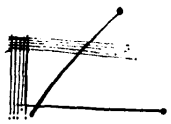
展　望



VLSI コンピュータ・アーキテクチャ†

坂村　健††　石川千秋††

1. はじめに

1970年を発端として進展をつづけてきた LSI 技術は最近ではさらに集積度が向上し VLSI と呼ばれるようになり，発展をつづけている．予測によればこの進展はあと 10 年，1990 年までとまることがないともいわれている．この進展は当然コンピュータの内部構造にも変革をもたらす．1980年代はデバイス技術とともに，VLSI の特質を生かしたコンピュータアーキテクチャ（これを VLSI アーキテクチャという．）が開発されなければならない．つまり VLSI を生かした，または意識したアーキテクチャを新たにデザインする必要があるというわけである．

ところで，コンピュータアーキテクチャがどの様につくられるかを考えると，問題からの要求と，半導体デバイスの技術的制限の二つがアーキテクチャ決定に対しての大きなファクタとなっている事が分かる．従来は，半導体デバイスの技術的制限の方が大きく，問題の要求からのフィードバックはあまりかからなかった．しかし，VLSI は違う．十分応用問題側，要求からのフィードバックを受け入れるだけの能力を持っている．

さらに VLSI アーキテクチャの特徴としてコンピュータなくしては VLSI を使ったコンピュータの設計，構築が不可能であることがある．VLSI アーキテクチャは CAD なくしてはその構築は無理であろう．従来デバイスの制限によりアーキテクチャが決まったのとおなじように CAD システムの制限でアーキテクチャがつくられることが VLSI アーキテクチャの特徴である．しかし逆に CAD の整備により従来コストパーフォーマンス的にひきあわず行えなかった少数多品種のチップも製作が可能となる．

本稿では以上述べた点を考慮しつつ VLSI アーキテクチャを概説する．VLSI の特質を生かしたコンピュータシステムが，どんな考え方で，どのように作れるかについて述べたい．まずアーキテクトからみた VLSI の特質を述べ（2章），次に VLSI アーキテクチャを分類する（3章）．さらに行われる分類の中で RISC と CISC（4章），均一マシンシステム（5章），トリーマシン（6章），シストリックマシン（7章）について詳細を述べる．そして最後に今後の動向と課題につきまとめる．なおここでは特に VLSI のデバイス側の基礎的事項には触れない．特に必要な場合には [JOHO 81]，[CLAR 80]，[MEAD 80] 等を参照されたい．

2. アーキテクトから見た VLSI の特質

VLSI アーキテクチャとは，VLSI 素子そのものの特質を十分に考慮し，それをアーキテクチャ，方式に反映させたものである．設計に於ける階層，PMS (Processor Memory Switch) レベル，ISP (Instruction Set Processor) レベル，RT (Register Transfer) レベル，論理レベルにおいて従来は論理素子が固定，限定されていたことに比べ，VLSI では自由に階層のレベルが選べ非常にスケールの大きい論理素子も使える．つまりアルゴリズムの実現に最適な論理素子が作れる．そしてこれが従来の素子技術では実現不可能であったアルゴリズムにもとづくアーキテクチャの再検討を要求する．

2.1 考えなければならない VLSI の特質

ところでアーキテクチャを構築する場合に考えなければならない VLSI の特質に VLSI では，通信コスト (communication cost) の方が計算コスト (computation cost) をはるかに上回るという事がある[SUTH 77]．というのはワイアが占有する面積がロジック素子の面積よりも大きいこと，回路のディレイは，そのほとんどが論理素子あるいはドライバ素子につながっているワイアに起因していること，また将来デバイス技術が改良されていったとしても，ワイアの

† VLSI-BASED COMPUTER ARCHITECTURE by Ken SAKAMURA and Chiaki ISHIKAWA (Department of Information Science, University of Tokyo).

†† 東京大学理学部情報科学科

ディレイは無視できるようにはならないことなどがあるからだ.

またVLSIで，ある問題解決のためのアルゴリズムを実現する場合，そのアルゴリズムが要求するデータの通信路が規則正しいことが，実現の容易さの点からは望ましい．例えば，非常に優れた能力を持つとされるが，通信路の形状が複雑なためにVLSI上の実現に不向きなネットワークの実例がある.

2.2 VLSI に適さない例

2^m 個のノードを考え，これらのノードに $0,1,\ldots,2^m-1$ の番号を割り当てる．そして，それぞれに割り当てられた番号の2進表示を考える．すなわち $i=(i_m i_{m-1}\cdots i_1)_2$ $\langle i_j=0,1\rangle$．2^m 個のノードを持ちすべてのノード i がノード $S(i)$ につながり，i が偶数ならばノード $E(i)$ にもつながっているようなネットワークのことをシャッフル交換ネットワークという．(図-1) ここで二つの関数シャッフル $S(i)$ と交換 $E(i)$ は次のように定義される.

シャッフル関数　$S(i)(=S((i_m i_{m-1}\cdots i_1)_2))$
$\equiv(i_{m-1} i_{m-2}\cdots i_1 i_m)_2$

交換関数　$E(i)(=E((i_m i_{m-1}\cdots i_1)_2))$
$\equiv(i_m i_{m-1},\cdots,\sim i_1)_2$

($\sim i_1$ は i_1 の 0,1 を反転した値を示す.)

$S(i)$ の結合をシャッフル結合，$E(i)$ の結合を交換結合という．(定義は [KUNG 80 b] によった.) このシャッフル交換ネットワーク上では，シャッフル結合と交換結合を使い，離れたノード間でデータ転送ができる．このシャッフル交換ネットワークを使うデータの転送は種々のアルゴリズムに現われる．シャッフル交換ネットワークの各ノードに比較交換，算術演算の機能を持たせることにより，n 個のデータのバイトニックソート [KNUT 73] を $O((\log n)^2)$ で行えるし，n 点の FFT を $O(\log n)$ 時間で行うことができる．($O(f(n))$ は n が大きいとき $f(n)$ のオーダであることを示す記法である.) 行列の転置，一次回帰式の評価も効率良く行える[STONE 71],[STONE 75]．(回帰式はトリーマシンとよばれるアーキテクチャで効率良く評価できる．これについては6章参照)

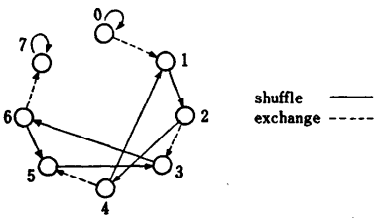


図-1　シャッフル交換ネットワーク

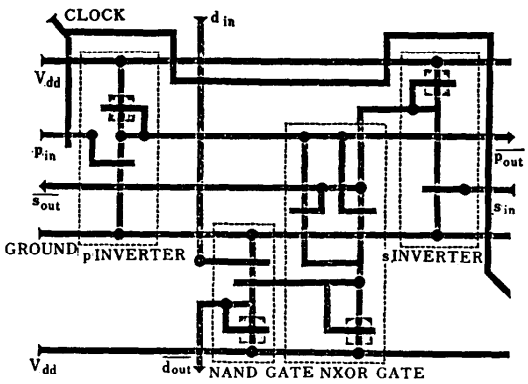
n 個のデータの順列を，シャッフル交換ネットワークを多段に組み合わせると $O(\log n)$ のディレイで実現できることが上述の応用で役に立つ理由の一つである．しかし，このような強力な通信の機能を持つ反面，その構造は規則正しさ，モジュラリティを欠き，ノード間の結合線の長さはまちまちとなる．このためVLSI上での実現には困難が伴う．つまりこのような接続法はVLSI向きではない．n 個のノードのシャッフル交換ネットワークには $O(n^2/(\log n)^2)$ のオーダの面積が必要なことが知られている[KUNG 79],[TOKU 82].

2.3 他のパーミュテーション・ネットワーク

ところで他のパーミュテーション・ネットワークはどうであろうか．図-2 をみるとクロスバは一番スイッチの数が多いので，従来は他の二つの方が好まれていた．しかしVLSIでインプリメントする場合，前述したようにワイアの面積が無視できない．不規則なパターンを示すワイアは必要以上に面積をしめる (図-3)．パターンを作る容易さを考えると n の値によってはクロスバ・スイッチが最も無難な選択になるこ

ネットワーク	スイッチの数	時間	ワイアの面積
クロスバ	n^2	1	n^2
Batcher's のソーティングネットワーク	n	$(\log n)^2$	$\geq n^2/(\log n)^4$
再構成のできる順列ネットワーク	$n\log n$	$\log n$	$\geq n^2/\log n$

図-2　パーミュテーション・ネットワークの性質



ゲートよりも面積を占めるワイア
これは簡単な回路の模式図であるが，かなり実物を反映している．
点線の四角で囲んだ部分は能動回路であるが，それらをつなぐための配線に必要な面積は無視できない.
参考文献 [FOST 80]

図-3　チップ内の通信コスト

とも考えられる.

そこでその結果通信の局所性を重視したアーキテクチャデザインが重要になる. 例えば, バス構造のような単純構成を取るより, ローカルな通信が多い構造の方がチップ面積使用効率も良く, 速度も上がる. 性能を上げるためにはチップ間の通信を少なくしなければならない. そこでどうしても避けられないデータ入出力時間を最大限利用するために, アーキテクチャ的にはパイプライン方式とマルチプロセッシング方式の混合アプローチなどを使うことなどが考えられる.

このようにVLSIアーキテクチャを考えるにあたっては, 通信のコストが計算のコストより大きくなるために, 新しい観点からアルゴリズム, アーキテクチャを再評価することが必要になる. なおコンピュータ構築におけるコストのトレードオフがどのように変化したかは [SUTH 77] を参照するとよい.

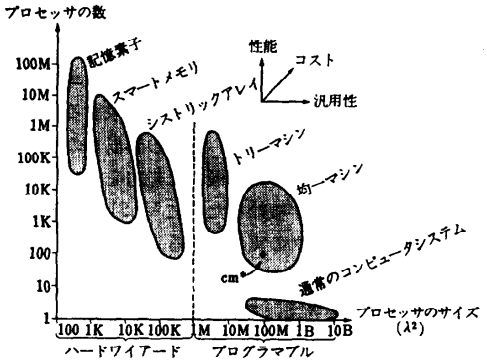
3. VLSI アーキテクチャの分類

ここでは [SEIT 82] に従ってプロセッサエレメント (以下PEと呼ぶ) の大きさとコンピュータシステムの中のPEの数という二つの因子によってコンピュータアーキテクチャを分類し, それぞれ簡単な説明を加え, いくつか重要と思われるものにつき詳論する. なおVLSIコンピュータ・アーキテクチャの分類法にはその他, PEの接続形態, PEの数, PEの大きさなどに注目し分類することも考えられる[KUNG 80 b].

さて図-4はPEの数とPEの大きさをもとに, 現存するコンピュータシステムおよびVLSI時代をむかえて注目を集めているコンピュータアーキテクチャをおおまかにプロットしたものである. それぞれのコンピュータシステムはVLSIによって作られた繰り返し構成要素の複合体よりなるため集団 (Ensemble) 形態をとる.

集団の構成要素としては, RAMからマイクロプロセッサなどがある. PEの大きさについては, 小さい方は専用化した少量のロジックと少量のメモリをもつようなPEであり, 大きい方は, 現在マイクロメインフレームとよばれるワンチップVLSIコンピュータである. 図中には6個のグループがある. そこで以下図中の右下のグループから順に説明を加える.

① 一番右下のグループはいくつかのマイクロメインフレームからなる通常のコンピュータシステムのVLSI化である.



参考文献 [SEIT82]

図-4 VLSIの分類

② つぎのグループは均一のマシンからなる並列計算機だ. それぞれのマシンは独自のランタイムサポートシステムをもつ. たとえばカーネギーメロン大学で作られたCM*[SWAN 77]がある. このシステムでは並列処理の記述には並列処理記述言語を使い, ランタイムサポートシステムがその記述をもとに並列処理による効率向上を図る. 均一であるためフォールト・トレラントシステム, 負荷を自動的に分散させるシステムなどに応用できる.

③ このグループはトリーマシンとよばれる. PEは通常のALUの果す機能の他に, 通信, 同期の機能があり, 数Kから数十Kビットのメモリをもち, トリー状のネットワークにつなげられている. 2と3のグループの違いはおもにプログラム上の違いで, トリーマシンでは, プログラマはトリーネットワークを意識してプログラムをつくり, コンパイラ, ローダがPEに仕事を分散する. あらかじめ, 分散の仕方を指定しなければならない. これにたいして2のグループでは, 負荷の分散は実行時にランタイムシステムがおこなう.

④ 1から3までのグループのPEはプログラム可能であるが, 4,5,6グループのPEはプログラムをすることを考えてはいない. 図中の点線はその区別を表わしている. 第4のグループはKUNGらによってシストリックアレイとよばれているPEネットワークである[KUNG 79a]. 繰り返し構成要素のPEは, 乗算, 加算などの機能を果して, 計算に伴うデータの流れにそった規則正しいネットワークにつながれている. 応用としては, 信号処理, 画像処理, 数値計算などがあげられる.

⑤ このグループはスマートメモリと呼ばれるもの

である．つまり，少量の論理と少し（数ビット）のメモリを組み合わせた PE のグループである．このようなスマートメモリのアプローチは連想処理のために考えられていた[YAU 77]．しかし，その真のメリットは VLSI により，同一のチップにメモリとロジックを乗せられるようになり，はじめて発揮できる．またこれはキャッシュ，データベースシステムへの応用が考えられる．

⑥ これは記憶素子，すなわちメモリ，シフトレジスタなどである．VLSI を RAM に使うことによって，メモリのコストが下がり，アクセス速度，集積密度が上昇する．

一般にいってコストは図の右上にいくほど大きくなる．ところで PE の規模を大きくするのは，汎用性を求めるためである．もし問題がはっきりと定義された場合には，PE，通信の方法を特殊化して，サイズを小さくすることにより，性能をあげられるのでそれほど大規模のものを作らなくても済む．つまり VLSI では専用化アプローチがコストパーフォーマンスを増大させるため重要なわけだ．

以下，第1のグループの従来のコンピュータシステムの VLSI 化，2の均一マシンのグループ，3のトリーマシン，4のシストリックアレイについて詳しくのべる．

4. 汎用化アプローチ-RISC と CISC

汎用のコンピュータシステムの CPU を VLSI 利用のマイクロ CPU で置き替えることは，現在のコンピュータシステムの利用形態の自然な延長として考えることができる．1971年の 4ビット CPU Intel 4004，そして1972年の 8ビット CPU Intel 8008 の登場以来，多くのマイクロ CPU が作られてきた．現在では 16ビットマイクロ CPU, Motorola 68000, Intel 8086, Zilog Z 8000 等の使用が盛んになり，32 ビットマイクロ CPU も登場している．

VLSI 上に CPU をのせる場合には，VLSI 技術の特質を考えたうえで，二つの相反するアプローチがある．

4.1 CISC と RISC

アーキテクチャを構築する際にはさまざまなトレードオフの決定をしなければならない．たとえばプログラム言語中のデータタイプをハードウェアで直接サポートするか否かを決定するには，ハードウェアのデザインの容易さ，また複雑な命令を使いきることのできるコンパイラのコードジェネレータの製作のコストなど多くの因子を考えなければならない．もちろん，できあがったコンピュータシステムでのソフトウェア開発の手間も考えねばならない．このようなトレードオフの決定に当って，システムの要求を満たすためにハードウェアを複雑にしソフトウェアの製作を簡単にする方向と，ハードウェアは VLSI 上にインプリメントしやすいように簡単なものにして，ソフトウェアでシステムの要求を満たすようにする方向の二つのアプローチがある．

今までソフトウェアがはたしてきた機能を VLSI を使ったハードウェアに肩代わりさせようというアプローチがある．そのアプローチにもとづいて作られたマシンは（複数の）高級言語，あるいは高度に抽象化された概念をサポートするための複雑な命令セットを持っている．ハードウェアを複雑にし，大量のマイクロプログラムを利用してそのインプリメンテーションを行っている．これらのコンピュータを CISC (COMPLEX INSTRUCTION SET COMPUTER) と呼ぶ．CISC の定義は厳密には出来ないが，通常 IBM のシステム 38，INTEL iAPX-432 および DEC VAX-11 などが CISC のアプローチのもとにハードウェアおよびマイクロプログラムを複雑にした例としてあげられる．

このアプローチには次のような問題がある．ハードウェアインプリメンテーションが以前よりも楽になったとはいえ，iAPX-432 のような複雑なアーキテクチャを VLSI 上にインプリメントするデザイン時間は長く，デザインエラーの入る可能性が高いことである．VLSI コンピュータアーキテクチャの利点をいかしきるにはハードウェアのデザイン（とくにマスクのレイアウト，チェック）にかかる時間を短くして VLSI 化を容易にしなければならない．また VLSI のもたらす技術の変化を考えて，ソフトウェア，ハードウェアのトレードオフを上手に設定しなければならない．ソフトウェアとハードウェアのトレードオフを決定する場合にはハードウェアのコスト，ハードウェアをデザインそしてデバッグするコスト，プログラミングのコストなどを考えなければならない．また CISC のアプローチにもとづいて複雑な命令を用意しても，複雑で多種にわたる命令を生成するコンパイラの作成が困難なこと，マイクロプログラムを多用する複雑な命令は命令デコーダの複雑化などのために単純な命令を遅くする可能性が高いことなども考える必要がある．

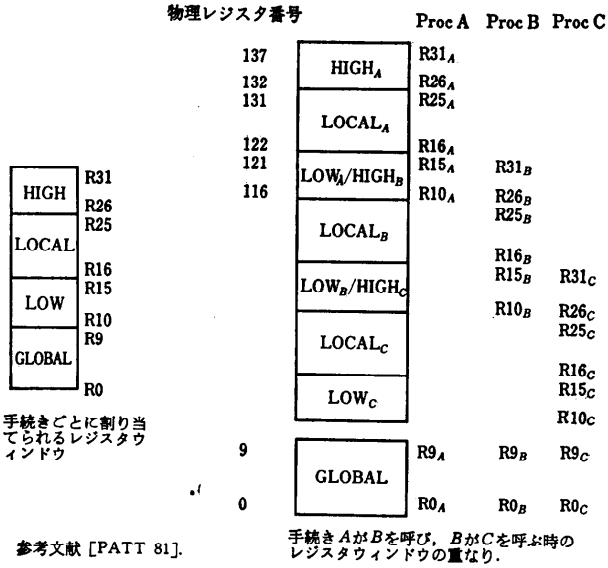


図-5 RISC-I レジスタウィンドウ

一方トレードオフを CISC とは異なる位置に設定したのが RISC (REDUCED INSTRUCTION SET COMPUTER) のアプローチである．RISC はなるだけ簡単な命令セットをもつコンピュータアーキテクチャを設計し，デザイン時間を短くし，同時にデザインエラーも減らし，市場に VLSI コンピュータを早く提供するメリットを追及している．さらに簡単な命令セットにすることにより VLSI 化したときのチップ面積を小さくできるためにチップの歩留りの向上が期待できる．コントロール回路が簡単になるので PLA，マイクロプログラムメモリ，そしてクリティカルパスの中のゲート数が減少してスピードが上昇するメリットも期待できる．コンパイラの作成もコードの選択の余地が少ないために容易である．通常 RISC のアプローチによって作られたコンピュータの例としては RISC-1, IBM-801 があげられる．

4.2 RISC コンピュータ

4.2.1 RISC-I

RISC-I は UC Berkeley で開発されている RISC コンピュータである．RISC-I の製作の目標はマイクロプログラムを使わずに直接に PLA を使った VLSI 上の回路にインプリメントできるアーキテクチャをつくることであった．なお RISC-I は現在プロトタイプが作られてチップの評価が行われている[PATT 81],[SHER 82],[FITZ 82],[PATT 82 b].

RISC-I の特徴としてはつぎのようなものがある．

1. 命令，データ，アドレス，レジスタが同一長 (32-bits).

2. メモリを参照する LOAD/STORE 命令を除いて，1マシンサイクルで実行ができる．1マシンサイクルはつぎのステップからなる．まずオペランドレジスタがアクセスされ，次に ALU 演算を行い，結果をレジスタに格納する．オペランド・レジスタのアクセスと次の命令のフェッチはオーバラップしている．

3. アドレッシングモードはインデックスモード（インデックスレジスタ＋ダイレクト・ディスプレイスメント）しかない．

4. RISC-I では引数およびローカル変数を異なるレジスタバンクに割り当てるようにし，手つづきの呼びだし時と戻るときにはポインタの付け替えのみでレジスタのセーブ，ロードが行える．これをレジスタウィンドウという．(図-5 参照)

レジスタウィンドウの使用は高級言語を効率良く実行するための工夫である．その背景にはソフトウェアの構築に際して，高級言語を使用して小さい手続きを多用する事があたりまえのことになっていくということがある．手続きの CALL/RETURN を高速化するためには，レジスタの退避と復帰およびパラメータと結果の引き渡しを効率良く行わねばならない．レジスタウィンドウの使用はそのための工夫である．プログラムの解析によるとローカル・スカラが最も多くオペランドとしてあらわれるのでローカル・スカラはレジスタに割り当てる．複数レジスタバンクを使い手つづきごとに物理的に別のレジスタセットを割り当てる．レジスタ0から9はグローバルレジスタとして使う．さらに手つづきを呼び出す際には，呼び出す側と呼ばれる側のレジスタ群をオーバラップすることにより引き数の受け渡しのオーバヘッドを少なくしている．

命令の先読みをしているときに問題になる分岐命令の処理は DELAYED JUMP の手法で行っている．DELAYED JUMP はその直後の命令を実行してから，分岐する命令であり，マイクロプログラムではよく使われている．図-6 のように JUMP 命令のあとに NOP 命令を入れてコンパイラにオプティマイズさせる．ダミーとして分岐命令のあとに入れた NOP 命令をどれだけ取り除けるかが，コード効率に関わってくるが，オプティマイザは無条件分岐の時には 90% 以上，条件分岐のときも 20% 以上取り除けたという．

番地	NORMAL JUMP	DELAYED JUMP	OPTIZED DELAYED JUMP.
100	LOAD *X, A*	LOAD *X, A*	LOAD *X, A*
101	ADD 1, *A*	ADD 1, *A*	JUMP 105
102	JUMP 105	JUMP 106	ADD 1, *A*
103	ADD *A, B*	NOP	ADD *A, B*
104	SUB *C, B*	ADD *A, B*	SUB *C, B*
105	STORE *A, Z*	SUB *C, B*	STORE *A, Z*
106		STORE *A, Z*	

参考文献 [PATT 81]

図-6 RISC-I DELAYED JUMP

	RISC-I	Z 8000	MC 68000	INTEL iAPX 432 43201	43202	43203
全体のデバイス数	44K	17.5K	68K	110K	49K	60K
ROM 容量	44K	17.5K	37K	44K	49K	44K
新しくデザインしたデバイス数	2K	3.5K	3K	5.6K	9.5K	5.7K
整然度因子	21.7	5.0	12.1	7.9	5.2	7.7
チップサイズ (square mils.)	124K	60K	69K	103K	115K	117K
コントロールサイズ	7K	37K	42K	67K	45K	47K
コントロールの占める割合	6%	53%	62%	65%	39%	40%
最初のチップができるまでの月数	17	30	30	33	33	21
デザインに要した man・months	30/2	60	100	170	170	130
レイアウトに要した man・months	12	70	70	90	100	50

参考文献 [FITZ 81]

図-7 RISC-I のデザインデータ

ベンチマーク	RISC-I	68000	Z 8002	VAX-11/780	11/70	C/70
	(msec.)	RISC-I の実行時間を1とする相対実行時間				
E-文字列のサーチ	.46	2.8	1.6	1.3	0.9	2.2
F-ビットテスト	.06	4.8	7.2	4.8	6.2	9.2
H-linked list	.10	1.6	2.4	1.2	1.9	2.5
K-ビットマトリックス	.43	4.0	5.2	3.0	4.0	9.3
I-クィックソート	50.4	4.1	5.2	3.0	3.6	5.8
Ackermann (3,6)	3200	—	2.8	1.6	1.6	—
再帰クイックソート	800	—	5.9	2.3	3.2	1.3
パズル (添え字を多用)	4700	—	4.2	2.0	1.6	3.4
パズル (ポインタを多用)	3200	4.2	2.3	1.3	2.0	2.1
SED (パッチエディタ)	5100	—	4.4	1.1	1.1	2.6
ハノイの塔 (18)	6800	—	4.2	1.8	2.3	1.6
平 均		3.5	4.1	2.1	2.6	4.0
標準偏差		1.8	1.6	1.1	1.5	2.8

参考文献 [PATT 82]

図-8 RISC-I のベンチマーク

RISC-I の設計を例にとると，デザイン時間が典型的な CISC VLSI コンピュータの 1/5 ですみ，さらにコントロール回路が VLSI の上で占める割合が普通の 1/10 程度になった (図-7 参照)．これは命令を簡単にしてマイクロプログラムを使わずに実現できるように努力したことの直接の帰結である。

このような命令セットをもつ RISC-I の性能評価については図-8 を参照されたいが，VAX-11 などのコンピュータにくらべても実行時間の点からはコストパーフォーマンスがよいことが分かる．

4.2.2 IBM-801

1975 年以来研究開発が続けられている IBM-801 は既存のシステムよりも良いコストパーフォーマンスをもつ高級言語システム (マシン，コンパイラ，そして OS) を作ることを目的とした研究の一環として作られた RISC コンピュータである[RADI 82]．IBM-801 の特徴は RISC-1 と同様に，基本マシンサイクル内で実行できる命令セットである．またメモリ階層，命令，I/O アーキテクチャそしてソフトウェアを設計するさいに CPU がアイドルとなる時間をすこしでも減らそうとしたことがあげられる．

IBM-801 のうえで動く高級言語は PL.8 とよばれる PL/I のサブセットである．このコンパイラには IBM のコードの最適化の研究の成果がつぎこまれていて，きわめてよいコードを出力する[AUSL 82]．複雑な命令を用意する代わりに，高速に実行できる単純な命令を用意し，最適化コンパイラで効率良くコードを生成することを方針とした．システム内でのプロテクションについてもハードウェアによる実行時のチェックよりも高級言語による実行以前のチェックを重視した．コンパイル時あるいはリンク時にチェックを行い，手続き呼びだしのパラメータの数と型が一致していること，分割コンパイルされた手続きによって共有されているデータの取り扱いが一貫していること，配列のインデックスがオーバフローしないこと，分岐の飛び先が正しいエントリであることなどを確かめている．

IBM-801 の場合にもプログラマが小さな手続きをたくさん使ってソフトウェアを作りあげることに大して心理的な障害をいだかないように，手続き呼び出しを高速化することに注意が払われた．リンクのオーバヘッドを少なくし，パラメータの受け渡しはレジスタを介して行うようにしている．レジスタの割り当てには，IBM で研究された最適割り当ての研究成果が使われている[CHAI 82]．

ジャンプの処理は RISC-I と同様の処理を行っている．命令の先読みをして実行速度を上げているが，分岐命令があると，その処理の流れが乱される．IBM-801 は Branch with Execute という命令を用意している．

コンパイラのモジュール名 (PL. 8)	コードサイズの比 801 対 S/370	実行命令数比 801 対 S/370	データ参照回数比 801 対 S/370
FIND	1.02	.91	.60
SEARCHV	.93	.83	.38
LOADS	.83	.91	.43
P2_EXTS	1.00	1.00	.57
SORT_S1	.86	.78	.59
PM_ADD1	.86	.96	.63
ELIMISS	.87	.86	.69
PM_GKV	.92	.76	.46
P5DBG	.98	.81	.52
DESCRPT	.86	.75	.42
ENTADD	.79	.76	.42
合計	.90	.80	.50

参考文献 [RADI 82]

図-9 IBM-801 のベンチマーク

これは RISC-1 のアプローチと似ている．Branch with Execute はその直後に置かれた命令を実行しながら，命令キャッシュに分岐先の命令を取り込む．例えば

```
LOAD R1, A
BNZ L
```

という命令列を考えてみる．分岐が起るとすると，CPU は命令キャッシュに L の命令がとりこまれるまでアイドルになってしまう．Branch Non-Zero を Branch Non-Zero with Execute にかえて，LOAD をそのあとに持ってくるとつぎの命令列になる．

```
BNZX L
LOAD R1, A
```

この命令列はもともとの命令列とおなじ意味をもち，しかも LOAD と分岐のための命令フェッチをオーバラップできる．IBM-801 ではプログラム中の分岐命令の 60% は Branch with Execute の形式にできるという．

性能評価として**図-9** に IBM S/370 との比較を上げた．システム全体のコストパーフォーマンスを追求してコンピュータの命令を決めていくのはアーキテクチャを決めるためのオーソドックスな方法であり，そのような地味な評価にもとづいてのみ RISC, CISC の利点，欠点の評価はおこなえる．その意味で CPU とともに，その特徴を生かすためのコンパイラ，そして OS を作りあげそれぞれの及ぼす影響を考慮しながらシステムアーキテクチャの設計・評価を行ってきた IBM-801 システムの将来は興味深い．

4.3 CISC コンピュータ

ここでは CISC の代表例として取り上げられることの多い Intel iAPX-432 をはじめとして，いくつかの大規模 VLSI マイクロ CPU をとりあげてみる．

4.3.1 インテル社の iAPX プロセッサ[RATT81] [JOHN 80a], [COMP 81]

このプロセッサは，H-MOS (High Performance MOS) により作られた 64 ピン QUIP (QUad In-line Packge) に納められた三チップよりなる．43201 は，命令デコーダで十一万トランジスタで構成されている．43202 は，マイクロ命令実行ユニットで四万九千トランジスタで構成され，43201 と合わせ GDP (General Data Processor) を構成する．もう一つの 43203 は，IP (Interface Processor) と呼ばれ，六万トランジスタより成り，入出力のサポートを行う．iAPX 432 は，マルチ・プロセッサ用オペレーティング・システム (iMAX) および高級言語 Ada のためのアーキテクチャ・サポートが成されている．プロセッサ間の通信はインタ・プロセッサ・コミュニケーション・プロトコルで限定されているが，バス構造は柔軟なデザインができる．また，ソフトウェアの変更なしでデータ・プロセッサの数の増減が可能となっている．IP は外部 I/O とのやりとりをするためのプロセッサで，GDP から見ると通常の割り込み動作はなく，メモリ・マップド・ペリフェラルとして扱える．

チップ・デザインの手法は [MEAD 80] とほぼ同様な手法で行われて，デザインが進むにつれて CAD が広く使われたという．GDP には 4K×16 ビット，IP には 2K×16 ビットのマイクロプログラム ROM を持っている．マイクロプログラムとハードウェア・サポートにより，シリコン・オペレーティング・システムが組み込まれている．これにより，例えば「Send Message」が，ミニコン用の高速 OS の五倍，メインフレームの OS の 20～30 倍の速度 (80.875 μs) となる．このシリコン・オペレーティグ・システムのための全マイクロプログラムの約 40% を使用している．また，基本命令セットのマイクロプログラムは 6% に過ぎず，ほとんどの命令は一マイクロ命令で実現されている．これは，43201 の命令デコーダ部に納められていて，メイン・マイクロプログラムメモリには含まれていない．最初のマイクロプログラムは ADA を拡張した言語で記述してアルゴリズムの正しさを検証した後に，ハンドアセンブルされた．記述，検証にほぼ同じ期間かかったという．(**図-10** 参照)

バーチャル・メモリのサポートはマイクロプログラムの 7% を使い，キーレジスタのバックアップ・ハードのサポートにより実現され，論理空間 1T (テラ) ア

機能ユニット	典型的な操作	実行時間 (8 MHz クロック [μs])
可変桁整数演算ユニット	32 bit 整数乗算	6.25 (16 μs on IBM 370/148)
浮動小数演算ユニット (マイクロ)	80 bit 浮動小数乗算	26.125 (38.5 μs on IBM 370/148)
バレルシフタ	32 bit フィールド抽出	1.875
アドレス生成部 (TBL 付き)	32 bit メモリ参照	0.75
シリコン オペレーティング システム	SEND MESSAGE	80.875

(a) iAPX 432 の実行速度 [RATT 81]

機 能	容量 (bits)	パーセンテージ
基本命令セット	3,680	6
浮動小数演算	11,680	18
実行環境	6,400	10
仮想アドレス	4,800	7
異常処理	2,640	4
シリコンオペレーティングシステム	26,400	40
マルチ・プロセッサ制御	8,640	13
デバッグサービス	1,280	2
合 計	64K	100

参考文献 [RATT 81]

(b) iAPX 43201/02 のマイクロプログラム容量

図-10

ドレス(アドレス幅 40 ビット), 物理空間 16 M アドレス (アドレス幅 24 ビット) をサポートできる[POLL 82].

このプロセッサにはフォールト・センシティブ機構を備えている. 即ち, 通常の動作を行うマスタ・モードからチェッカ・モードに切り換えると, プロセッサはバスへの出力をせず, センスを行い, 自分の計算結果とバスとを比較して, 違っている場合にエラーを出すことによって異常を検出する.

機械命令セットは高級言語指向でアドレス・モードは, ゼロアドレス, BASE+INDEX, BASE+D (Direct オペランド), BASE+D+INDEX があり, 命令コードとオペランドとは直交型の (Orthogonal) 命令フォーマットをとっている. また, コンパイラの負担軽減のため, 命令からはレジスタを見せず, メモリおよびハードウェア・サポートの式評価用スタックをオペランドとしている. さらに命令は可変ビット長で, 長さは 6〜200 ビット程度となっている. INTEL はこのマシンのアプリケーションプログラムはほとんどすべてが ADA で書くものとして, アセンブラは公開していない.

IP は 16 ビット・マイクロ・プロセッサで, 16 ビット ALU とテンポラリ・レジスタ三個および, 作業レジスタ・ファイルを 30 個持っていて, 3.3 M バイト/秒の転送を行える.

オペレーティング・システム iMAX は ADA で書かれていて, ユーザの書いた OS の部分と, 用意されているカーネル (核) とを完全に混合することができる. これは, 単に言語がおなじであるという理由ではなく, モジュラ設計が成されていることによっている.

4.3.2 HP 社の 32 ビット VLSI チップ

このチップは 45 万トランジスタにより構成され, 18 MHz のクロックで動作する. 9 K×38 ビットの制御記憶 (ROM) を持ち, マイクロ命令は PLA でデコードされ各制御線を駆動する. レジスタ・スタックは 28×32 ビットのレジスタを持っている. ALU には四つのレジスタと 0〜31 ビットのバレルシフタを備えている. また, 特別なロジックにより 32 ビット×32 ビットの乗算を 33 クロック (1.8 μs) で, 64 ビット÷32 ビットの除算を 65 クロック (3.5 μs) で実行する. MPB (Memory Processor Bus) インタフェースは七つのアドレス/データ/メッセージ・レジスタと制御ロジックにより外部のバスとインタフェースする.

CPU の命令セットはスタック指向で, 以下のように大別される.

1. ロード/ストア命令

TOS (Top of Stack) とメモリ間のデータ (ビット, バイト, ハーフワード, ワード, ダブルワード) 転送命令.

2. スタック/レジスタ操作命令

TOS のシフト, ローテイト, およびスタックとレジスタ間のデータ転送命令.

3. 算術論理命令

32 ビットの整数演算, 32 ビットと 64 ビットの IEEE 標準浮動小数演算, 型の変換, および 32 ビット論理演算命令.

4. プログラム制御および分岐命令

条件, 無条件分岐, プロシジャコール, イグジット命令. 高級言語のためのディスパッチングとデバッギング・エイドをサポートしている.

5. MOVE およびストリング命令

バイト・アレイおよびストリング・タイプのデータ操作命令.

6. I/O および割り込み命令

7. オペレーティング・システム開発用の特殊命令

HP 3000 と同様四つの TOS レジスタのプッシュ, ポップおよびデータ・バリッド・インジケータを

ハードウェアでサポートし，メモリ上のスタックとのやりとりをマイクロプログラムで効率よく行えるように工夫してある．またメモリ参照命令のためにベース・レジスタとのオフセット加減算のためのハードウェア・サポートがなされている．これにより，領域チェックを含めた標準的なロード命令は 550 ns, 64 ビットの浮動小数乗算は 10.4 μs で実行する．

CPU のパーフォーマンスは広範なパイプラインの採用により高められている．パイプラインは以下に示す部分で特に効果を上げている．

1. 機械命令の実行（フェッチ，デコード，実行）

現在実行中の命令は，CIR (Current Instruction Register) に保持する一方，NIR (Next Instruction Register) にある次の命令がデコードされ，生成されたマイクロ命令がシーケンシング・スタックに積まれる．命令がデコードされると次の命令がメモリからフェッチされ PIR (Prefetch Instruction Register) に置かれる．

2. マイクロ命令の実行（32 ワード・ページアドレス，ワードアドレス／デコード，実行）

マイクロ命令の実行は三サイクルで行われる．最初のサイクルでは，要求されたマイクロ命令の入った 32 ワードのページがアクセスされる．次のサイクルでは，このページの中から必要なマイクロ命令を取りだしデコードする．最後のサイクルでマイクロ命令の実行が行われる．この三つのサイクルがパイプラインによってオーバラップされるため，実質のマイクロ命令の実行は一サイクルで行われる．シーケンシャルでないオペレーションの影響を最小化するようなハードウェア機構が内蔵されている．スキップや短い分岐は 32 ワードのページ内で納まるようにされ，さらに長い分岐でも，もう一サイクルの増加で済むようになっている．

3. メモリ・オペレーション

32 ビットのメモリ・リードは二サイクル (110 ns) ごとに開始できる．これによって命令，データフェッチ，およびブロックデータ転送がオーバラップ可能となる．

チップが複雑なため，セルフテスト用のハードウェア，マイクロ命令が納められている．電源が投入されるとチップ内部のセルフテストが行われ，これが完了するとプロセッサメモリ間オペレーションのテストが行われる．

ソフトウェアのデバック機能も用意されていて，チップのデバッグ・ポートを使用してマイクロプログラムと機械命令のデバックができる．マイクロ命令レジスタやスタック・レジスタの入出力がビット・シリアルに行えるのみならず，マイクロ命令のブレークポイント・レジスタ，Halt/Step コントロール・ビットをアクセスできる．また高級言語のデバック用にプロセッサ・ステータス・レジスタと結合された特別のロジックとデバッグ用機械命令群が用意されていて，プロシジャ・レベルおよびライン・レベルでのブレークポイント設定，シングルステップ，トレースなどが行える．

この VLSI CPU のプロトタイプは，2.5 μm ピッチの N チャネル MOS プロセスによって作られた．また，同社では線幅 1.5 μm 線間 1.0 μm のハイ・パフォーマンス N チャネル MOS プロセスを開発している．

4.3.3 ベル研究所の 1 チップ C-MOS マイクロプロセッサ[COMP 81]

このマイクロプロセッサは MAC 32 と呼ばれ，ベル研究所 (Murray Hill, NJ) によって開発された 32 ビット 1 チップのマイクロプロセッサである．このチップの開発の目的は，C で書かれたソフトウェアおよび UNIX との互換性を保ったままで，システム全体のコストパーフォーマンスを上げることにあった．実装は寄生容量によってひきおこされるラッチアップの無い「TWIN-TUB」と呼ばれる C-MOS 回路技術で行われている．ユーザの観点からは，コードサイズと実行速度の両者に関して高級言語のコンパイルが効果的に行われるように，またオペレーティング・システムの効率良いサポートをするように CPU に工夫がなされている．命令セットはモトローラ 68000 と同様に，どの命令コードにもいずれのオペランド・ディスクリプタもが使える直交型 (Orthogonal) である．命令にはビット操作，フィールド操作，ブロック転送，コンテクスト・スイッチングのためのスタックに関するレジスタのプッシュ／ポップなどがある．命令セットの最適化には，現存する C で書いた C のセルフ・コンパイラを利用してシミュレーションを行った．これによって，C のコードは VAX に比べて 10～20% 短くすることができた．MAC 32 がサポートするデータ・タイプは，バイト，半語，語，ビットストリングである．

オペレーティング・システムのサポートとしては，プロセスのスイッチングを行う命令や，例外事象の処

理を行う機構が組み込まれている.

CPU の内部は外部メモリとの通信を司るフェッチ・ユニットと，データ処理を行うエグゼキューション・ユニットの二つの分離した部分から構成されている. この方法はインテル 8086 や，TI 99000 でも採用されている. フェッチ・ユニットには，フェッチした命令を貯えるインストラクション・キューとアドレス計算を行うアリスメティック・アドレス・ユニット (AAU と呼ぶ) がある. エグゼキューション・ユニットには 0〜31 ビットのバレル・シフタと加算を 60 ns で行う ALU があり，32 ビット・バスと 17×32 ビットのレジスタ・ファイルが接続されている.

チップの実装は，以下のように三つのフェーズで進められた.

1. 第一フェーズでは，チップの 32 ビット・データパス部分と，ALU，AAU，バレル・シフタ，レジスタ・ファイル，インストラクション・キューがゲート・マトリクス手法によって作られた. ALU は，domino CMOS と呼ばれる回路技術によってグリッチのないダイナミック CMOS 回路によって構成され，3.5 μm ルールによって 32 ビットの加算を従来の CMOS の約半分の 60 ns で演算する. ゲート・マトリックス・データベース (ゲートアレイを使った論理回路データベース) は，記号による回路記述とテクノロジ・データとを分けて構成してあるため，例えば 2.5 μm テクノロジ・ファイルができ上がると，一晩コンピュータを走らせることによってマスク・データは更新され，シミュレーション・ファイルも間もなく使えるようになった. このようにして得られた 2.5 μm ルールのシミュレーションによると，加算時間は 30 ns に改善されると予測されている.

2. 第二のフェーズでは，マイクロプロセッサ全体，即ちすでに作られている 32 ビット・データパスと，PLA とポリーセル (poly-cell)・ロジックによるコントロール・ロジックがシリコン・ブレッドボードとして作られた. これは，10.5 mm×13.9 mm のチップで十万トランジスタからなる. このチップは五人のデザイナがロジックを受け取ってから三か月で完成した. 最終的につくりあげられるプロダクション用チップに比べるとサイズも大きく，遅いが，デザインのチェック，ロジックやタイミングのシミュレーション・ツールや，レイアウト CAD などを整えるのに役立ち，コンピュータ・デザイン全体のソフト面，ハード面の検討に役立った.

3. 第三フェーズは現在実装中のスーパセルによるプロダクション・チップである. 一つのスーパセルは二万トランジスタからなる現在のものとおなじ RALU である. その他のスーパセルは，PLA ユニットとゲート・マトリックス・モジュールによるコントロール・セクションである. 2.5 μm ルールで，32 MHz クロックで動作する予定である.

アーキテクチャについての詳細は [BERE 82] を参照されたい.

4.3.4 NS 社の 32 ビットマイクロプロセッサ NS 16000[COMP 81],[BAL 82]

このチップは，290 ミル角 (7.37 mm 角) に約六万トランジスタを載せた 16032 と呼ばれるプロセッサである. スレーブ・プロセッサによってソフトウェアからトランスペアレントに基本命令セットの拡張ができる. 例えば，メモリ・マネージメント・ユニット (16082) を付加するとデマンド・ページ方式のバーチャル・メモリ・システムが使用できるようになる. 82 の基本命令セットは，8，16，32 ビットの整数，ストリング，パックド・デシマル，ビット／ビットフィールド，アレイの操作および高級言語プログラムの制御構造をサポートする.

プロセッサは 48 ピンの DIP に実装され，24 ビットのアドレス，16 ビットのデータバスをマルチプレクスして使用している. プロセッサの内部では，32 ビットのデータパスおよび 32 ビット・アドレス計算機構を持つ. バーチャル・メモリのサポートのため，現在実行中の命令を再実行可能なようにメモリ・サイクル・アボート・ピンが用意されている. アドレス変換キャッシュのミスが起きたとき，プロセッサがメモリ・テーブルを参照できるようにフロート・ピンが用意されている. また高速バス・プロトコルによりメモリ・マネージメント用のスレーブ命令を送出している.

バス・サイクルは内部の処理とオーバラップしている. 命令の実行もフェッチとオーバラップするため，8 バイトの命令先読みキューを持っている. 命令の実行は三ステージのパイプラインで行われ，ローダーステージでは，キューから命令を取りだし，命令を基本オペコードとオペランド長，アドレス・モード，ALU 制御にデコードする. ローダは 100 ns ごとに 1〜2 命令バイトのデコードを行う. 現在の命令のデコードが完了するとローダはマイクロプログラムによって次の命令のための準備をする. プロセッサステージで

は，マイクロプログラム・シーケンスのエントリを選択する．マイクロプログラム・ステージでは，内部バス転送，ALU，特殊機能ロジック，バス・インタフェースへの転送要求などの制御を行う．

マイクロ命令は 100 ns で実行され，マイクロプログラムメモリの大きさは，127 ワードのセルフテスト・ルーチンを含め，1300×18 ビットである．

レジスタには，8 個の汎用レジスタ，8 個の特殊レジスタ，8 個の 32 ビット浮動小数レジスタがある．演算のアドレシング・モードは，DEC PDP-11 と同様な完全対称型で，直接 (immediate)，メモリ相対，スケール付きインデクス (scaled index. VAX-11 のものと同様)，エクスターナル (表を使い，間接にアクセスする)，スタックおよびインプライド (implied) がある．

メモリ・マネージメント・ユニットはダイナミック・アドレス変換の機能，128 バイトの変換表にもとづく仮想記憶のサポート機能，ソフトウェア・デバッギングのサポート機能 (スタックのオーバフロー，アンダフローチェック等々)，記憶領域の保護機能を持つ．仮想記憶サポートのためのアドレス変換バッファは，メモリ・マネージメント・ユニットに内蔵され，32 個の連想メモリ (Content Addressable Memory) で構成されていて，LRU (Least Recently Used) アルゴリズムで管理されている．

これらのチップで構成される 32 ビット・マシンでは，また仮想マシンをサポートするための考慮もなされている．実行モードとして，ユーザ・モードとスーパバイザ・モードが用意され，これらの空間は分離されている．そして，ユーザ・モードで NS 16000 の機械語のすべてをシミュレートすることが可能である．これらの機能を使った OS および開発システムは，現在製作中とのことである．

ハードウェア・デバッギング機能として，ブレーク・ポイント命令，配列添字範囲チェック命令，トレース・モード実行，メモリ・マネージメント・ユニットによるスタック・オーバフロー・チェック機能がある．これらの機能を使用することにより，現在広く使われているインサーキット・エミュレータなしでデバッグが可能となる．

モジュラ・ソフトウェアのためのサポートも行っている．モジュール単位でプログラムを作り，モジュールごとにモジュール表をつくりだす．この表には，スタック・ベース，リンク・ベース，リンク・ページ，プログラム・ページなどの情報が入っていて，これらの外部アドレシング機能などを使うことによって，リンケージ・エディト・プロセスが不用になる．また，ROM のなかでもコードの再配置が可能となる．

現在開発中の OS の核の部分は，多くの機能を持っている．例えば，実時間応答可能なようにスケジューラを変更したり，プロセス間通信機能，空間の分離機能，マルチ・プロセッサへの拡張などが容易にできるよう工夫されている[COM 81], [BAL 82]．

このように CISC を実現するマイクロプロセッサは通常のミニコンピュータとよばれるコンピュータに匹敵する能力を備えている．これらの VLSI のインプリメントには高度なプロセス技術が必要とされ，ウェーハ (wafer) の歩留まりもあまり高くないと言われている．(1981 年の MICRO-14 の会場で BELL 研の 32 ビットコンピュータの歩留まりは数%だったと噂されていた．) 一方では RISC-I は大学でインプリメントが進んでいることを考えると VLSI 技術のもとではインプリメントのしやすさという因子もアーキテクチャを考えるうえで無視できないと言えよう．これらの議論については [CLAR 80], [PATT 80], [PATT 81] [PATT 82 b] を参照されたい．

5. 均一マシンシステム

均一マシンシステムには X-TREE システム，CM* などがある．X-TREE システムは UC Berkeley で開発が進められている単一 VLSI プロセッサを使い，それをトリー状のネットワークにつないでシステムを構築する．それぞれのプロセッサは 100 万個のトランジスタが 1 チップにのるという前提のもとに，グローバルなデータの転送を減らすようにローカルデータ用のメモリを 64 K バイト以上もつ．このプロセッサは演算機能だけではなく，入出力もサポートできるように設計されている．X-TREE プロセッサのコードはスタック指向であり，高級言語向きに設計されている．

ネットワークとしてバイナリトリーを選んだのはグローバルな通信があるときには，リングネットワークよりもよい性能をもつことがシミュレーションで分かったからとされている．実際にはプロセッサ間の通信量が極度に多くなる部分の負荷分散，信頼性の向上のためにプロセッサ間のリンクは 5 本まで用意する．

X-TREE システムは XOS とよばれる OS の管理のもとで動作する．XOS はケーパビリティアドレッ

シングをサポートして，システムオブジェクト（メモリ，プロセッサ，プロセス，プログラム等）のプロテクションを行う．また X-TREE のプロセッサがグローバルメモリを参照したときに，そのデータをローカルメモリにロードするための制御を行う．信頼性をあげるための工夫，負荷の分散を行うための機能を XOS に入れるべく VAX-11 上のクロスソフトウェアシステムで開発が行われている．X-TREE システムについての詳細ならびに CM* については [DESP 78], [SEQU 78], [SWAN 77] などを参照されたい．

6. トリー・マシン

トリー・マシンはトリー・アルゴリズムにもとづく構成形態をとるコンピュータシステムである．トリー・アルゴリズムはトリーの形の階層構造をした通信形態を持つ．ノードにプロセシング・エレメント (PE) があり，ノード間の枝が通信路に対応している(図-11)．

このようなトリー構造は，

1. 通信時間，サーチ時間，ファンインが n をノードの数として $O(\log n)$ である．

2. ルート・ノードが入出力のためのノードに自然に対応する．

3. 取り扱える問題のサイズには制限がない．小さな問題はルート付近で解き，大きな時にはトリーを大きくすることで対応できる．

4. 多数の PE が同時に処理を行う高並列処理システムである．

などの特徴を持つ．

VLSI 上で実現する際に問題となるのは，接続が一様ではなく，ルート・ノード付近で長い線がいることである．けれども，図-12 のように PE とローカル・メモリのペアを並べれば，面積使用効率を落すことなく，しかも信号線ドライブに必要なトランジスタはトリーノードの個数の対数に比例してしか大きくならないので都合良くインプリメントできる．

トリー・アルゴリズムはトリーの持つ上記の特長を生かすようなアルゴリズムである．通信帯域幅はルートに近づくほど通信路が少なくなり，狭くなっているので，下位の PE になるだけ処理を集中し，上位の PE がボトルネックにならないようにすることが大事である．

トリー・マシンでは，$O(n)$ のソーティング，$O(n)$ の行列乗算が可能であることが分かっている[MEAD80]．

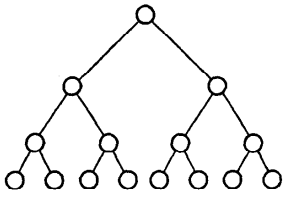
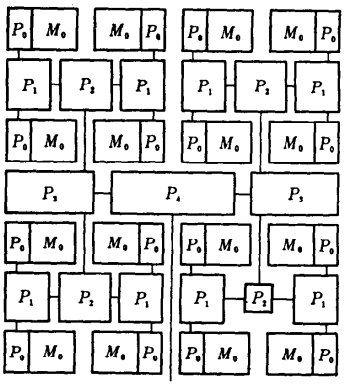


図-11 木構造



参考文献 [MEAD 80]

図-12 バイナリ・トリー・プロセッサのレイアウト例

6.1 トリー・サーチ・アルゴリズム

トリー・アルゴリズムの一つにトリー・サーチ・アルゴリズムがある．トリー・サーチ・アルゴリズムはトリーのリーフ（一番下のノード）の PE にデータを保持して，データに関する問い合わせを $O(\log n)$ 時間で処理するアルゴリズムである．ここで問い合わせとは，あるデータに一番近い値を持つデータを決定すること，データの順位を決定すること，関係データベースの操作などである．

また問い合わせは次のように行われる．

1. ルート・ノードからリーフにデータを送る．

2. すべてのリーフで送られたデータをもとに比較などを行う．

3. 比較結果が中間ノードでまとめられながらルートにたどり着き，最終の答えが得られる．(図-13参照)

ある問題の解の候補をすべて生成して，その中から条件を満たす正しい解を選びだす場合にトリー・サーチ・アルゴリズムを応用することができる．このときリーフの PE には解の候補の一つが対応する．この方法によって NP 問題を解くこともできる．ただし多項式時間で，それを解くには，非常に多数の PE が必要である．(NP 問題とは，Nondeterministic なチ

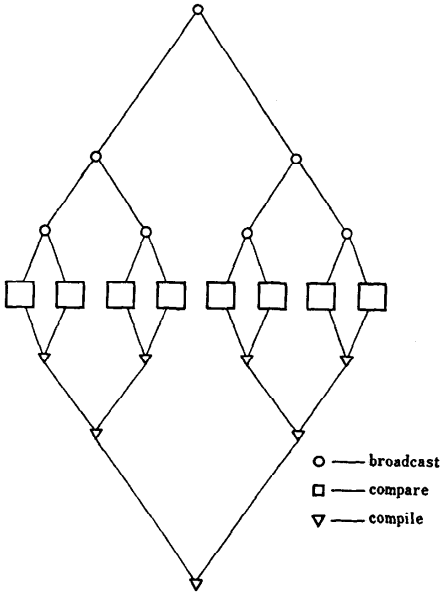


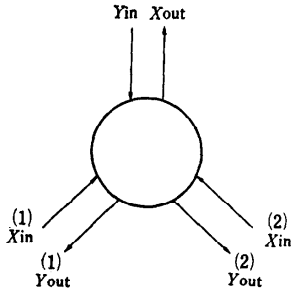
図-13 トリーマシンによるデータベースオペレーション

問 題	時 間	PE の数
ソーティング	N	N
行列乗算	N^3	$2N^3-1$
行列乗算（改良）	N	
CLIQUE	N^3	$2N^3-1$
COLORCOST	N^3	$2N^N-1$

参考文献 [MEAD 80]

図-14 トリーマシンでの計算量

ューリング機械によって，問題のサイズをNとすると，Nの多項式時間内に解けることが分かっている問題のことをいう．シーケンシャルなマシン上でシミュレートして NP 問題を解くには，しばしばNの指数関数時間かかる．）少なくとも NP 問題のある種のものはトリー・マシンの上にうまく写像できるということだけはいえる．図-14にいくつかの例で必要な PE の数，時間を示す[MEAD 80]．表で CLIQUE というのは無向グラフ中の最大次数の完全サブグラフを見付ける問題である．COLORCOST というのは，グラフ中の各ノードに色を塗り，そのとき色についているコストをすべてのノードに対して総和して，それを最小にする問題である．ただし辺を共有するノードには異なる色を塗らなければならない制限がある．CLIQUE も，COLORCOST も NP 問題であり，多項式時間で解くには，トリー・マシン上で多数の PE がいることがわかる．



$$Xout \leftarrow F\ (\overset{(1)}{Xin}, \overset{(2)}{Xin}, Yin)$$
$$\overset{(1)}{Yout} \leftarrow G\ (\overset{(1)}{Xin}, \overset{(2)}{Xin}, Yin)$$
$$\overset{(2)}{Yout} \leftarrow G\ (\overset{(2)}{Xin}, \overset{(1)}{Xin}, Yin)$$

図-15 トリー用の一般化された内積ステップ・プロセッサ

6.2 シストリック・サーチ・トリー

通常のサーチ・トリー（AVL トリー，2-3 トリー）では，挿入，削除，要素かどうかのチェックが O(log n) 時間でできる[AHO 74]．しかし木をバランスさせるためにポインタを付け替えたり，データの動きが局所的でない点が VLSI での実現に向いていない．これを工夫して Leiserson は，トリーをバランスさせるデータの動きが局所的で，しかも O(log n) の処理時間しか要しないシストリック・サーチ・トリーを考案した[LEIS 79]．

6.3 算術式，回帰式の評価

n変数のどんな算術式も高々 $4\lceil \log n \rceil$ レベルのトリーを使って評価することができる[BREN 74]．しかし変数の値を入力するのに O(n) 時間かかってしまう．この入力時間は，回帰式の計算をしているときには，計算時間とオーバラップできるので，トリーのもつ並列処理の機能を生かした高速の計算を行える．それには図-15のようなセルを使うと良い．

このようにトリー構造を持ったマシン上で解くことのできる問題は多い．データの動きを局所的にして，下位のノードで処理を大量に行うようにすることがトリー・マシンの特長を生かすポイントである．

7. シストリック・アルゴリズム アプローチ

ここでは H. T. KUNG に従い，（同一の）簡単なプロセッサをネットワークにして，データが規則正しく循環しているようなシステムをシストリック・システムと呼ぶ．シストリック (systolic) とは，血液循環系において心臓が収縮して血液を体に送り出すことを意味する動詞，SYSTOLE の形容詞である[KUNG78],[KUN

形状	例
一次元アレイ	行列・ベクトル乗算，行列乗算，逆行列計算，フィルタ，畳み込み，DFT，相関，回帰式計算，三角行列ないしは一般の行列からなる一次連立方程式の解，ソーティング，パターンマッチング
二次元正方格子	行列乗算，リレーショナルデータベース演算，緩和法，イメージ処理，ソーティング，グラフに関連する種々のアルゴリズム，一次元アレイを使うシストリックアルゴリズムのビット単位でのインプリメンテーション
二次元六方格子	行列乗算，LU 分解，QR 分解，推移閉包の計算
トリー	サーチ，ソーティング，回帰式の計算

図-16 シストリックアルゴリズムの例

79a],[KUNG 80a]. またシストリックアルゴリズムとはプロセッサのネットワーク形態，プロセッサの機能，データの入出力方法が上で述べたシストリックの定義に沿っているものを言う．

VLSI 上に論理回路を実現する際には配線の容易さ，面積の有効利用が重要であり，それらは通信路の形状により決まる．そこでシストリック・アルゴリズムでは通信路でアルゴリズムを分類する．その結果を図-16 に示す．ここでは一次元アレイ，二次元正方格子，二次元ヘキサゴナル格子，トリーに分類している．通信路はなるだけ規則正しい形状を持ち，短い方が望ましい．以下では通信路にもとづいた分類に従ってアルゴリズムの説明を行う．

7.1 一次元アレイを使うアルゴリズム

一次元アレイはパイプとみなすことができ，パイプラインを使う計算に向いている．データは一方向または二方向に流れる．

・ 一方向パイプライン・アルゴリズム

一方向パイプライン計算は，データがパイプラインを一方向に流れるうちに結果を得るパイプライン計算である．その応用例として行列・ベクトル乗算などが考えられる．図-17 に示すのは，行列・ベクトル乗算のアルゴリズム例である．

一般に互いに独立なおなじタイプの多段計算を含む問題の構造は行列・ベクトル乗算の構造に対応づけることができる．即ち独立な計算の一つ一つは，結果のベクトルの成分に対応し，計算ステップの $y \leftarrow F(a, x, y)$ (F は関数) が行列・ベクトル乗算の内積ステップに対応する．したがって，このような問題は一方向パイプライン・アルゴリズムで解くことができる．

・ 二方向パイプライン・アルゴリズム

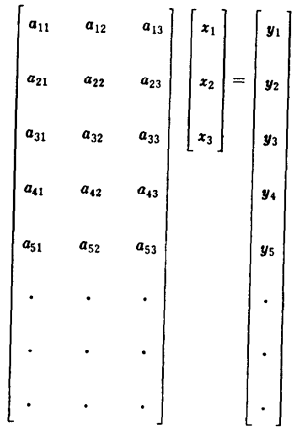
二方向パイプライン計算では入力データが互いに逆向きに流れる．二方向パイプラインの応用例としては帯の幅の最大値はあらかじめ決まっている帯行列・ベクトル積の計算が挙げられる．行列の次元がいくらでも大きくなるとすると，一方向パイプラインでは，入力と中間結果がパイプラインよりも大きくなってしまうが，二方向にデータを流すことにより，データ同志がうまく出合うようにして計算を行える (図-18)．この計算は n を行列の次元とすると $O(n)$ で行える．この計算の特殊な場合として DFT, FIR (Finite Impulse Response filter) も，サンプル点の数を n とすると $O(n)$ で実現できる．また一次元二方向パイプラインの応用例として，多くの応用に現われる回帰式の計算がある．k 次の回帰式の計算とは $X_0, X_{-1}, X_{-2}, \cdots, X_{-k+1}$ をえて，$X_1, X_2, \cdots$ を

$$X_l = R_l(X_{l-1}, \cdots, X_{l-k}) \quad (l>0)$$

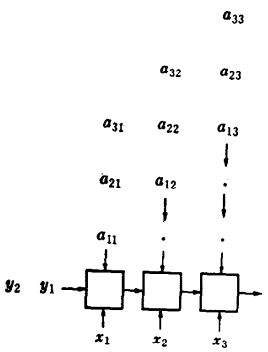
の回帰式に従って計算していくことを意味する．ここで R_l は与えられた回帰関数である．例えば

$$X_l = aX_{l-1} + bX_{l-2} + cX_{l-3} + d$$

を考える．これは図-19 のようにすると実時間で計算

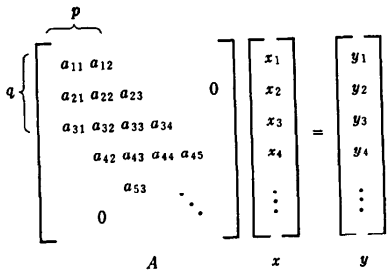


(a) 行列・ベクトル乗算



(b) 一方向パイプラインを使う計算

図-17



(a) ベトルと帯行列の乗算 (p=2, q=3)

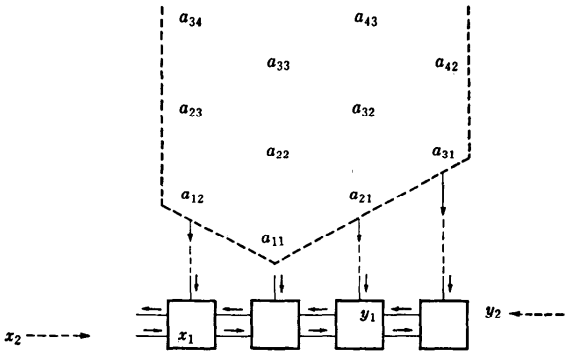


図-18 ベクトルと帯行列の乗算のための一次元シストリック・アレイ [MEAD 80]

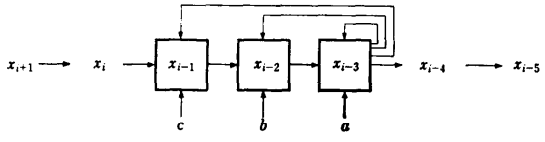
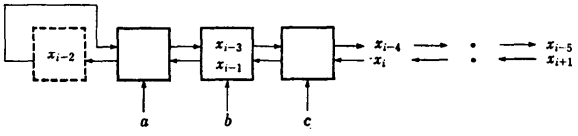
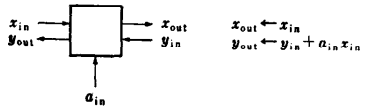


図-19 回帰式を評価するための一次元アレイ（ループあり） [KUNG 79a]



(a) 線形回帰式を評価するための二方向パイプラインを使う一次元アレイ



(b) 内積ステッププロセッサ [KUNG 79a]

図-20

できる．しかし図にあらわれているフィードバック・ループは通信ネットワークを不規則にするので VLSI 向きではない．

ここでは二方向パイプラインを使い，このフィードバック・ループをなくす．それには **図-20** のようにする．一番左側のセルはディレイ・ボックスである．このような方法で，一般に

$$X_i = F_1(a_{i-1}, X_{i-1}, F_2(b_{i-2}, X_{i-2}, F_3(c_{i-3}, X_{i-3}, d_{i-4})))$$

の形の回帰式を計算できる．ここで F_1, F_2, F_3 は関数，a_j, b_j, c_j は回帰式を決めるパラメータである．さらに個々の計算セルに状態変数も持たせて次のような計算を行うことも可能である．

$$X \leftarrow F_{(1)}(a, S, X, Y)$$
$$Y \leftarrow F_{(2)}(a, S, X, Y)$$
$$S \leftarrow F_{(3)}(a, S, X, Y)$$

一次元二方向パイプラインの応用例はこれだけに留らず，順序付きキューも含む．順序付きキューと言うのは，挿入，消去，最小値を取り出す操作のできるデータ構造を一般にこう呼ぶ．通常のシーケンシャル・コンピュータ上のプログラムでは普通の場合，平衡木を使い，順序付きキューを実現する．これを操作するのに必要な時間は，データの数を n とすると $O(\log n)$ である．これを一次元二方向パイプラインを使って実現すると，操作の手間は $O(1)$ になる．その原理は一次元に並んだセルが隣同志でデータの比較交換をして，順序を保つようにすることにもとづく．この順序付きキューでは各操作は次のようにして実現される．

・挿入：一端から重みを付けたデータを入れてやると，重みに従う正しい位置に動いていくはずである．データを一端に入れるだけで，ネットワーク中で自動的に処理されるので，処理の手間は $O(1)$ で済む．

・削除：特定の重みを付けた「反データ」(anti-element) を入れてやると，望む位置までうごいていって，データと出合うので，そのデータをそのセル上で消去する．空白は他のデータの移動で埋められる．「反データ」を入れる手間は $O(1)$ である．

・最小値を取り出す操作：最小のデータは常に一端にあるので，それを取り出すには $O(1)$ の

時間で済む.

このような順序付きキューの応用例としては，n回の挿入の後でn回の最小値を取り出す操作を行って，$O(n)$時間でn個のデータをソートすることが挙げられた[LEIS 79],[GUIB 82].

7.2 2次元アレイを使うアルゴリズム

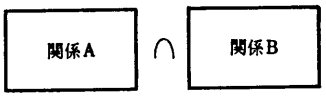
2次元アレイとしてここで取り上げるのは，2次元正方格子と2次元ヘキサゴナル格子である.

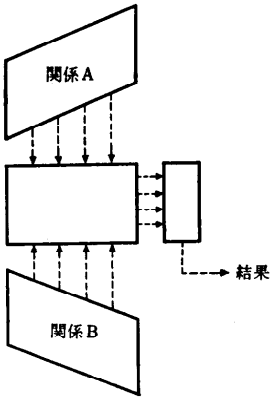
2次元正方格子を使うアルゴリズムの研究は cellular automata の研究に始まる．しかし，初期の cellular automata の研究はデータ移動の規則正しさ，cell の簡潔さ，一様性を強調して考えてはいないのでVLSI で直接実現するものには向いていないことが多かった．実際の応用例としては Illiac-IV にみられるような正方格子を使ったアレイ・プロセッサがあげられる．この場合プロセシング・エレメント (PE) 間の広域通信機能が，行列計算における緩和因子と収束の判断に有効である．収束の判断の場合，ほとんどすべての PE から，制御を行っている PE への通信が必要となるからである．緩和手法は PE がピクセルに対応するイメージ処理でも使われる．ただし広域通信は VLSI 向きではない．このような2次元正方格子アレイの他の応用例としてはパターン認識，パターン・マッチング（これのための LSI は実際に作られている．[FOST 80] 参照），グラフ理論，スイッチング理論，関係データベースなどがある.

- 関係データベース用プロセッサ

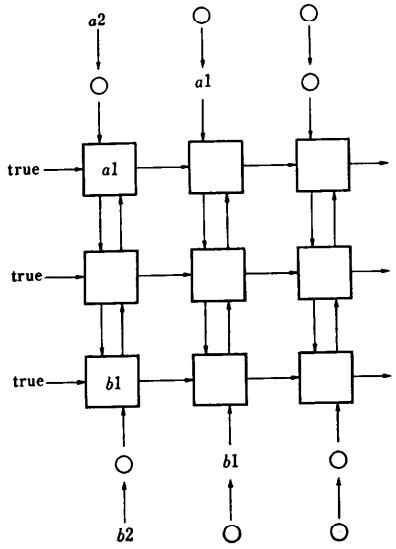
関係データベースはデータベースの種々の演算を集合演算とみなして，データ表現によらない演算の一様な取り扱いを許す．データの追加などが容易という長所をもつが，データが多いときには，他のデータモデルにもとづくデータベースと同じく，処理時間の増加に悩ませられる．ここでは，2次元正方格子アレイを使って関係データベースの比較，共通部分，結合などの演算を高速にするアイデアを紹介する[KUNG 79].

シストリック・アレイによる共通部分の計算の例を**図-21** に示す．それぞれの関係（リレーション）はm個の列からできている m-tuple の集合とする．Aの大きさを n, B の大きさを n'とする．($n \geq n'$ と仮定する.) 図-21のような構成にすると，プロセシング・エレメントの数は $(n+n'-1)\cdot m$, 処理ステップ数は $3n+n'+m-3$ である．共に $O(n+n')$ である．その他の演算についても，**図-22** に示したような時間と PE の数で処理できる．表中の改良とは，PE を共有した





(a) 関係データベースでの共通部分を取り出す演算



(b) 関係データベースの共通部分計算用アレイとプロセッサ

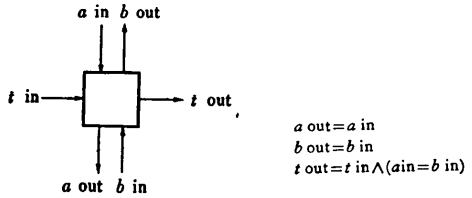


図-21

演 算	PE の数	ステップ
比 較	$(n+n'-1)\cdot m$	$3n+n'+m-3$
比較（改良）	$n'\cdot m$	$n+n'+m-1$
Intersection	$(n+n'-1)(m+1)$	$3n+n'+m-2$
Intersection（改良）	$n'\cdot(m+1)$	$n+n'+m$
Remove-duplicate	$(2n-1)\cdot(m+1)$	$4n+m-2$
Remove-duplicate（改良）	$n\cdot(m+1)$	$2n+m$
division	$n''\cdot(n'''+2)$	$n+n'+1$
Join (k 列)	$(n+n'-1)\cdot k$	$2n+n'+k-2$
Join（改良）	$n\cdot k$	$n+n'+k-1$

$|A|=n$, $|B|=n'$, tuple は m 列からなる．
n'' は A 中で相異なるデータの数，n''' は B 中で相異なるデータの数 [KUNG 79 b]

図-22 関係データベース演算の PE の数と速度

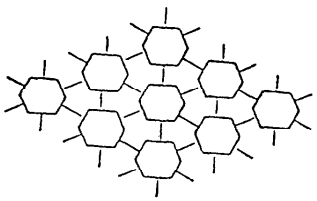


図-23 3×3 の六方格子

- 行列乗算
  $C_{ij}^{(1)}=0$
  $C_{ij}^{(k+1)}=C_{ij}^{(k)}+a_{ik}\cdot b_{kj}$
  $C_{ij}=C_{ij}^{(n+1)}$
- LU 分解
  $a_{ij}^{(1)}=a_{ij}$
  $a_{ij}^{(k+1)}=a_{ij}^{(k)}+I_{ik}\cdot(-u_{kj})$

$$I_{ik}=\begin{cases}0 & i<k\\ 1 & i=k\\ a_{ik}^{(k)}\cdot u_{kk}(-1) & i>k\end{cases}$$

$$u_{kj}=\begin{cases}0 & k>j\\ a_{kj}^{(k)} & k\leq j\end{cases}$$

- 推移閉包
  $a_{ij}^{(1)}=a_{ij}$
  $a_{ij}^{(k+1)}=a_{ij}^{(k)}+a_{ik}^{(k)}\cdot a_{kj}^{(k)}$

図-24 2次元ヘキサゴナル格子を使うアルゴリズムの回帰式

演 算	形 状	PE の数	ステップ
帯行列・ベクトル積 n 次元，帯幅$=W$	1次元	W (n)	$2n+W$ $(3n)$
帯行列の乗算 n 次元，幅$=W_1$, W_2	2次元 ヘキサゴナル	$W_1\cdot W_2$ (n^2)	$3n+\min(W_1, W_2)$ $(4n)$
帯・三角行列の方程式を解く n 次元，幅$=W$	1次元	W (n)	$2n+W$ $(3n)$

（括弧の中に示した値は帯行列が密になって幅が n になった場合の値を示している．）

図-25 シストリックアルゴリズムによる行列計算

りして PE の数を減らす工夫を凝らしたことを意味する．データの数に比例した高速な演算が期待できることが分かる．しかし，データの数が多いときには，PE の数が多くなり過ぎるかもしれない．必要 PE の数よりもデータが多いときに，データを分割して処理することも議論されており，それによればデータの分割をうまく行えば，限られた数の PE を使いデータの大きさに比例した処理も可能である．

- 2次元ヘキサゴナル格子を使うアルゴリズム

ヘキサゴナル格子は（図-32）三つの方向に対称性をもっていて，計算結果と二入力を対称的に送り出せる．ヘキサゴナル格子を使った応用には，行列乗算，LU 分解，推移閉包，QR 分解などの（同一の構造をもつ）計算がある．

これらは，どれも似たような回帰式にもとづいて計算がなされている．それを図-24 にまとめた．表の式から分かるようにどれも似た形をしているため，2次元ヘキサゴナル格子を使い，殆どおなじような PE を使うことによって上記の例を計算することができる．

図-25 に行列の演算についてシストリック・アレイを使った場合の時間，PE の数を示した．

8. 今後の動向，課題

今後の課題，動向について以下まとめる．

8.1 汎用コンピュータシステムと専用コンピュータシステム

VLSI の応用には大きく分けて二つのアプローチがある．一つは，汎用プロセッサを使い，特定の応用分野に合ったプログラム言語を使って問題の解決にあたるアプローチである．VLSI を使った CPU と，高解像度ディスプレイ，ローカルディスク，ネットワークインタフェースを持った，現在よりもはるかに強力なパーソナルコンピュータが広く使われるようになろう．しかし，このアプローチにはコストパーフォーマンス上昇の限界がある．つまり汎用化からの脱却化を強め，それぞれの問題への適応化を強めるあまり命令が複雑になり，論理回路，通信回路，マイクロプログラムメモリの増大がおこる．また通信のディレイが大きくなり，複雑さがある点を越えると，コストパーフォーマンスはむしろ減少する．

ところでもう一つのアプローチはシストリック・アレイのような専用化したプロセッサを大量につかう方法である．シストリック・アルゴリズムおよびトリー・アルゴリズムのような VLSI 向きアルゴリズムで多くの問題を解くことができる．アレイとトリー，またはその組み合わせによる通信形状は多くの問題に適応できる．しかしこのアプローチの実際的な欠点は個々の応用問題ごとに，最適の通信形態を考えて，PE の機能の設計をして行かなければならないことで

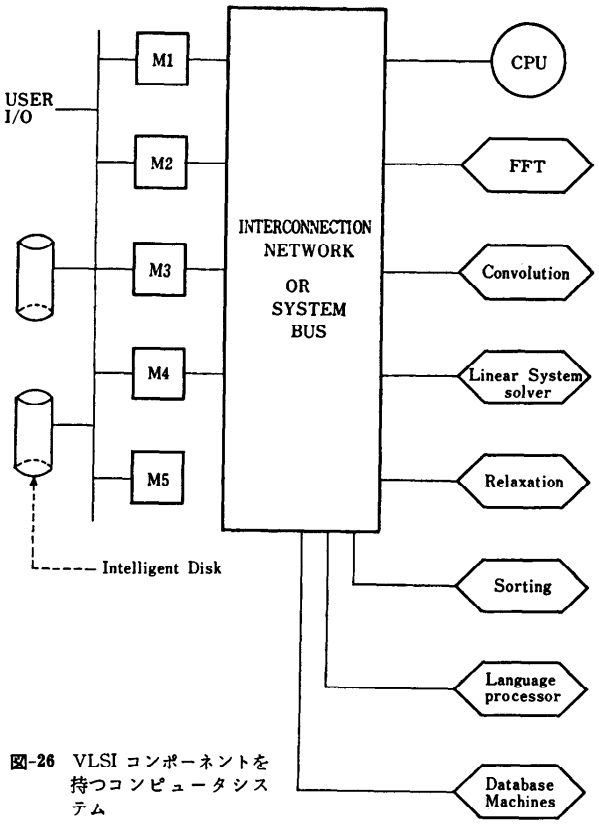


図-26 VLSI コンポーネントを持つコンピュータシステム

ある．そのため現在では信号処理のような汎用性のあるごく一部の分野でしか，応用の研究が成されていない．そこで今後は個々の問題に対してシストリック・アレイ，トリー・マシンなどを応用する際に最適の通信形態をえらびだす手間を減らすことが重要になる．

8.2 問題のサイズの巨大化への対応

大きな問題を解くにはひとつのチップでは解けない．よってチップをネットワークにするかまたは，問題そのものを分割する必要がでてくる．ネットワークのサイズよりも大きな問題をどのように分割して取り扱うかの考察，あるいはまた分解を容易にするようなアルゴリズムの考案も重要である．

8.3 VLSI 構成要素からシステム構成まで

また VLSI チップで一つのアルゴリズムを実現するようになると，それらのチップは計算機システムの構成要素として使われることになろう．よってそのような構成要素を多くもったシステムの構造をどのようにするかが重要な課題となる．そのようなシステムの持たなければならない機能としては，問題をシステムの構成要素に割り当てること，問題が大き過ぎるときにはそれを分割して解き，結果を統合することなどがあげられる．システム・レベルで VLSI チップをどう使うかも課題となる．(図-26 参照)

8.4 回帰式からチップの自動生成

シストリック・アルゴリズムと問題に内在する回帰性とは密接に対応している．図-27 に回帰式の例を示す．VLSI チップを作る際の省力化の一つの方法としては，回帰式で記述できる問題に対しては自動的にマスクパターンを作ることができないだろうかということがある．つまり図-28 の手順が自動的に行われてシストリック VLSI チップが作れないだろうか等の課題がある．

8.5 VLSI アルゴリズムを記述するための並列言語

VLSI アルゴリズムの大半は，並列性を生かした処理を行う．そのようなアルゴリズムを記述するための記法を考案しなければならない．またそのような記述にもとづくアルゴリズムの正しさの証明も行いたい．記述言語の備えるべき性質をあげてみる。

データ $a_1,\ a_2,\cdots$
$b_1,\ b_2,\cdots$

- フィルタ
  $a_1b_1+a_2b_2+a_3b_3+\cdots$
  $a_1b_2+a_2b_3+a_3b_4+\cdots$
  $a_1b_3+a_2b_4+a_3b_5+\cdots$
- パターン・マッチング
  $a_1=b_1\wedge a_2=b_2\wedge a_3=b_3\wedge\cdots$
  $a_1=b_2\wedge a_2=b_3\wedge a_3=b_4\wedge\cdots$
  $a_1=b_3\wedge a_2=b_4\wedge a_3=b_5\wedge\cdots$
- 畳み込み
  a_1b_1
  $a_1b_2+a_2b_1$
  $a_1b_3+a_2b_2+a_3b_1$
- DFT
  $a_1b_1+a_2b^2{}_1+a_3b^3{}_1+\cdots$
  $a_1b_2+a_2b^2{}_2+a_3b^3{}_2+\cdots$
  $a_1b_3+a_2b^2{}_3+a_3b^3{}_3+\cdots$
- 回帰式の一般形

フィルタ，パターン・マッチング

$$R_i^{(m+1)}=R_i^{(m)}\oplus a_m\odot b_{i+m}$$

畳み込み

$$R_i^{(m+1)}=R_i^{(m)}\oplus a_m\odot b_{i-m}$$

DFT

$$R_i^{(m+1)}=R_i^{(m)}\oplus R_i^{(m)}\odot b_i$$

制限：$S_R(V_a-V_b)=S_a(V_R-V_b)=S_b(V_a-V_R)$
S：間隔　V：転送速度　R…結果

図-27 回帰式の例

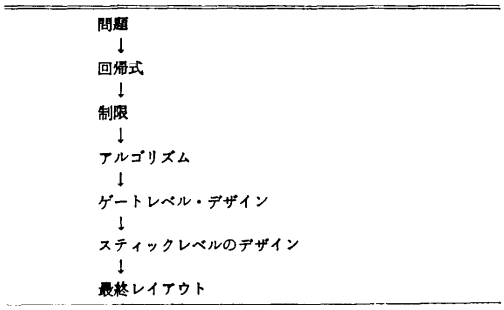


図-28 回帰式からマスクパターンまでの一貫設計

- ネットワークをデータの流れの方向，速さ，形状によって記述できること．
- アルゴリズムの性質がよく分かり，正しさの形式的証明ができること．
- 新しいアルゴリズムの発見に導けるように自由に表現の変更ができること．

このような記述言語をつくることが必要である．このような記述言語の一つとして，波面(wavefront)の概念にもとづいた記述言語が提案されている[KUNG 82], [WEIS 81]．今後このような言語を CAD システムにつなげていくことが課題となっていくだろう．

8.6 VLSI 計算の理論の構築

VLSI で新たな問題として登場した配線面積の増加を防ぐ問題などを系統的に解決する必要もあろう．現在たとえば

仮定 1 平面上のある1点の上を通過するワイアの数には上限がある．ワイアの幅は長さの1単位幅であるとする．

仮定 2 1単位時間中にワイア上を1ビットの情報を送ることができる．

仮定 3 チップの形状は凸形である．

などを仮定して，N個のノードを持つシャッフル交換ネットワークには $O(n^2/(\log n)^2)$ の面積が必要であること，Nビット乗算チップにはTを必要な時間とすると，$O(n^2/64\ T^2)$ の面積が必要であることなどが知られているが，この理論をおしすすめ VLSI 計算の理論の構築を行うことが将来の課題である[BREN 81], [TOKU 82]．

参 考 文 献

[AHO 74] Aho, A. V., Hopcroft, J. E. and Ullman, J. D.: The Design and Analysis of Computer Algorithms, Addison-Wesley, Reading, Mass.

[AUSL 82] Auslander, M. and Hopkins, M.: An Overview of the PL. 8 Compiler, SIGPLAN 82 Symposium on Compiler Construction, Boston, Massachusetts, pp. 22-31 (June 1982).

[BAL 82] Bal, Subhash et al.: The NS 16000 Family-Advances in Architecture and Hardware, IEEE COMPUTER, Vol. 15, No. 6, pp. 58-68 (June 1982).

[BERE 82] Berenbaum, A. et al. The Operating-System and language support features of the BELLMAC-32. Symp. on Architectural Support for Programmig Language and Operating Systems, March 1982, ACM.

[BREN 74] Brent, R. P.: The parallel Evaluation of General Arithmetic Expressions, Journal of the ACM, Vol. 21, No. 2, pp. 201-206 (Apr. 1974).

[BREN 81] Brent, R. P and Kung, H. T.: The Area-Time Complexity of Binary Multiplication, JACM Vol. 28, No. 3, pp. 521-534 (July 1981).

[CHAI 82] Chaitin, G. J.: Register Allocation and Spilling via Graph Coloring, SIGPLAN 82 Symposium on Compiler Construction, Boston, Massachusetts, pp. 98-107 (June 1982).

[CLAR 80] Clark, W. A.: From Electron Mobility to Logical Structure: A View of Integerated Circuits. Computing Survey, Vol. 12, No. 3, pp. 323-356 (Sept. 1980).

[CLAR 80] Clark, D. W. and Strecker, W. D.: Comments on The Case for the Reduced Instruction Set Computer, Computer Architecture News, Vol. 8, No. 6, pp. 34-38 (Oct. 1980).

[COMP 81] Digest of Papers, COMPCON '81, Spring, IEEE, およびその場での口頭による発表，質問による．

[DESP 78] Despain, A. M. and Patterson, D. A.: X-TREE: A Tree Structured Multiprocessor Computer Architecture, Proc. 5th Annual Symposium on Computer Architecture (Apr. 1978).

[FITZ 81] Fitzpatrick, D. et al.: VLSI Implementations of a Reduced Instruction Set Computer, VLSI Systems and Computations, Carnegie Mellon Univ. Conference, Computer Science Press, pp. 327-336 (Oct. 1981).

[FOST 80] Foster, M. J. and Kung, H. T.: The design of special purpose chips. IEEE COMPUTER pp. 26-40 (Jan. 1980).

[GUIB 82] Guibas, L. J. and Liang, F. M.: Systolic Stacks, Queues, and Counters. Proc. Conference on Advanced Research in VLSI, MIT, pp. 155-164 ARTECH HOUSE, Inc. (Jan. 1982).

[JOHN 80] Johnson, R.C.: Intel's 32-bits three-chip sets promises software-transparent multiprocessing systems, Electronics. pp. 42-44 (Nov. 6, 1980).

[JOHN 81a] Johnson, R.C.: 32-bit microprocessors inherit mainframe features, Electronics p. 138 (Feb. 24, 1981).

[JOHO 81] 小特集: VLSI の CAD, 情報処理, Vol. 22, No. 8 (Aug. 1981).

[KNUT 73] Knuth, D.E.: The Art of Computer Programming, Vol. III: Sorting and Searching, Addison-Wesley, Reading, Mass.

[KUNG 79 a] Kung, H.T.: Let's Design Algorithms for VLSI, CMU Technical Report, CMU-CS-79-151.

[KUNG 79 b] Kung, H.T. and Lehman, P.L.: Systolic (VLSI) Arrays for Relational Database Operations, CMU Computer Science Department Technical Report (1979).

[KUNG 80 a] Kung, H.T.: Notes on VLSI Computation, Crest Parallel Processing Systems Course, Loughborough, England (1980).

[KUNG 80 b] Kung, H.T.: The Structure of Parallel Algorithms, in Advances in Computers, Vol. 19.

[KUNG 82] Kung, S.Y., Gal-Ezer, R.J. and Arun, K.S.: Wavefront Array Processor: Architecture, Language and Applications, 1982 Conference on Advanced Research in VLSI, MIT, ARTECH HOUSE, INC., pp. 4-19 (Jan. 1982).

[LEIS 79] Leiserson, C.E.: Systolic Priority Queues, Proc. Caltech Conference on VLSI (Jan. 1979).

[MEAD 80] Mead and Conway, Introduction to VLSI system, Addison Wesley (1980).

[PATT 80] Patterson, D.A. and Ditzel, D.R.: The Case for the Reduced Instruction Set Computer, Computer Architecture News. Vol. 8, No. 6 (Oct. 1980).

[PATT 81] Patterson, D.A. and Sequin, C.H.: RISC I: A Reduced Instruction Set VLSI Computer, in Proc. 8th Annual Symposium on Computer Architecture, ACM SIGARCH, pp. 443-457 (May 1981).

[PATT 82 a] Patterson, D.A. and Richard, S.: Piepho. RISC ASSESSMENT: A High-Level Language Experiment. Proc. 9th Int'l. Symp. Computer Architecture, pp. 3-8.

[Patt 82 b] Patterson, D.A. and Sequin, C.H.: A VLSI RISC, IEEE COMPUTER, Vol. 15, No. 9, pp. 8-21 (Sept. 1982).

[POLL 82] Pollack, F.J. et al.: Supporting Ada Memory Management in the iAPX-432, Symp. on Architectural Support for Programming Languages and Operating Systems (Mar. 1982).

[RADI 82] Radin, George.: THE 801 MINICOMPUTER, Symp. on Architectural Support for Programming Languages and Operating Systems (Mar. 1982).

[RATT 81] Rattner, J. and Lattin, W.W.: Ada determines architecture of 32-bit microprocessor, Electronics (Feb. 24, 1981).

[SCIE 77] Scientific American, Vol. 237, No. 3, 1977, Sept. Special issue on Microelectronics, その日本語訳マイクロエレクトロニクス日本経済新聞社.

[SEIT 82] Seitz, C.L.: Ensemble Architectures for VLSI —A Survey and Taxonomy, Proc. Conf. on Advanced Research in VLSI, ARTECH HOUSE, Inc. MIT. pp. 130-135 (Jan. 1982).

[SEQU 78] Sequin, C.H. et al.: Communication in X-tree, A Modular Multiprocessor System. Proc. ACM National Conference (Dec. 1978).

[SHER 82] Sherburne, R.W. et al.: Datapath Design for RISC, Proc. Conf. on Advanced Research in VLSI, ARTECH HOUSE, Inc. MIT. pp. 53-62 (Jan. 1982).

[STONE 71] STONE, H.S.: Parallel processing with the perfect shuffle, IEEE Trans, Comput. C-20, pp. 153-161 (1971).

[STONE 75] Stone, H.S.: Parallel Computation in Introduction to Computer Architecture (H. Stone, ed.), pp. 318-374, Science Research Associates.

[SUTH 77] Sutherland, I.E. and Mead, C.A.: Microelectronics and Computer Science, Scientific American, 237(3): 210-228, Sept. 1977. その日本語訳サイエンス日本経済新聞社.

[SWAN 77] Swan, R.J., Fulltr, S.H. and Siewieorek, D.P.: Cm* —A Modular Multiprocessor, AFIPS CONFERENCE PROCEEDINGS 46 (1977).

[TOKU 82] 都倉信樹: VLSI アルゴリズムおよび面積時間複雑度, 情報処理, Vol. 23, No. 3, pp. 176-186 (1982).

[WEIS 81] Weiser, U. and Davis, Al.: A Wavefront Notation Tool for VLSI Array Design, Conference on VLSI Systems and Computations, Carnegie-Mellon Univ., Computer Science Press. p. 226 (Oct. 1981).

[YAU 77] Yau, S.S. and Fung, H.S. Associative Processor Architecture—A Survey Computing Surveys, Vol. 9, No. 1 (Mar. 1977).

(昭和57年9月27日受付)