限り他方を解くことはできないのだから，2つを一緒に解くという飛躍が必要となるのだ。なぜ1つ1つを別々に解くことはできないのか？　コンピュータが遅すぎるという第1の問題だけを解決しようとする場合，既存のノイマン式言語，即ち低級なノイマン式言語に固執していてはより強力なコンピュータを実現できない。それは，皆さん知っているように，プログラミングが困難だからである。既存の低級なプログラミング言語に並列性とコミュニケーションを追加しようとすることは，プログラミングにとってはまさに後退現象となることだろう。機械語やノイマン式言語より優れたものを，われわれは既に知っている。だから，プログラマの生産性という問題を放っておいて並列処理の問題，即ち処理スピードの遅さという問題だけを解決しようとしても，失敗は目に見えている。

また，第2の問題だけを別個に解決することもできない。既存のマシンを使っては，プログラマの生産性は向上できない。なぜだろうか？　最高の頭脳と最高のツールを使ったとしても，従来のコンピュータで高級言語を遅々としたスピードでしか実行できないためである。だからわれわれは大量の計算を要する問題を解くのに低級な言語に頼らざるを得ず，その結果として困難なプログラミング作業を大量に行なう羽目に陥るのだ。新しい技術を実現し遅すぎるコンピュータと遅すぎるプログラマの2つの問題を両方解決するには，2つの問題に並行して，つまり一緒に取り組んで行かなければならない。逐次型コンピュータで走る現在最高の言語にも劣らない言語を実行できるような並列型マシンを作らなければならない。第5世代計画の優れた点は，こうした方針を設定しそれに向って突き進んでいる点だと思う。

質問（Van de Rietオランダフリー大学）：もう少し技術的な質問をしたいと思う。ただ今のShapiro博士の発言は，第5世代プロジェクトは単にエキスパート・システムを開発するだけにとどまらず，ソフトウエア・エンジニアリングにおける問題の解決も重視している，と私は理解している。第5世代プロジェクトの最初の言語KL 0はProlog，即ち，逐次型のPrologをベースとしており，一方，第2の言語KL 1は，Concurrent Prolog，即ち並列型のPrologをベースとしているが，常に問題となるのは，パラレリズムを一体どのくらい利用できるのか，という点である。4－Queensと6－Queensの問題については，研究成果を拝見する機会があった。その他の問題一例えば，6－Queens－については，これから成果の発表があると思う。

次に，ソフトウエア・エンジニアリングの問題としてまさにうってつけの例であるSIMPOSオペレーティング・システムについて触れてみたいと思う。正確には知らないが，このオペレーティング・システムは30 KのPrologラインからできあがっている，と聞いている。そこで技術的な

点を2つほど聞きたい。その1つは，現在のPrologは逐次型であり，またオペレーティング・システムの多くの部分は逐次性を必要とするが── つまり，こっちは先に，そっちはその次に表現しなければならないが ―Concurrent Prologで OSを記述する場合にはこれを変えなければならない，という点に関する問題である。Prologの逐次性を使えないため，ガードにそして，恐らくはフラグにも，何かを付け加えなければならない。

私の第1の質問は，この点が SIMPOSで検討されているのか，という点である。逐次性を非逐次性に変えるのに，どれだけの時間と労力がいるのか？ SIMPOSを見ると，そこには AND並列が使われているため，AND並列がどの程度可能かが分かる。これはすべてのパラレリズムについて行なわれているのか？ もしそうなら，その結果を教えていただきたい。

古川：われわれは並列型言語の見直し，作り直しをしなければならないと思う。仕様の部分を考えると，共通な点をかなり見い出せる。SIMPOSからその仕様の部分を抜き出せば，その部分を並列論理型プログラミング言語に変換することが可能である。直接変換するのは難しいかも知れないが，従来の言語の場合と比べたらはるかにやさしいと思う。もう1つのご質問は，どのようにSIMPOSからその，パラレリズムを抜き出すのだろうか？

質問：SIMPOSにパラレリズムがどのくらい含まれているかを評価しようとしたことがあるか？

古川：まだない。SIMPOSはできあがったばかりのオペレーティング・システムで，その開発には1年半しかかけていない。時間が十分になかった。この点については，Shapiro博士が素晴らしいご意見をお持ちと思うが……

Shapiro：前にも述べたように，パラレリズムを何かのついでに発見できるとはどうしても思えない。少なくとも本日，そしてあと数ヵ月あるいはあと数年は，パラレリズムを発見するために計画を立てなければならないし，はっきりとプログラミングしなければならない。この理由から私は SIMPOSを Concurrent Prologに直接変換しようとは思わないのだ。SIMPOSはかなり複雑になってしまったが，これは Prologに並列性が足りないためである。従って，SIMPOSの設計者は，メッセージ転送やオブジェクト指向プログラミングをサポートできるように Prologを構成し直さなければならなかった。この機構が，逐次型 Prologの基本機構でなかったためである。

思うに，オペレーティング・システムが Concurrent Prologで直接書かれていれば，オペレーティング・システム全体をインプリメントする際に言語をほとんどもしくはまったく拡張する必要はないであろうし，その結果は非常に小型で明快なものとなるだろう。必要なことは，何らかのストリーム・ライクなインタフェースを様々な入出力デバイスに追加することである。しかし，われわれの研究の現在の段階では，これはオペレーティング・システム全体を Concurrent Prologで書くために必要なことのように思える。正しい方針の下に Concurrent Prologで書かれたオペレーティング・システムは，SIMPOSとはまるで違ったものに見えるに相違ない。

質問（Prof. Arvind, 米 MIT）：いくつかコメントがある。実際には質問ですが……May 氏も Shapiro 氏もパラレリズムを自動的に利用するのは時機尚早と言ったが，この見解には同意しかねる。というのは，まず第1に，アルゴリズムに含まれるパラレリズム全部，もしくはアルゴリズムに含まれる固有のパラレリズム全部が明瞭になるように，アルゴリズムをコーディングする方法を，われわれは知っている。従って，この問題は既に解決済みの問題であるととらえている。

2つ目は，パラレリズムを利用可能なマシンについて，大変優れた抽象モデルをわれわれは確か

に持っているという点である。われわれがまだ持っていないものもある。つまり，われわれはまだ大規模なアプリケーションをこれらの言語の1つでコーディングしたことがないし，それをこれらのアーキテクチャの1つで実行して，パラレリズムを実際に利用したかどうかを確認したこともない。これを実行するには，たくさんのツールを開発しなくてはならない。かなり頑強で非常に優れたコンパイラも必要となるし，1～2週間，連続して稼動しても10分毎にダウンすることのないマシンも作らなければならない。そして，こういったマシンは小型というわけにはいかない。というのは，小型のマシンはないし，また，小型のマシンとして作ったとすると従来のマシンのアイデアをシミュレートしてしまう恐れがあるためである。この点が，このゲームにおけ　最も難しい点である。

われわれは今，実験の段階である。この段階を経てみないと，果して正しい道を歩んでいるかは分からない。この点を，私は明らかにしたかった。

May：その点では，私は意見を異にしたとは思わない。パラレリズムを自動的に実行する方法を探すのは時機尚早だと，私は本当にいっただろうか？　今の段階では，パラレリズムを自動的に利用する方法がなくても直接取り組めるアプリケーションはたくさんあると思う。自動的にパラレリズムを利用することについて何等かの実験を行ってみることは無駄ではない点，それと大規模なシミュレーションが恐らく必要になる点では，私もまったく同意見である。

恐らくは，大規模なパラレリズムを使わないような単純な記法で大規模なシミュレータを設計したくなるのでは，と考えている。

Prolog や Prolog をベースとするプログラミング言語について良く分からないことの一つに，インプリメンテーションのすべてのレベルで Prolog をベースとする言語をどうして使わなければならないのか，ということがある。マイクロ・コードのレベルでは，Prolog をベースとする言語を使うのが最良だとは，確信するに至ってない。マイクロ・コード・レベルのすぐ上のレベルでは C, Occam等の レベルの言語が優れている。それより上のレベルでは，Prolog をインプリメントするこれらの言語の一つがたぶん有望であろう。この種のレイヤの方法は，長年に渡って非常に効果的な手法であった。

確かなことは，マイクロコードのレベルからトランジスタ上で Prolog を使おうとは恐らく思わないことである。だから，Prolog のような言語を使って今後ずっとやって行こうという考え方に関し私が特にひっかかるのは，なぜそのような考え方に至ったのか，という点なのだ。われわれは，最下位レベルで，役に立つツールをいくつか手に入れている。

Stolfo：もう1つだけ手短かにShapiro氏の発言へコメントを述べたい。

「アメリカ的誤信」という言葉は，第5世代コンピュータは，エキスパート・システムマシンではない，という意味で使ったのか。

Shapiro：「アメリカ的誤信」という言葉は，第5世代プロジェクトは，エキスパート・システムの開発を目指している，という意味で使ったのだ。

Stolfo：結構。それなら，私がこれまで2時間の間に使ったエキスパート・システムという言葉を第5世代プロジェクトという言葉で置き換えていただきたい。誰が第5世代プロジェクトを計画したのか？　第5世代プロジェクトとは何なのか？　私の予想では，現在，存在する最良の例は，エキスパート・システム計画である。

その点を明らかにしたいと思う。第5世代計画は新しいコンピュータ技術を開発するプロジェクトだといったが，新しい技術をどうやって開発するのだろうか？　どうやって技術を開発するのだろうか？　科学ではなく，技術である。技術は優れた科学を使って開発する。それでは，優れた科

学とは一体何だろうか？　現象を説明するためには，誰でも理論を構築する。その理論が正しいかどうかは，実験でテストする。

実験を行なってループを評価していただきたい。

理論とは何だろうか？　理論はいかにして構築できるのだろうか？　現象がなければ，理論を構築することは不可能である。それでは，現象とは一体何だろうか？　第５世代計画とは，つまり私が指摘したいのは，この点なのである。

古川：LISP の発明について触れてみたいと思う。LISP はかって，それほどポピュラーな言語ではなかった。LISP が考案されたそもそもの理由は，エキスパート・システムの開発のためではなかったと思う。それが現在では，エキスパート・システムの開発の主要ツールとなっている。そこで私が指摘したいのは，優れた言語というのはそれ自体に傑出した表現力が備わっているものであり，新しい文化というのはそういった優れた言語をベースとして発展するものだ，という点である。この点はこのプロジェクトに含まれていると思う。

座長：並列処理についてはまだまだたくさんの質問があると思うが，残念ながらそろそろ閉会の時刻となった。このパネルは参加された皆様にとって大変に有益なものであったと確信している。貴重な発表と議論を展開されたパネリストの方々と，この会議の運営に協力していただいた皆様全員に，ここで感謝の意を表したいと思う。

この会議を新たな出発点として新世代コンピュータの分野での研究活動と国際協力が，今後より一層加速されることを祈っている。

これで，この国際会議を閉会したいと思う。

どうもありがとう。（了）