計算機アーキテクチャ59−9
オペレーティング・システム29−9
(1985・10・24)

科学技術計算用データ駆動計算機 ＳＩＧＭＡ−１のＬＳＩ化構成技術

A LSI Architecture of the SIGMA−1: A Data Driven Computer for Scientific Computations

平木　敬　西田健次　関口智嗣　島田俊夫
Kei Hiraki　Kenji Nishida　Satoshi Sekiguchi　Toshio Shimada
電子技術総合研究所
Electrotechnical Laboratory

1．はじめに

科学技術計算用データ駆動計算機ＳＩＧＭＡ−１は、数値計算を高速に実行することを目的とした大規模データ駆動計算機である。データ駆動アーキテクチャはプログラムの持つ並列性を自然に利用する可能性を持つアーキテクチャとして様々な研究が行われてきている【１、２、３】。我々は既にＳＩＧＭＡ−１予備試作機を完成し、各種プログラムを実行してきた【４、５、６】。本論文ではこの経験を踏まえ、処理装置台数の増加したシステムで実用規模の計算を実現するための問題点および我々が選択した解決のための方策について述べる。

実用的な並列計算機を構築するために解決しなければならない問題点として、同期、負荷分散、接続ネットワーク、直列プログラム部分の高速化を挙げることができる。第１の問題点である同期とは、並行して動作するプロセス間および処理装置とメモリ間等の同期を如何にオーバーヘッドを少なく取るかということである。データ駆動アーキテクチャではデータ依存関係に基づく同期はハードウェアで自動的に取られオーバーヘッドを産出しない。しかしながら、データ依存関係とは独立の同期関係をデータ依存関係だけで記述することには多大の困難がある。もっとも、データ駆動計算機が１台しかない場合には１度に１命令ずつしか実行されないため、実行順序はデータ依存関係により規定される半順序より強い条件で順序づけ可能である。しかしながら、多数台が同時に実行する環境では、並行に実行されるプログラム部分について、実行順序関係は１台で実行する場合と比較してより弱くしか規定し得ない。この結果、多数台を結合して使用するための命令セットと、命令１個だけでは実現不可能な同期基本操作の実現が問題となる。本論文第２節では、ＳＩＧＭＡ−１における同期および順序制御について述べる。

第２の問題点である負荷分散とは、各要素プロセッサを平均的に稼働させ、最短実行時間を得るための負荷分散を行う方式である。プログラムを処理装置に分割して実行を行う場合、実行時間はもっとも負荷の重い処理装置で決定される。したがって、負荷分散の片寄りは実行時間の大きな遅れとして反映する。ＳＩＧＭＡ−１ではネットワークを用いた負荷分散方式【７】を採用し、最適状況に近い負荷分散を行う。

第３の問題点、ネットワークとは、並列計算機システムの性能は多くの場合ネットワーク性能により決定されることである。ネットワークの問題点は更にネットワーク自身の性能と、ネットワークに起因する遅れ時間の待ち合わせに分けられる。データ駆動アーキテクチャにおいては後者は問題にならない。前者については、現在までのところ、クロスバー・スイッチが最も理想に近いネットワークである。ＳＩＧＭＡ−１ではＬＳＩ化する利点を生かしたネットワークにより、この問題点を解決している。

ＳＩＧＭＡ−１は最終的には２００台規模の構成となるため、予備試作機でのＭＳＩを使用した実現手法では、大きさ、消費電力、信頼性等の困難に直面する。従ってＬＳＩを用いた実現手法でこれらの問題点を解決する必要ある。ＬＳＩ化した実現手法は商用計算機では既に一般的なものになってきているが、ＳＩＧＭＡ−１をはじめとする実験機においてはコスト、設計工数などの点からＬＳＩ化、特にカスタムまたはセミカスタムＬＳＩ化は殆ど行われなかった。第３節ではこれらの諸問題点およびネットワークを含めたＬＳＩ化実現手法について述べる。

ＬＳＩ化した実現手法を採用すると、殆どすべての論理回路について直接動作検証することが不可能になる。またＳＩＧＭＡ−１は全体で数千万ゲート規模となるため、ハードウェアのメンテナンス、ソフトウェアのデバグにおい

ても従来から用いられてきた手法では多大な時間を消費する。この問題点を解決するため、ＳＩＧＭＡ－１では並列化したメンテナンス・アーキテクチャを採用した。第４節ではこれらメンテナンスおよびデバグの実現方式について述べる。

データ駆動計算機は、基本的には、プログラムの並列実行に大変都合の良いアーキテクチャであり、並列化に伴う性能低下が少ないと考えられてきている。しかしながら、並列性の高いプログラムでさえ直列に近い状況で動作するプログラム部分を含む。プログラムの直列部分及び並列部分の立ち上がりでは命令の設計が性能を左右する。第５節ではＳＩＧＭＡ－１の命令設計について述べる。

本節の最後に、ＳＩＧＭＡ－１アーキテクチャの概略を説明する。ＳＩＧＭＡ－１全体のアーキテクチャは図１に示すように、２階層のネットワークで相互に結合した演算処理装置（ＰＥ）および構造体処理装置（ＳＥ）で構成される。ＰＥは構造体記憶以外の全ての命令実行、データ処理を担当する処理装置であり、図２に示されるように２段のサーキュラ・パイプライン構成である。ＳＥは構造体記憶を実現するための処理装置であり【８】、図３に示すように入力ＦＩＦＯキュー（ＳＢユニット）および非同期配列記憶（Ｓユニット）で構成される。

これらＰＥ４台とＳＥ４台をローカルネットワークで結合して、１個のＰＥ／ＳＥグループを作成する。グループはＳＩＧＭＡ－１におけるプロセス（サブルーチン）実行の単位であり、１グループがあたかも１台の高能力プロセッサであるかのように使用する。従って合計８台のＰＥ／ＳＥ間の結合は、非常に密であり且つ高速である事が要請される。ＳＩＧＭＡ－１ではこの目的のため、１０　Ｘ　１０クロスバースイッチによるローカル・パケット転送ネットワークを用いる。なおグローバルネットワークにも同一素子でオメガ網を構成して使用する。

２．同期と実行順序の規定

無限に並列なデータ駆動計算機ではプログラムのもつ半順序的データ依存関係だけが実行順序を規定する。この性質を生かすために、データ駆動計算機では、単一代入型の言語【９】を専ら使用してきている。一方、データ駆動計算機を自立したシステムとして動作させるためには、資源の分配、相互排他、制御された並列度の現出、実行制御（プログラム部分の終了確認）など、半順序的データ依存関係の枠を越えた非決定的プログラムが必要である。これらの非決定的問題を実行するためには、発火規則を拡張すること、または単一代入の枠を越えることが必要であり、その記述言語を含め、未解決の問題が多い。

非決定的データフロープログラムの研究は【１０、１１、１２、１３】などで行われ、基本的枠組を示してきた。非決定的プログラミングを実現するための基本的問題は、順序関係の保存と、直列化の機構にある。従来提案されてきた方式としては、操作が許可されるまで繰りかえし操作を再実行する方式、タグのマッチングに順序関係を持ち込む方式などが知られている。しかしながら、不必要な繰りかえしによる性能低下、またはマッチング機構が複雑となり実現が困難であるなどの点が指摘される。

ＳＩＧＭＡ－１では、マッチング方式として、多重マッチを許す方式を採用し、同時に順序関係を、ネットワーク、多重マッチング、処理装置を通して、データ依存関係以外は保存するアーキテクチャを用いるため、より簡単な方式で非決定的プログラミングが実現する。

ＳＩＧＭＡ－１マッチングユニットの特徴として、多重マッチ機能がある。多重マッチ機能としては、１個のトークン入力に対して１個のマッチングユニットに既に入っているトークンを発火させる場合と、マッチングユニット内に存在する同一タグのトークン全てを発火させる場合があ

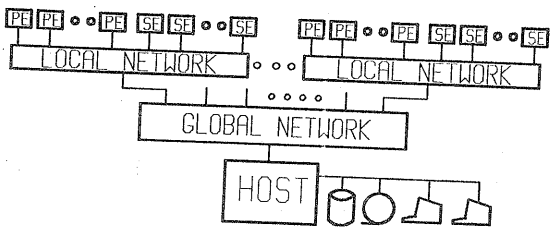


図１．　ＳＩＧＭＡ－１全体構成

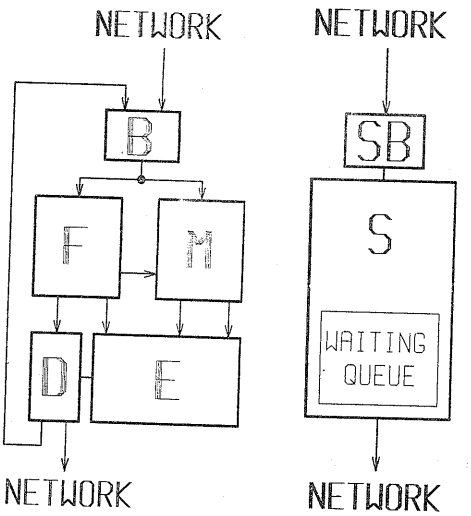


図２．　演算処理装置（ＰＥ）図３．　構造体処理装置（ＳＥ）

る。また、発火させた入力トークンをマッチングユニットに残すことと、消去することの双方が可能であり、また発火したマッチングユニットに既に入っていたトークンを残す事も、消去することも可能である。これらの制御はすべてトークンのＦＬＧフィールドによって指定される。また、或る種の問題では、マッチングユニット上でのトークンの零テストが必要である。ＳＩＧＭＡ－１では零テストのために、【１２】のFETCH or ABORT および SAVE or ABORT と同等の機能も持っている。

順序保存の機能としては、（１）多重マッチで１個の入力トークンにより発火可能なタグを持つトークンが複数個マッチングユニット内に存在する場合に、ＦＩＦＯの順、即ちマッチングユニットに挿入された順番で発火すること、（２）ネットワークはユニークなパスを持つ方式を使用するため、追い越しが発生せず、また、処理装置中のバッファなどでも追い越しが発生しないことにより、順序の保存が行われる。具体的には、図４に示すようにデータフローグラフ上のノードＡでトークンを出力する順とＣで受けとる順が一致する。この結果、演算が完全に非決定的に発火するモデルと比較して同期基本操作の実現などが容易になる。

ここでは非決定的プログラムの典型的な例として、同期基本操作の実現を示そう。

同期問題の最も簡単な例として、図５に生産者消費者問題を示す。ＳＩＧＭＡ－１では先に述べたように、ａｒｃ上に任意の個数のトークンを置くことが可能である、ＦＩＦＯ的サービスが保証されるので、容易にインプリメント可能である。図にはバッファサイズがｎの場合を示す。

相互排他問題はサブグラフ、中間結果の共有など様々な局面で必要な基本操作である。図６ではＳＹＮＣ＿Ｌの左ａｒｃを無限バッファとして使い、直列化している。

３．ＬＳＩ化したアーキテクチャ

ＳＩＧＭＡ－１予備試作機は先に述べたように全てＭＳＩで構成され、約１９００個のＩＣを８枚のプリント・ボードで実装している。しかし、このままの姿で２００台規模実装することはプリント基板枚数、大きさ、消費電力、信頼性、保守性等が原因となり困難である。従ってＳＩＧＭＡ－１システム実現のためにはＬＳＩ化は不可欠の条件である。

ＬＳＩ化という観点から見たデータ駆動アーキテクチャは、フォンノイマン・アーキテクチャと比較して、

（１）レジスタ等の記憶要素を持たないため、処理装置の構成要素が論理回路部分とメモリ部分に明確に分離可能なこと、

（２）アーキテクチャを構成する各パイプライン要素が互いに副作用を介して干渉しないため、独立性を持って設計可能であること、

（３）処理装置間の結合がパケットの転送だけで行われるため、簡易であること、

（４）２入力命令では１個の命令処理に、３回のマッチング・メモリ読み出しおよび１回の書き込み、命令読み出しの計５回のメモリ・アクセスを行い、レジスタを用いたフォンノイマン計算機における２回と比較して、倍以上多いこと、

（５）プログラムにおける並列度の制御が不能なため、プロセッサ数より多い並列度を吸収するためのバッファがかなり多量に必要であること、

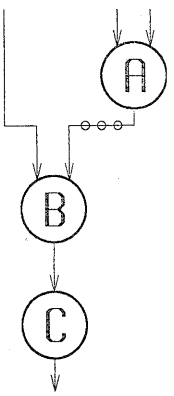


図４．　パケット到着順序保存

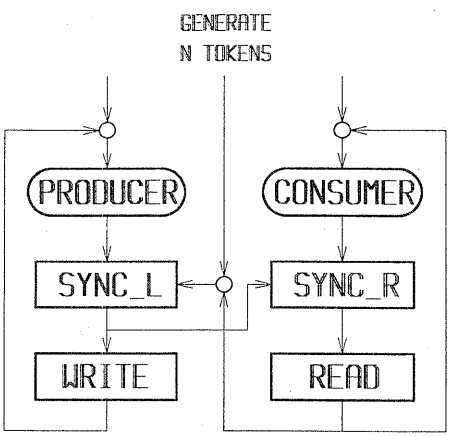


図５．　生産者消費者問題の実現

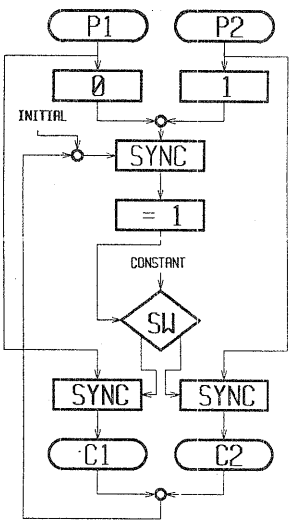


図６．　相互排他問題の実現

が特徴として列挙される。
このなかで（1）～（3）はＬＳＩ化にとって有利な条件である一方、（4）～（5）はフォンノイマン計算機より広いバンド幅でメモリと接続する必要を示唆する。

データ駆動アーキテクチャをＬＳＩ化する手法にはアーキテクチャを簡略化し、1チップＬＳＩとメモリで構成する方法、ゲートアレイを使用して基本アーキテクチャを忠実にＬＳＩ化する方法、市販ビットスライスＬＳＩ、マイクロプロセッサなどを使用する方法が検討可能である。

現時点でのＬＳＩ技術では、マッチングメモリ、バッファメモリを含め全てを1チップに格納することは不可能であり、この両者は外部メモリで実現しなければならない。また、要求されるメモリとの広い転送量を実現するためには、複数組のメモリ接続が不可欠である。また現時点の技術では1チップにするためにかなりの機能簡略化を行う必要があり、現実的とはいえない。更に２００台規模の台数では、専用化した1チップで構成することは経済的およびチップ設計の工数の点からもメリットが少ない。

既存のＬＳＩまたはマイクロプロセッサを活用する方式は、浮動小数点演算器、汎用算術論理回路等フォンノイマン計算機と共通な部分では使用するメリットが多い。しかし、マッチングメモリ、命令アドレス管理、出力パケット生成などデータ駆動特有なパケット転送を基礎とする機能要素は、フォンノイマン計算機を構成することを目的としたマイクロプロセッサ、ビットスライスＬＳＩと相性が悪い。更に消費電力、実装面積が大きくなり且つ保守機能組込みが困難である。

以上の点を鑑みデータ駆動計算機は、現時点の技術では、構成要素の各ユニットごとにゲートアレイを基本構成要素として使用することが好適と判断される。商用計算機においては既にゲートアレイを構成要素とすることは一般的になって来ている。しかしながら、実験機レベルのアーキテクチャをＬＳＩ化すると、台数が少ないことによりコスト上昇が懸念される。さらに一旦ＬＳＩ化すると修正が出来ないこともＬＳＩ化を妨げる要因である。

ＳＩＧＭＡ－１では、第1の問題点は、２００台規模であることに加え、大部分のＬＳＩをビットスライス構成で設計することにより1台あたりの使用個数を増加させ、ＭＳＩで製造するよりむしろ低コスト化に成功した。第2の問題点では、パイプラインを構成する各ユニット間の独立性が高いというデータ駆動アーキテクチャの特質が設計上有利であり、この点ではＬＳＩ化に適していることが判明した。

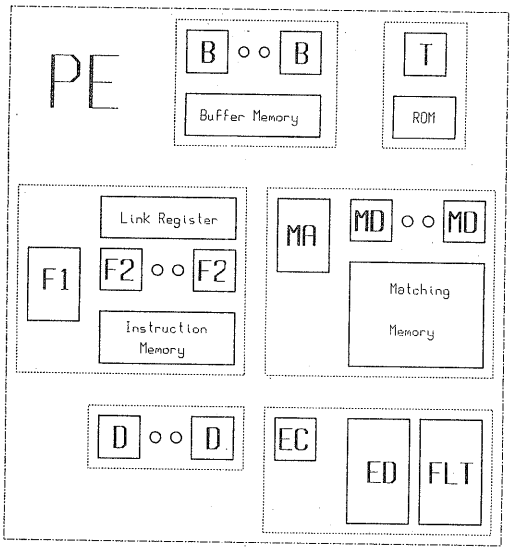


図７．ＬＳＩによるＰＥ構成

表1　ＳＩＧＭＡ－１のハードウェア構成

項目		予備試作版	ＬＳＩ化版
構成素子		Advanced STTL CMOS SRAM	CMOSゲートアレイ CMOS SRAM,DRAM
	ユニット名	N L	N L M G
素子数	Bユニット	178 (166)	23 (11) 6 約14300
	Fユニット	237 (186)	80 (29) 6 約15400
	Mユニット	385 (282)	144 (36) 7 約21500
	Eユニット	673 (673)	45 (45) 3 約12200*
	Dユニット	97 (97)	39 (39) 6 約14300
	Tユニット	0 (0)	11 (11) 1 約2300
	ＰＥ小計	1570 (1404)	342 (171) 29 約78000
	ＳＢユニット	78 (30)	23 (11) 6 約14300
	Sユニット	242 (176)	319 (220) 1 約8000
	Tユニット	0 (0)	11 (11) 1 約2300
	ＳＥ小計	320 (206)	353 (242) 8 約24600
	合計	1890 (1610)	695 (413) 37 約102600*
ＰＣＢ		8	2
			N M G
ローカルネットワーク			32 16 約128000
グローバルネットワーク			512 256 約2048000

但しNは素子数、Lはメモリを除く素子数、MはＬＳＩ数、Gは概算使用ゲート数
＊　浮動小数点回路（約1万ゲート）を除く。

ＰＥはドライバ等一部のＳＳＩを除き全てゲートアレイＬＳＩとメモリで構成される。ゲートアレイは９種類２８個をＰＥに使用する。図７にＰＥで使用するＬＳＩの分割を示す。これらのＬＳＩとメモリを１枚のプリント基板に実装する。総ゲート数は浮動小数点回路を除き約８７０００ゲートであり、メモリは約１Ｍバイトである。従ってゲートアレイによりＬＳＩ化することによりＩＣ個数、プリント基板上の実装面積共ほぼ５～６分の１になっている。表１に予備試作版及び多数結合版の諸元を示す。

４．デバグおよびメンテナンス

４－１．メンテナンス・アーキテクチャの必要性

計算機システム使用者にとっては、プログラム処理時間は計算機に関係する時間の一部にすぎず、プログラム処理時間以外のシステム速度を向上が真に高速な計算機システム構築に不可欠である。プログラム処理以外のシステム時間とは、システムの初期化、プログラム終了時の処理、プログラムのデバグ、およびハードウェアの故障診断等である。更に、ＳＩＧＭＡ－１のようにＬＳＩ化した並列プロセッサでは特に論理素子を直接検査することが不可能なため、メンテナンスおよびデバグ処理をアーキテクチャ内に取り込むことが重要な課題である。

並列計算機システムにおけるメンテナンスおよびデバグ処理は、現在までのところ、要素プロセッサ自身で相互に実行【１４】するか、または単一の通常型ホスト・プロセッサに接続して実現してきた。要素プロセッサ自身で相互に実行する方式は、並列処理計算機の並列性を生かした方式であるが、並列計算機で発生する資源の分配に関するオーバーフローまたはデッドロックが発生している場面のメンテナンスおよびデバグ処理に困難がある。一方、所謂サービスプロセッサにあたる動作を単一ホスト計算機が担当する場合には、相互方式にくらべ、簡単にかつ広い範囲のメンテナンスおよびデバグを実現できるが、Ｎの増加に比例して処理時間が増大するため、要素プロセッサ数の多い並列計算機システムでは現実的なものとは言えない。

したがって、速度と機能の両面を満たしつつメンテナンスおよびデバグ動作を実現するためには、専用の並列アーキテクチャが必要である。ここではこれをメンテナンス・アーキテクチャと呼ぶ。一方、プログラム実行を担当する表側の構成をプログラム・アーキテクチャと呼び、区別する。すなわち、最も基本的な演算器、メモリおよびレジスタを中心線として、片方は応用プログラム実行に好適に編成したプログラム・アーキテクチャを形成し、反対側では、メンテナンス及びデバグに好適に編成したメンテナンス・アーキテクチャを形成する。

プログラム・アーキテクチャがデータ駆動方式である計算機においては、メンテナンス・アーキテクチャの重要性は、プログラム・アーキテクチャが通常型の計算機である場合よりも増す。すなわち、データ駆動計算だけでは検出できない種類のメンテナンスまたはデバグ情報を得るためには、データ駆動方式ではないメンテナンス・アーキテクチャを負荷することが不可欠である。また、履歴依存性が本質的なメンテナンスおよびデバグではデータ駆動方式をメンテナンス・アーキテクチャとして使用する事が困難である。具体的には、あるデータフローグラフ中のノードにデータが到着したことは、データ依存関係から検出可能であるが、データが到着しないこと、または誤って他のノードに到着したことは検出不可能である。更に、メンテナンスにおける故障診断では、各構成要素間の接続の検査が最も基本的なものであるが、これはデータ駆動方式では検査不可能である。したがって、メンテナンス・アーキテクチャとしては通常型もしくはコントロール駆動型のアーキテクチャが要求される。言い替えると、データ駆動方式は、データ依存関係だけで決定される半順序に従って演算が実行されるため、正常動作では合理的にプログラム実行を行うが、異常動作が発生した場合には、その状況をデータ依存関係の半順序だけでは解析不能であり、全順序関係が必要となる場面がある。このことから、データ駆動アーキテクチャはそれだけで自立した計算機とはなりえず、通常型のメンテナンス・アーキテクチャと合わせ、表裏一体で１個の計算機システムとなる。従ってメンテナンス・アーキテクチャの設計が大きくデータ駆動計算機の総合性能を左右すると推察される。

単一プロセッサにおいては、プロセッサ内部状態は命令実行順序だけに依存し、実時間は意味を持たなかった。しかしながら、並列計算機システム、特にデータ駆動方式の並列計算機では、時間的要素は、陽な形でプログラム進行に影響をあたえる。従って、プログラム・アーキテクチャ上で動作するデバグ・ソフトウェアはプログラムデータ依存関係を乱さない範囲で実行順序の変更を行う可能性があり、使用範囲が限定される。

４－２．ＳＩＧＭＡ－１のメンテナンス・アーキテクチャ

ＳＩＧＭＡ－１におけるメンテナンス・アーキテクチャの課題は、高速なプログラム・アーキテクチャに見合う速度でメンテナンス、デバグ作業を可能とすることである。このためには、プログラム・アーキテクチャと同程度以上の並列性がメンテナンス・アーキテクチャに必要である。一方、上記メンテナンスおよびデバグ作業は、その本質上

システム全体に渡る制御およびデータの交換が必要であり、統合化された並列性が必要である。このような要請に答えるためには、メンテナンス・アーキテクチャを並列アーキテクチャとして実現する事が必須である。しかしながらメンテナンス・アーキテクチャには事の本質上簡易な構成が要請される。この目的のためにＳＩＧＭＡ－１のメンテナンス・アーキテクチャとして、最下層には微細な単位でのメンテナンス動作を同時並行に実行するＳＩＭＤ保守専用プロセッサを配することにより高速化を図り、上層にマイクロプロセッサを用いるＭＩＭＤアーキテクチャを用いるＳＩＭＤ／ＭＩＭＤ複合階層構成を採用した。

メンテナンス・アーキテクチャ最下層は、ネットワーク、演算プロセッサ（ＰＥ）および構造プロセッサ（ＳＥ）内のゲートアレイおよびＳＥのメンテナンス回路である。ここではビット単位の内部レジスタおよびメモリ制御線を取扱う。第２層は、ローカルネットワーク・ボード、グローバルネットワーク・ボード、ＰＥボードおよびＳＥボードに内蔵されるメンテナンス回路であり、保守ユニット（Ｔユニット）が中心となる階層である。第３層は、ＰＥ、ＳＥ、およびローカルネットワークで構成されるグループまたはグローバルネットワークに対応して実装されるメンテナンス・プロセッサであり、第４層即ち最上層が、サービス・プロセッサである。これらの階層のうち、第１層および第２層はＳＩＭＤ的に動作し、第３層及び第４層はＭＩＭＤ的に動作する（図８）。

各階層で扱うデータは、上部階層になると順次抽象性を増し、情報量は圧縮しかつメンテナンスを行う人間に理解し易い形式へと変換されて行く。

第１層では、ハードウェアの持つレジスタ、重要信号線、メモリコントロール信号線の値を、メンテナンス専用バス（Ｍバス）を介して読み書きする。第一階層の保守対象には、全てＭバス上のアドレス（Ｍアドレス）が付けられ、読み出し１ビット、書き込み８ビット単位で操作が行われる。なお、これらの操作はシステム・クロック停止時にも、メンテナンス・クロックが印加されている場合には可能である。Ｍバスは更にエラーの通知も行う。具体的には、各ゲートアレイ・チップ内でエラー（例えば、パリティエラー、オーバーフロー等）が検出されるとアテンションとして通知する。

ビット単位の操作を複合して、論理的にまとまった単位の操作を実現することが、第２層の役割である。また、第２層では、第１層からのアテンションを統合して、有効な情報にまとめ、システムクロック操作と関連付ける操作も担当する。

第２層と第３層を結合するバスをＴバスと呼ぶ。第３層は１６ビットマイクロプロセッサを用いて実現する。従ってＴバスはマイクロプロセッサの内部バスを直接使用する。

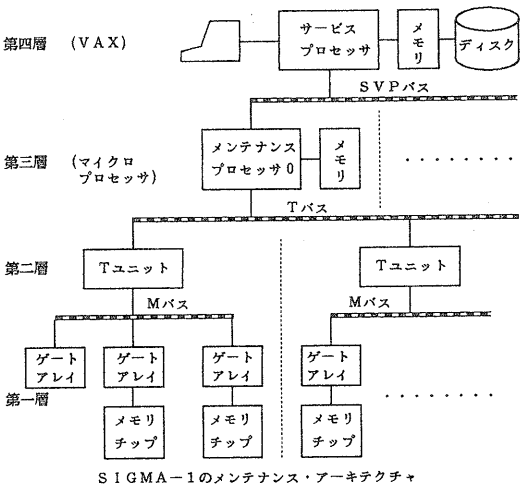
第３層は、第２層からのメンテナンス・データを統合し、高次のメンテナンス動作およびデバグ動作を実現する。即ち、第２層からメモリに送られてくるメンテナンス・データに対して、故障診断プログラム又はデバッガで処理を行い、必要なデータだけを選択して第４階層に送る。

デバグに関する機能は、本論文の範囲外であるが、ハードウェアが関係する機能として搭載を予定している項目は以下の通りである。

（１）ステッパーおよびマルチ・ステッパー。
（２）ブレークポイント作成機能。
（３）リンクレジスタ、マッチングメモリの使用率測定機能。
（４）ＰＥの稼働率測定機能。
（５）ＳＥの稼働率測定機能。
（６）ネットワークの衝突発生率測定機能。
（７）特定の命令にわなを仕掛ける機能。
（８）デッドロックの発生、プログラムの中断または終了を検出する機能。
（９）デバグ用のデータタイプをサポートする機能。

これらの機能は、全てプログラム・アーキテクチャだけでは実現不可能であり、メンテナンス・アーキテクチャの支援が必要である。

デバグ機能の高速な実現のためには、全ＰＥおよびＳＥで同時にデバグ機能が動作することが必要である。故障検出とデバグ機能の最も異なる点は、故障検出においては、



ＳＩＧＭＡ－１のメンテナンス・アーキテクチャ

図８．　ＳＩＧＭＡ－１メンテナンス・アーキテクチャ

全ＳＩＧＭＡ－１システムで同一のメンテナンス操作、即ちＳＩＭＤ的動作を行えば良いことに対し、デバグのためにはプログラムとの交信が必要となるため、各ＰＥまたはＳＥで異なるデバグ操作を実現する、即ちＭＩＭＤ的動作が必要となってくる。更に、故障診断においては、操作は専らメンテナンス・プロセッサおよびサービス・プロセッサ側が主体となって行うことに対し、デバグにおいてはプログラム側からの働きかけ、例えばトレース通知、ブレークポイント動作、各種稼働率測定機能等が中心的働きをすることである。

5．命令の設計

5－1．命令形式

フォンノイマン計算機の命令設計においては、命令に対するオペランドをどのように与えるかが、最も重要な選択項目の一つであった。しかしながら、データ駆動計算機では、命令に対する入力データは直接命令にたいして与えられる。一方命令からの出力データは次に実行される命令に直接送り込むため、次命令の指定方式がフォンノイマン計算機におけるオペランド形式と同等の作用をもつと推察される。

ＳＩＧＭＡ－１では出力パケットの行き先が命令である場合は、（ＰＥ，Ｉ，ＬＮ，Ｄ）で決定される。ただしＰＥは命令の存在するＰＥ番号、ＬＮはプロセスまたはサブルーチンを識別するリンク番号、Ｄはプロセス内部での命令相対アドレス、Ｉはループ中でのループカウントを示す。出力パケットには更にＦＬＧフィールドが付属し、命令発火におけるマッチング操作を指定する。

命令は出力パケットの（ＰＥ，Ｉ，ＬＮ，Ｄ，ＦＬＧ）を（１）入力オペランドで直接指定する、（２）命令のデスティネーション・フィールドで１命令に付き最大３個までグループ内の命令を指定する、（３）リンクレジスタ中の戻り番地を使用する基本形式に加え、パケットを発する命令のＰＥ番号、Ｉ、ＬＮの値でグループ内で分散させてＰＥ負荷の均一化を図っている。これらの所謂スタティックな出力指定の他に、ダイナミックな出力指定が、特に構造データの表現に必要である。ＳＩＧＭＡ－１では入力データによるＩおよびＤの指定機能によりダイナミック出力を実現している。

5－2．命令設計の概要

データ駆動計算機で命令レベルは、データ駆動原理によって命令を発火する単位に対応する。従ってそのアーキテクチャ上で実行が想定されている応用分野で要求されるデータ駆動の単位に命令を設定することが望ましい。すなわち、フォンノイマン計算機においては、個々の命令を単純化し、組み合せて使用することによって高性能のアーキテクチャを設計することが可能であるが、データ駆動計算機では、命令の発火操作は命令の直列要素として実行時間の一部を必ず占有するため、命令のレベルを低くすることは、プログラムが直列な場合等で性能の低下をもたらす。

ＳＩＧＭＡ－１が想定している応用分野は、科学技術計算、特にモンテカルロ法に基づく粒子系のシミュレーション等従来のベクトル型スーパーコンピュータで得意としない分野である。しかし、データ駆動計算方式の一般的有効性を示すためには、従来のベクトル型スーパーコンピュータで能率的に実行できない応用プログラムだけでなく、能率的に実行してきた応用プログラム、例えば行列やベクトルの基本演算、線形方程式の求解なども有効に実行し得る事を示す必要がある。

従来からのベクトル計算機の高速性は、パイプライン化した演算器、ベクトルレジスタ、主記憶装置の連係動作により実現されてきた。ベクトル計算機は特に演算器を効率よく稼働させることを目的として、各構成要素のバランスをとっている。従ってＳＩＧＭＡ－１をベクトル計算機で得意としている応用分野で効率よく稼働させるためには、ベクトルまたはマトリクスに対応する高レベルにデータ駆動原理と単位を設定することが有効である。一方、従来からのベクトル計算機で有効に高速化できなかった応用プログラムでは、サブルーチンコール・リターンが多発する場合、スカラー変数または主記憶上で離散している変数等多く用いられる場合、個々の命令実行順序が予測されない場合、並列性の再生産周期が短い場合等が発生していると考えられる。これらのケースを高速に実行することこそＳＩＧＭＡ－１の主目的である。この目的では、サブルーチン引き数内の並列性、細かい並列性を引き出す所謂命令レベルがデータ駆動レベルとして好適である。ＳＩＧＭＡ－１ではこれらの考察をもとに変数一個一個が命令発火の単位である命令レベルデータ駆動アーキテクチャを採用した。但しベクトルまたはマトリクスレベルにデータ駆動レベルを上げることは、ＰＥの一部にベクトル演算器を備えることにより容易に実現する。

プログラムを表現するデータフローグラフ中にあるノードの再利用、並びにデータ依存関係では表現されない順序関係の規定はＳＩＧＭＡ－１命令設計時のもう一つのガイドラインである。自立するデータ駆動計算機システム構築のためには、全てのプログラム要素を単一代入規則だけに基づいて記述することは不可能である。

以上まとめると、ＳＩＧＭＡ－１命令設計時の前提として以下に列挙する項目を選択した。

（１）ＳＩＧＭＡ－１処理装置はリンクレジスタ、命令メモリを除き永続的記憶要素を持たない。特にサブルーチン呼び出し、保守関係を除きＳＩＧＭＡ－１命令はレジスタのような永続的記憶要素なしに動作する。

（２）各処理装置を結ぶデータは全て単一の大きさのパケットを介して転送され、特殊なデータパスを持たない。

（３）ＮＯＰ以外に出力パケットを出さない命令を作らない。このことは安全なデータフロープログラムを作る上で重要である。

（４）命令実行が終了したことを確認するためだけに挿入される無駄な命令実行を抑えられるように、各命令の出力パケット順序を与える。

上記一般的項目に加えＳＩＧＭＡ－１固有の特徴を生かすために、

（５）不必要なコピー命令を極力減らすため、命令に許される最大３出力を活用すること。このことは、命令実行に依ってデータを消費するデータ駆動計算機特有の弱点を補うことになる。

（６）プログラム記述を２入力のマッチングなしでも行い得ること。すなわち、１入力命令と即値を持つ２入力命令だけで記述することを可能とすること。このことの副作用として不必要なマッチング操作が減少し、高速化にも貢献する。

（７）同一ＰＥ内であっても、パイプラインを止めないと実現しないような機能、例えば、マッチングユニットやフェッチユニット操作は、命令が直接行わず、一旦パケットを介して実現する。このことは、パイプライン・ハードウェア実現を非常に簡易にする。

このような前提の下に命令セットとして（１）算術演算命令、（２）ビット演算命令、（３）論理演算命令、（４）プレディケイト、（５）スイッチ命令、（６）分岐命令、（７）分岐スイッチ命令、（８）Ｉフィールド命令、（９）ループ命令、（１０）サブルーチン命令、（１１）アーギュメント受け渡し命令、（１２）構造体メモリ取扱い命令、（１３）システム命令、（１４）保守命令および（１５）その他の命令を作成し、ＰＥで直接実行するパケットとして、（１６）ＳＹＳタイプパケット、（１７）ＳＰＥＣＩＡＬタイプパケット、（１８）ＭＡＩＮＴタイプパケットを、ＳＥで直接実行するパケットとして（１９）ＳＴＲＵＣＴタイプパケットを作成した。表２に代表的なＳＩＧＭＡ－１命令を示す。

５－３．ＳＩＧＭＡ－１命令

まずベクトル計算機が得意とする分野に対する命令セットを述べる。数値計算を主目的とする性格上浮動小数点演算は高速に実行されなければならない。しかしながら、並列処理の実験機としては浮動小数点演算の高速化は相反する２種の結果を導く。すなわち、浮動小数点演算の高速化によりＳＩＧＭＡ－１自身の演算速度は向上し、応用プログラムの短時間での実行が可能となる。その半面全実行時間に対する浮動小数点演算の相対比率が低下する。その原因は、基本的な整数命令、分岐命令等の時間は、使用する論理素子テクノロジーで決定され、余り変化する要因が無い為である。浮動小数点演算回路の使用効率、即ち最高速度に対する平均実行速度の低下の割合は、現在の所アーキテクチャ評価項目として重要な意義を持つ。その結果、全体的な能力向上策がかえってアーキテクチャとしての評価を低めることになって跳ね返って来る。

具体的に説明すると、ＳＩＧＭＡ－１各構成要素の能力を上げることにより、かえって浮動小数点演算の最大能力と実際のプログラムにおける平均浮動小数点演算速度との比率が低下する結果となる。

ＳＩＧＭＡ－１の設計に際しては、近い将来浮動小数点演算回路の計算機全体に対する比重が著しく低下するであろう事を予測して、浮動小数点演算回路の使用効率を二義的なものとし、全体としての性能を重視して命令およびハードウェアを設計した。具体的には、浮動小数点演算命令として四則演算および整数と浮動小数点数間の変換命令を持たせ、演算回路をよりよく活用する演算器内部でのパイプライン動作、ベクトル演算命令的動作を採用しなかった。なお、後に述べるように、パケット長の制約が原因となって浮動小数点数としてはＩＥＥＥ標準形式の単精度数だけを備えた。ある分野の応用プログラムでは、単精度数だけでは解の精度・安定性等のからみで不十分であるが、ＳＩＧＭＡ－１が持つ実験機としての性格からやむをえぬ選択といえる。

先に述べたように、浮動小数点演算命令を有効に活用するためには、ループの制御方式は重要な意味を持つ。ＳＩＧＭＡ－１ではデータを運ぶ各パケットにループカウントを持たせて１次元ベクトルを同時に解く（ＵＮＦＯＬＤ）方式をハードウェアとして採用している。従ってループカウントを扱う命令の実行速度を構造体メモリアクセスおよび浮動小数点演算と最適化する必要がある。この要求に基づき、ループをほどくための複合的命令を作成した。

その他の一般命令は通常のフォンノイマン型計算機とほぼ同じレパートリーを持つ。但し入力オペランドの与えかた、結果をどこかに格納する代わりに出力パケットとして送り出す点が異なっている。

通常命令の中では分岐命令および同期命令がデータ駆動計算機特有な命令といえる。フォンノイマン計算機において条件分岐命令は、命令の実行順序を条件に従って変更することに対し、データ駆動計算機ではデータの流れ自体を直接条件に従って変更する。この結果、条件分岐命令の出力に連なっている命令部分に流入するデータ全てを同じ条件でスイッチまたはゲートしない限り、プログラム実行終了時にパケットがごみとして計算機中に残ってしまう。

同期命令（ＳＹＮＣ）はこのような状況において、データのゲートを作成する時などに使用する命令である。即ち、２個の入力双方にデータが到着した時点で、入力の左、右または双方を出力する。

上記の例に現れているように、プログラム実行終了時に計算機中にごみパケットを残さないことは、データ駆動計算機を実用機とするために必須の条件である。このため、唯一つの例外（ＮＯＰ命令）を除いて、命令が出力パケットを出さないことを禁止している。これは命令実行が終了したこと、即ちマッチング・ユニット中のパケットが消費されたことは出力パケットが出てくる事実だけから判断できるからである。

ＳＹＳタイプパケットはＰＥに対しサブルーチンまたはシステムサブルーチンを結合するために用いられる。ＰＥはＳＹＳタイプパケットを受理すると、未使用のＬＮを探索し、新たなサブルーチン環境を作成してその動作を開始させる。

ＳＰＥＣＩＡＬタイプパケットは、ＰＥ内部のマッチングユニットに対する特殊動作を規定する。具体的には、マッチングユニット中のデータに対し（１）消去（＃ＡＢＳＯＲＢ）、（２）取りだし（＃ＳＡＬＶ）、（３）代実行（＃ＥＸＥＣＵＴＥ）、および（４）再実行（＃ＲＥＳＵＭＥ１，＃ＲＥＳＵＭＥ２）を行う。何れの場合もＫＥＹ部ＩＴＡＧフィールド上位３ビットでＳＰＥＣＩＡＬタイプパケットであることを、下位５ビットでパケット種を示し、ＫＥＹ部３２ビットで操作を行う対象番地を（Ｉ，ＬＮ，Ｄ，ＦＬＧ）で示す。データ部の解釈はパケット種で異なるが、操作の結果を通知するＰＥ番号及び（Ｉ，ＬＮ，Ｄ，ＦＬＧ）を保持するか（＃ＡＢＳＯＲＢ，＃ＳＡＬＶの場合）、代実行の値を保持する。

ＭＡＩＮＴタイプパケットはＳＩＧＭＡ－１ハードウェア内のメモリ及びレジスタ等を初期化または読み書きするものである。ＫＥＹ部３２ビットで操作の対象となるロケーションを、データ部で操作結果の送り先（@ＬＤｘｘｘの場合）または書き込みデータ（@ＳＴＲｘｘｘの場合）を保持する。なお、ＳＥ内のメモリまたはレジスタはＭＡＩＮＴタイプでなくＳＴＲＵＣＴタイプパケットで読み書き、初期化される。

ＳＴＲＵＣＴタイプパケットはＳＥに対するアクセスおよび管理を行うパケットである。ＳＴＲＵＣＴタイプパケットは構造体メモリの初期化、ベクトルの割り付けおよび解放、読み書きアクセス、および待ち合わせキュー管理などの操作を行う。パケットＫＥＹ部ＩＴＡＧフィールド最上位ビットでＳＴＲＵＣＴタイプであることを示し、下位７ビットで操作の種類を示す。ＫＥＹ部３２ビットは大部分のパケットでは対象となる構造体メモリ番地を示し、データ部タグフィールド及びデータフィールドで、書き込みデータまたは読みだしデータの送り先等を示す。

６．おわりに

大規模データ駆動計算機構成上の問題点およびその対応策について様々な角度から述べた。並列計算機実用化のために必要な基本的問題点、すなわち同期、負荷分散、ネットワーク、直列部分の高速化はＳＩＧＭＡ－１では原則的には解決した。しかし、データ駆動アーキテクチャ固有の問題点である、並列度の制御、構造体の扱いについては、解決すべき課題が多い。データ駆動アーキテクチャを実用化するためには、これ以外にも問題点が出現する可能性があり、ＳＩＧＭＡ－１の目的の一つともなっている。

ＳＩＧＭＡ－１は現在構成要素であるＬＳＩの設計、製作をほぼ終了しＰＥ、ＳＥおよびネットワークの製造を行っている。

本研究を遂行するに当たり、柏木電子計算機部長、弓場計算機方式研究室長並びに研究室の同僚諸氏に感謝します。なお、本研究は大型プロジェクト「科学技術用高速計算システム」の一環として遂行した。

参考文献

【１】 Dennis,J.B. and Misunas,D.P,"A preliminary architecture for a basic dataflow processor,"Proc. 2nd Ann. Int. Symp. Comput. Architecture,pp. 126-132,1975.

【２】 Arvind,Gostelow,K.P. and Pingali,K.,"A dataflow architecture with tagged tokens,"Tech. Rep. TR-174, Lab. Computer Science,MIT,1980.

【３】 Gurd,J.,Kirkham,C.C. and Watson,I.,"The Manchester prototype dataflow computer,"Commun. ACM, Vol21,No.1,pp. 34-52,1985.

【４】 Hiraki,K.,Nishida,K.,Shimada,T.,"A Hardware Design of a Data flow Computer for Scientific Computations," Proc. of Int. Conf. on Parallel Processing, 1984.

【５】 島田、関口、平木、西田: 科学技術計算用データ駆動計算機ＳＩＧＭＡ－１の予備試作、情処学会第３０回全国大会、４Ｃ－８。

【６】 島田、関口、平木、西田: 科学技術計算用データ駆動計算機ＳＩＧＭＡ－１の予備試作版基本プロセッサの評価、当研究会予稿、１９８５。

【７】 平木、関口、島田: 科学技術用データ駆動計算機ＳＩＧＭＡ－１における負荷分散機構、信学技報ＥＣ８５－６。

【８】 平木、西田、島田: 科学技術用データ駆動計算機ＳＩＧＭＡ－１の構造体記憶の試作、情処学会第３１回全国大会、６Ｄ－２。

【９】D.D.Chamberlin:The "single-assignment" approach to parallel processing, FJCC,pp.263-269, 1971.

【１０】 Arvind,K.P.Gostelow and W.Plouffe:Indeterminacy, Monitors, and Dataflow, Proc. of the 6th Symp. on Operating Systems Principles, pp.159-169, 1977.

【１１】N.D.Francesco, et al.:On the Feasibility of Nondeterministic and Interprocess Construction in Data Flow Computing Systems, Proc. of the 1st European Conf. on Parallel and Distributed Computing,pp.93-100, 1979.

【１２】A.J.Catto and J.R.Gurd:Nondeterministic Dataflow Graphs,Information Processing 80, pp.251-256,1980.

【１３】小野、高橋、長谷川、雨宮: データフローマシンにおける資源管理方式, 信学技報ＥＣ８２－７６。

【１４】 Hosseini,S.H.,Kuhl,J.G. and Reddy,S.M.,"A diagnosis algorithm for distributed computing systems with dynamic failure and repair,"IEEE Trans. Comput.Vol.C-33,No.3,pp. 223-233,1984.

表２　ＳＩＧＭＡ－１命令

[Arithmetic] ADD SUB MUL DIV MOD FADD FSUB FMUL FDIV FIX FLOAT DIST CLR INC DEC SHIFTL SHIFTR SLA SRA

[Bit] AND OR XOR NOT

[Boolean] LAND LOR LNOT TRUE FALSE

[Predicates] GE GT LE LT FGE FGT FLE FLT EQ NEQ TEQ EQINT NEQINT EQFLT NEQFLT

[Branch] SGE SGT SFGE SFGT SLT SLE SEQ STEQ SEQINT SEQFLT

[Switch and Branch] PSGE PSGT PSFGE PSFGT PSLT PSLE PSFLT PSFLE PSEQ PSTEQ PSEQINT PSEQFLT

[Switch] SWITCH SYNC_L SYNC_R SYNC_LR

[Iteration] I_INC I_CLR I_DEC SI_INC SI_DEC

[Loop] INC_GE INC_GT INC_LE INC_LT DEC_GE DEC_GT DEC_LE DEC_LT LOOP UNFOLD

[Sub-frame] CALL SCALL LCALL LINK SLINK GLINK UNLINK GUNLINK GETGRP KILL SKILL SYSCALL HOSTCALL TRAP

[Argument] SENDARG RETARG RETKILL

[Structure Handling] MK_GVST MK_GHST MK_LVST MK_LHST MK_SST INDEX FETCH AFETCH STORE ASTORE ID_FETCH ID_AFETCH DEL_STRUCT FETCH_WQ DEL_WQ TST_FLG

[System] SALV EXECUTE ABSORB RESUME1 RESUME2

[Miscellaneous] NOP TRIM_DATA TRIM_TAG TRIM_KEY CHGTAG MKFID MKPACKET

[Maintenance] LDLRT LDLRP LDIM STRLRT STRLRP STRIM LDIMM STRIMM LDPLC

[SYS packets] *CALL *SCALL *TRAP *SYSCALL

[STRUCT packets] %READ %WRITE %AREAD %AWRITE %MWRITE %MK_GVST %MK_GHST %MK_LVST %MK_LHST %MK_SST %FETCH_WQ %DEL_WQ %DEL_GVST %DEL_GHST %DEL_LVST %DEL_LHST %DEL_SST %SET_FLG %TST_FLG

[Special packets] #SALV #EXECUTE #ABSORB #RESUME1 #RESUME2

[Maintenance packet] @LINK @UNLINK @KILL @LDLRT @LDLRP @LDIM @STRLRT @STRLRP @STRIM