SAMD計算機 ～ A High Level Data Flow Machine ～

元岡 達，鈴木 達郎，喜連川 優，新岡 香織
（東大 工学部）

＜要旨＞

現在までに data flow machine など新しいアーキテクチャーがいくつか発表されている．その特徴として大きな並列度を引き出せることはもちろんだが，ここではその data driven による制御の局所性に注目し，対象とする大規模なマルチマイクロプロセッサの制御を，多くのアルゴリズムごとに分割されたタスクレベルの data flow によって分散的に行なうことを目的とする．さらに，問題点の1つである負荷の分散は，各プロセッサが自由競争を行なうことで解決した．

ハードウェアに関しては，Z-80 やインテルの 8080 ファミリー素子を中心として，プロセッシングモジュールとメモリモジュールの2種類による完全なモジュール構造を持つ TOPSTAR システムを設計作製した．

結合方式は，DMA による高速転送，割込みによるメモリモジュールへのアクセス競合の解決などを行なう．

各プロセッシングモジュールは，メモリモジュールを共有することで結合され，全体を部分結合として，大規模システムでの実現性を考慮している．

現在メモリモジュール2台，プロセッシングモジュール3台から成るシステムが完成し動作しているが，これに並列マージソートなどの応用プログラムを乗せて，システムの評価を行なっている．

§1. 序論

§1－1 はじめに

LSI の急激な進歩によるマイクロプロセッサなどの低価格化，高性能化は，計算機の構成法に大きな影響を与えつつあり，その1つに多数のマイクロプロセッサを用いて高性能化を計ったマルチマイクロプロセッサシステムがある．

8080 プロセッサを多数用いて，気象計算を行なう SMS 201[1] や，LSI－11 を用いた Cm*[2] などいくつか提案されているが，適用範囲の狭さや，並列度が大きい場合の結合方式，制御方式にまだ問題点がある．

ここでは，タスクレベルに data driven の概念を用いて，並列度が十分大きく，しかも柔軟な構造を持ちうるシステム構成を提案する．

data flow machine としては，J. B. Dennis and R. P. Misnus[3] や J. Rumbaugh[4] などのシステムがすでに提案されている．これらは instruction レベルで，計算機の内部に data flow の概念を持つものであるが，ここでは通常のマイクロプロセッサを要素として用いたマルチプロセッサシステムでの制御のやりとりを，タスクレベルの data flow control で行なうことを目指したもので，P. J. Denning の分類[5]では，中規模の network における data flow の適用に相当するものと考えられる．

§1－2 方針

(1) パターン認識や人工知能などのように，大きな並列性を持ち，リアルタイムに連続的にデータが入ってくるものを効果的に処理する．

(2) プロセッサなどを冗長に用いることで，分散処理を行ない，OS を簡素にする．

(3) 100台以上の高並列度で処理することを考慮して，ハードウェア，ソ

フトウェアの設計を行なう．

(4) data flow の考えを中心として，プログラム構造の作成法を考える．

以上のことをもとに、ＳＡＭＤ処理方式及びその実装に適したアーキテクチャーを持つＴＯＰＳＴＡＲシステムを以下に提案する．

§２．ＳＡＭＤ処理方式

§２－１ ＳＡＭＤ処理とは．

ＳＡＭＤ（Single Algorithm-stream Multiple Data-stream）とは，以下の条件をみたす並列処理方式である．

(1) 一連のアルゴリズムから構成される．

(2) 各アルゴリズムは複数の data stream を受けることで，複数のタスクを発生する．（このタスクの発生は data driven に行なわれ．ダイナミックに数が変化できる．）

(3) 多くのデータが実時間的に連続に入力するとき，アルゴリズムを単位としたパイプライン を形成する．

このＳＡＭＤ処理におけるデータ，アルゴリズム，プロセッサの関係を示したのが，図２－１である．

data stream の図は一種の data flow chart で，各ノードは data 駆動されたタスクに対応する．

また，algorithm stream につながっているプロセッサは（アルゴリズム，データ）の組に対応したタスクごとに各アルゴリズムに雇われて，その処理を実行する．この雇用関係は data flow に応じて自由に変化する．

§２－２ 制御方式

マルチマイクロプロセッサの上で，アルゴリズム，データ及びプロセッサの３つの資源を効率的に制御するための規則をまとめると次のようになる．

(1) data driven schedule

各アルゴリズムに入力される data flow に応じて，タスクを data driven で発生させる．

(2) 自由競争

各プロセッサは各アルゴリズムに対して，自律的にタスクを要求し．タスクの存在するアルゴリズムに対して雇用関係を結ぶ．

(3) dynamic structure

タスクを実行するプロセッサは．

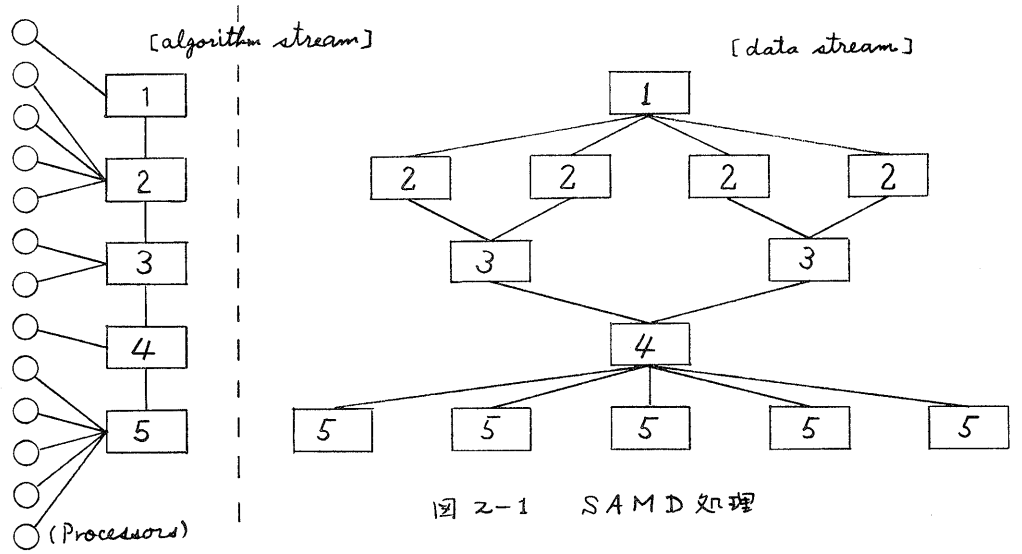


図 ２－１ ＳＡＭＤ処理

アルゴリズムに従って，入力データの内容に応じた処理を行ない，その結果に対して，出力データをそれぞれ適当な次のアルゴリズムの所へ送出する．

(4) interaction の禁止

タスク同志の interaction は行なわない．従って，タスク間の同期はアルゴリズムに対する data 駆動によってのみ行なわれる．

§2-3 SAMD処理の特徴

SAMD処理の特徴を以下にまとめる．

(1) 高並列度

SAMD処理は data flow chart の node の数だけの並列度を持つことが可能である．

即ち，各アルゴリズムの持つ並列度は従来のSIMD処理と同様で，さらにアルゴリズムがそれぞれの並列度を持って連結することで，パイプラインを形成するため，全体の並列度はアルゴリズムの数と，各アルゴリズムの持つ並列度の積になる．

(2) 可変構造パイプライン

一般にパイプラインはいろいろなレベルで行なわれるが，通常は要素となった処理装置の組み合せであり，パイプラインの構造も固定である．

しかしSAMD方式ではパイプラインの構成要素としてマイクロプロセッサを考えており，それらは対等に扱える．このプロセッサが自由競争を行なって，仕事が多く長い所に自然に集まることにより，つねに最適の配置になるように構造がダイナミックに変化する．従って利用性を持つことができる上に，データの質や量などのばらつきに対して柔軟に対処できる．

(3) 分散制御と flow control

マルチプロセッサシステムの制御法として，マスタースレーブ方式などの集中制御と，平等方式の分散制御が考えられるが，システムが高並列度になってくると集中方式では制御がネックになってしまう．一方分散制御を行なうためには制御の局所性が必要である．即ち局所的な情報のみを使って制御して，全体として矛盾なく処理が進まねばならない．この点，data driven 制御は，各々自分の所へ到着する情報のみを用いてタスクを起動できるので，分散制御に適している．

またシステム全体として，データの流れがある程度一様でないと，各ノードでのデータ容量を超えて，over flow が生じる恐れがある．これも分散制御の持つ問題点の1つであるが，これはデッドロックを招く原因ともなりうるので，semaphore などで管理する必要がある．

§3 SAMD計算機TOPSTARの設計と製作

§3-1 TOPSTARの方針

これまでに述べたSAMD処理を効率的に実現できるハードウェアシステムを作製したので報告する．

TOPSTARを作製する上での方針は次のようなものである．

(1) SAMD処理に適したマルチマイクロプロセッサ

algorithm stream を担うメモリモジュールと，data driven されたタスクを自由競争で処理するプロセッシングモジュールにより構成する．

(2) 高並列度の実現 ～部分結合～

すべてのメモリモジュールとプロセッシングモジュールが直接結合している完全結合システムは理論的には望ましいが，並列度が大きくな

って くると結合部のハードウェアが膨大となり非現実的になる．またバス結合のような時間分割的な方法はメモリモジュールへのアクセス競合がネックとなりやすい．そこで，algorithm streamにほぼ沿って dataが流れるとの仮定の下で，各メモリモジュールの近傍の（具体的には8or16程度）プロセッシングモジュールとのみ直接に結合する部分結合方式を採用する．

(3) モジュール構成

システム全体はメモリモジュールとプロセッシングモジュールの2種類のみで構成し，各モジュールはすべて同一構造とすることで製作を容易にし，さらに故障診断，交換に適するようにする．またモジュール内部も基板に対応する程度のサブモジュールにより構成する．

(4) LSIの利用

マイクロプロセッサメーカーは各種の機能を持つLSIをファミリーとして作製しているが，それらを最大限利用してモジュールの構造を単純にする．

(5) DMA結合

SAMD処理での通信は，データなどのブロック転送が中心なので，それに適した高速DMAによる結合を考える．

以上の方針のもとに作製したTOPSTARの構成は，図3－1のブロック図のようになっている．

2種のモジュールはそれぞれプロセッサ，メモリおよび結合部から成り，結合部を除いてはほぼ同一の構造を持っている．

メモリモジュールの持つメモリが，

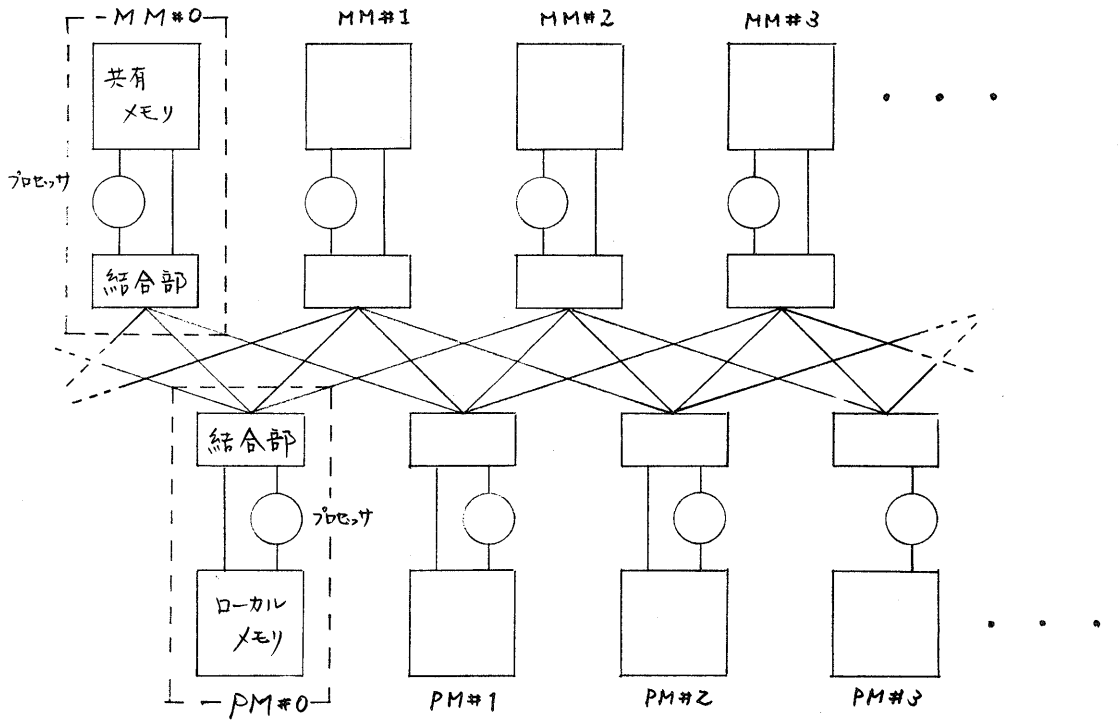


図3－1　TOPSTARのブロック図

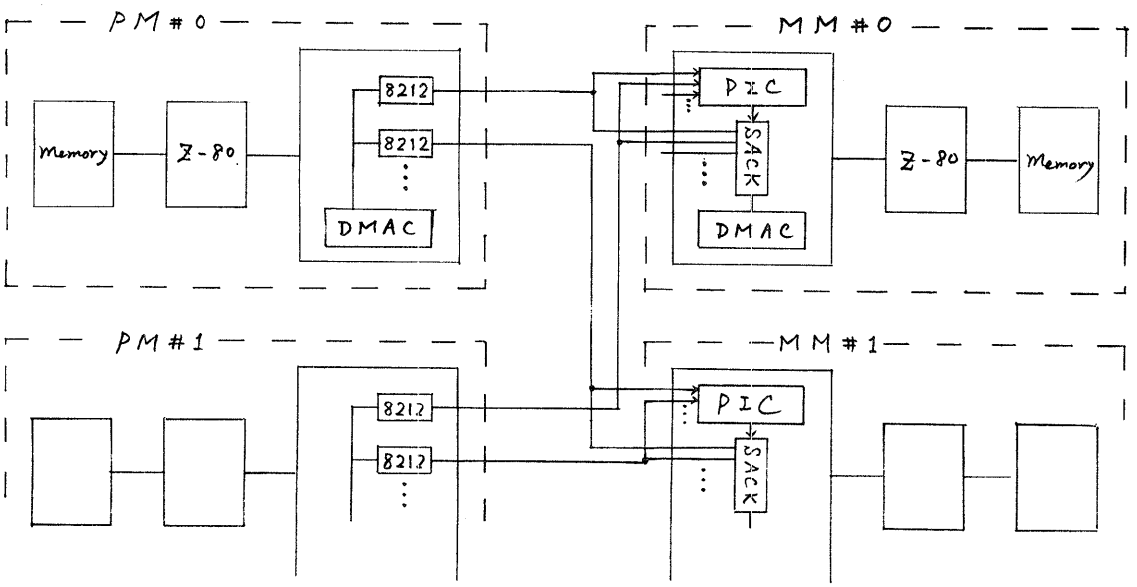


図 3-2　TOPSTARの結合方式

共有メモリで，ここにメモリモジュールを管理するシステムプログラム及びSAMD処理におけるアルゴリズムとそれに対応するデータが格納される．

プロセッシングモジュール内のメモリはローカルメモリで，プロセッシングモジュールの自由競争などを行なう固定のシステムプログラムと，自由競争の結果与えられたタスクの（アルゴリズム，データ）の組をロードして処理を実行する領域とから成る．

§3-2　DMA結合方式

メモリモジュールとプロセッシングモジュールは，それぞれの結合部のDMAコントローラ（インテル8257）が直接結合することによりデータのやりとりを行なう．この結合方式を示すのが図3-2である．

結合の要求割込はプロセッシングモジュールからおし，メモリモジュール結合部内の割込コントローラ（インテル8259）により競合を解決して，2つのDMAコントローラの結合が完成する．またこのときDMAのためのクロックは同一のものでなければならないので，結合が完了した時点でメモリモジュールの結合部からプロセッシングモジュールの結合部に向けてクロックが送り出される．

§3-3　結合方式の評価

メモリモジュールとプロセッシングモジュールとの間のデータのやりとりは，コマンドの通信とデータの授受の2回のDMA通信により行なう．

nバイトの連続データをソースからデスティネーションまで送るのに要する時間をT(n)とすると，

$$T(n) = T_1 + T_2 + T_3 + T_4 + 4n$$

ただし

T_1（1次DMAのためのDMACのセット）
$= 100$ [clock]

T_2 (1次DMAでのコマンド送出)
= DMAの同期時間 + DMA時間
= 41 [clock]
T_3 (2次DMAのためのDMAC
のセット)
= 124 [clock]
T_4 (2次DMAの同期時間)
= 25 [clock]
$4n$ = データのDMA時間
従って
$T(n) = 290 + 4n$ [clock]
となる.
なお比較のため, TOPSTARにはプログラムモードでのハンドシェイクによるバイト単位の転送モードも設けているが, その値は
$T(n) = 144 + 45n$ [clock]
である.
(TOPSTARでのクロックの値は2.5 MHzである)

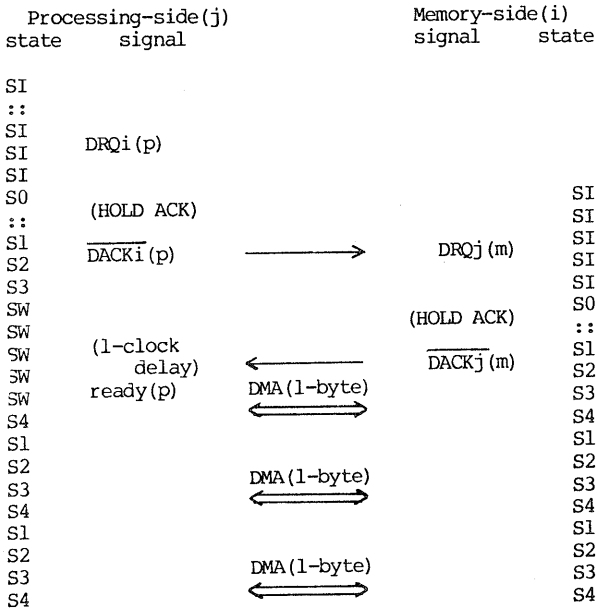


図 3-3 Synchronization of two DMACs

参考のため2つのDMACの同期関係を図3-3に示す.
図はプロセッシングモジュールが同期待ちを行なった場合について述べてあるが, ハードウェア的には対称に作ったので, 一般に早い方のDMACが同期待ちを行なうようになっている.
また, DMAの打ち切りは, どちらか一方のDMACが指令すればよく, 従ってDMAを行なう data の数はどちらか一方が知っていればよい.

§4. システムプログラム

§4-1 システムプログラムの機能

SAMD処理におけるシステムプログラムは, メモリモジュールとプロセッシングモジュールの各々に常駐されるが, 大きく分けて次のような機能を行なう.

(1) data driven 制御

あるアルゴリズムに対して, それが動作するのに必要なすべてのデータがそろったとき, その(アルゴリズム, データ)の組に対応するタスクが実行可能になる. 各メモリモジュールは自分の中に存在するアルゴリズムに対して送られてくる入力データを管理し, data driven 制御を行なう. この時次に述べる flow control の許す範囲で起動されたタスクを, アルゴリズムごとに設けられたスケジュールキューに積む.

(2) flow control

各アルゴリズムの存在するメモリモジュールにとって, 送られてくる入力データを蓄える領域は現実には有限の大きさで, これを超えてデータが送られてくると, over flow が生じてデータが失なわれてしまう.
これを解決する1つの方法は, 出力データを送出するプロセッシング

モジュールがチェックして メモリモジュールのデータ領域に空きができるまでその状態で wait することが考えられるが、ここでは、デッドロックを防ぐ意味と、すべてのタスク同期関係をメモリモジュールが管理するという立場から、処理中に wait しなければならない可能性がないことを確かめてから、タスクをプロセッシングモジュールに渡すようにする。このための制御が flow control である。flow control は、メモリモジュール内の data 領域に対する semaphore 管理で行なわれる。即ち、データ領域にデータを送り込む場合は P-operation, データ領域からデータを取り出す場合には V-operation を行なう。

(3) 自由競争制御

各プロセッシングモジュールは、1つの仕事が終わると次の仕事を求めてメモリモジュールに要求を出すこれは特定のアルゴリズムに固定的に従属するものではなくて、システム全体としてどこかに仕事がある限りそれを求めて回る。

ただし通信の量や回路を減らすために、通常はまず直前に雇われていたアルゴリズムの所へ仕事を求めることにする。この場合プロセッシングモジュールの持つアルゴリズムの入れかえが必要でない可能性が高く、システム全体として無駄な動きをしなくて済む。

またこの自由競争を適当に制限することで、処理に優先度を付けたり、特殊プロセッサをシステムに結合したりする場合の細かい制御も可能になる。

§4-2 コマンド

ユーザのプログラムはすべてプロセッシングモジュールの上で実行され、メモリモジュールはすべて受け身である。このときメモリモジュールは一種の知能メモリとして動作するわけだが、このメモリモジュールに対するアクセスは、コマンドの形でまとめることができる。

§4-1で述べた機能を実行するために、次のようなコマンドが、プロセッシングモジュールからメモリモジュールへ送られる。

(1) DEQ (引数: アルゴリズム番号, アルゴリズムの入替指定)

アルゴリズムごとに設けられたスケジュールキューから実行可能なタスクを取り出す動作の要求。

これに対してメモリモジュールはデータ及び必要に応じてアルゴリズムをプロセッシングモジュールに送出する。(もし仕事がない場合は、その旨知らせる。)

(2) ENQ (引数: アルゴリズム番号, データED)

プロセッシングモジュールは処理の結果に応じて、出力データを適当な次のアルゴリズムの所へ送出する。

これを受けたメモリモジュールは次の data driven 制御を行なう。

(3) V-OP (引数: アルゴリズム番号, データED)

メモリモジュールからデータを取り出した後、データ領域の空きができたことを、そこへデータを送る可能性のあるアルゴリズムに対して知らせる。これにより flow control が行なわれる。

(P-operation は DEQ が実行された時点で、メモリモジュールが自動的に知ることができるので、特にコマンドとして知らせる必要はない。)

(4) BYPASS (引数: 受け側アルゴリズム番号, 受け側メモリモジュール番号)

部分結合の制限から直接データを転送できない場合，このコマンドと共に，目的地に近づくメモリモジュールにデータを送出する．受けたメモリモジュールは適当な他のプロセッシングモジュールを雇って，そのデータの転送を仕事として与える．

これらの動作をまとめて1例を示すと，図4－1のようになる．

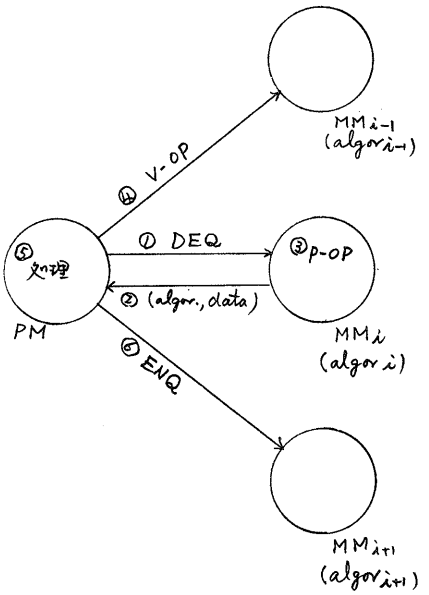


図4-1　PMとMMの動作例

図4－2にメモリモジュールの動作，図4－3にプロセッシングモジュールの動作のフローチャートを示す．

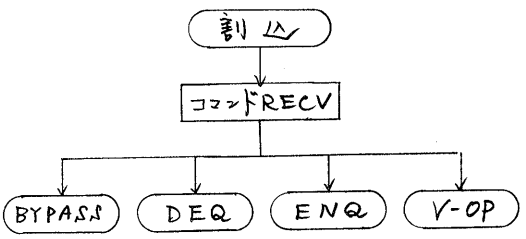
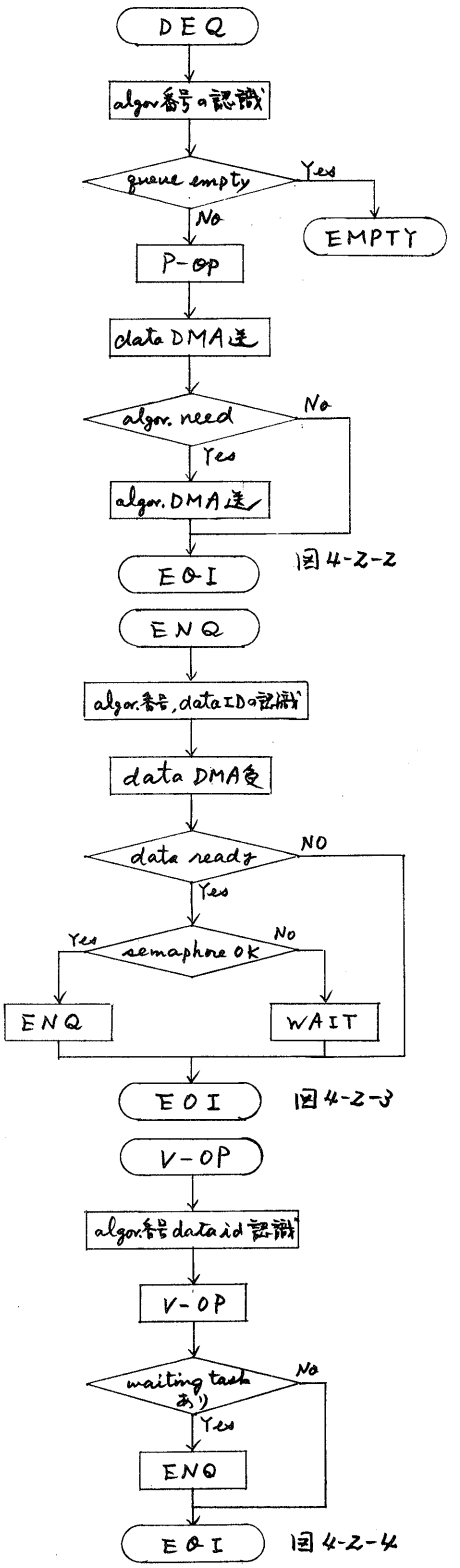


図 4-2-1



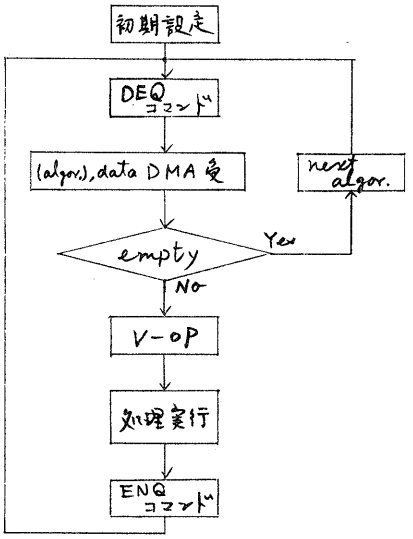


図 4-3 プロセッシングモジュールの動作

§5 並列マージソートプログラムの実装

SAMD計算機のデモンストレーションと評価を兼ねて，並列マージソートプログラムを実装する．

並列マージソートの data flow chart は図5-1のようになっている．

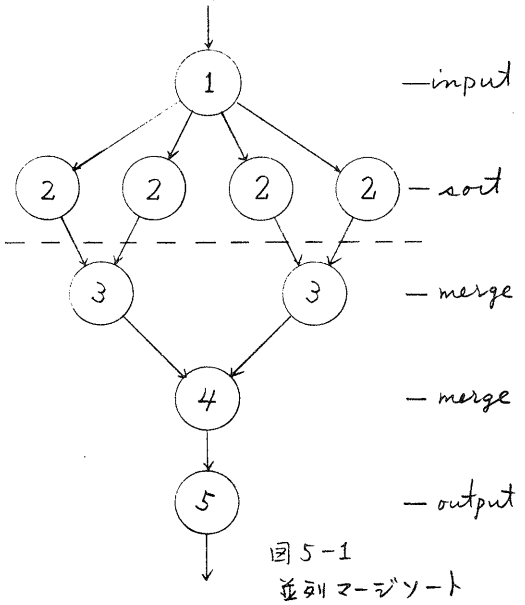


図5-1
並列マージソート

アルゴリズム1は入力プログラムで256ケの4バイトデータを64ケづつ4組に分けて，アルゴリズム2のソートプログラムを data 駆動する．ソートプログラムの出力は2組づつ join してアルゴリズム3のマージプログラムを駆動し，アルゴリズム4のマージプログラムで256ケのデータすべてがソートされる。アルゴリズム5は出力プログラムである．

現在完成している，メモリモジュール2台，プロセッシングモジュール3台のシステムでは，アルゴリズム1,2をメモリモジュール#0に，アルゴリズム3,4,5をメモリモジュール#1に乗せ，プロセッシングモジュール3台は完全結合で行なった．

§6. 今後の問題点

§6-1 システムシミュレーション

ハードウェア構成が固定すれば，プリント基板をおこして量産する予定である．印刷漢字認識システムをSAMD処理した場合の様子を以前にPASCALを用いてシミュレーションしているが(1)部分結合の影響や割込みによる競合解決などを考慮していなかったので，メモリモジュールとプロセッシングモジュールの台数や比率及び部分結合の数などの最適値を含めてもう一度シミュレーションを詳しくやって定める必要がある．

§6-2 高級言語の導入

並列処理に適した高級言語はこれまでも多くあり，P.B.Hansen のマルチマイクロプロセッサ向き言語(2)なども興味深いが，これはプロセスとプロセッサが固定的に対応しており，もっと柔軟で data flow の概念を導入した言語構成が必要ではないかと考えている．

また現在 LISP の data flow 処理の

研究も始めており，LISPシステム自体をdata flowで並列処理することと，LISPの言語内にdata flowの記述能力をとり入れることなどを考えている。

§6-3 SAMD処理の拡張

基本的にSAMD処理ではアルゴリズム間の起動関係はループフリーであるが，システムプログラムの大きな変更なしに，ループ処理を導入することは可能である．ただし部分結合の範囲を大きく越えるような場合は効率の点で問題が出るかも知れない．

また，ユーザプログラムをメモリモジュールで直接実行することにより，abstruct data typeの実現も可能である。この場合，ユーザの走めたコマンドにより，メモリモジュールに，アクセスすることになる．

§7. 結論

data flow contralをタスク制御のレベルで取り入れたSAMD処理方式により，大規模リマルチマイクロプロセッサでの有効なハードウェア，ソフトウェアの構成を示すことができた．

ハードウェアのモジュール化及び，LSIの積極的利用により，作製の容易さも確認された．

今後は，システムを大きくして，その上での動作経験を通じて，自由競争制御やdata flowに適した並列プログラムの構成，言語言語による記述などを研究して行きたい。

<参考文献>

(1) H. Kopp, "Numerical Weather Forecast with Multi-microcomputer System SMS 201," Parallel Computers Parallel Mathematics, pp. 265-268, 1977.

(2) R.J. Swan, S.H. Fuller and D.P. Siewiorek, "The Structure and Architecture of Cm*: a Modular, Multi-Microprocessor." Computer Science Research Review 1975~1976 CMU.

(3) J.B. Dennis and R.P. Misunas, "A Computer Architecture for Highly Parallel Signal Processing," Proc. ACM Annual Conf., 1974, pp. 402-409

(4) J. Rumbaugh, "A Data Flow Multiprocessor," IEEE trans., vol. C-26, No. 2, 1977, pp. 138-146

(5) P.J. Denning, "Operating Systems Principles for Data Flow Networks," computer, July, 1978, pp. 86-96

(6) P.B. Hansen, "Distributed Processes: A Concurrent Programming Concept," CACM, vol. 21-11, Nov., 1978, pp. 934-941

(7) T. Suzuki and T. Matoba, "Pipeline SAMD Machine and its Applications to Recognition of Printed Chinese Characters," Proc. of 4th IJCPR, 1978.

(8) 元岡達，鈴木達郎，「SAMD (Single Algorithm-Stream Multiple Data-stream)」，昭54電気学会全国大会シンポジウム pp. S13-13 ~ S13-16.