直並列フローで記述される並列パイプライン処理のためのマルチプロセッサ

PARALLEL AND PIPELINED SCHEME USING SERIES-PARALLEL FLOW GRAPH AND ITS REALIZATION ON MULTIPROCESSOR

阿江 忠† 相原 玲二† 栄藤 稔† 森福 茂††

TADASHI AE REIJI AIBARA MINORU ETOH SHIGERU MORIFUKU

† 広島大学工学部 ††現 沖電気（株）

FACULTY OF ENGINEERING HIROSHIMA UNIVERSITY

1. まえがき

並列処理を目的とした計算機システムの研究は、ハードウエアを中心にしたものが多く[1]，これらの研究は並列処理の効用を直接的に示すことができる反面、一般的な設計論への展開にまで至っていないのが、現状である。並列処理システムの設計を論理面と物理面に分けた場合、論理面からのアプローチとして並列処理の理論面を追求するものは、多く見かけられるが、物理面まで結びつけるものは、さほど多くなく、代表的な例はデータフロー計算機[2]に関するぐらいであろう。データフロー計算機はよく知られているようにデータフロー言語を論理記述の手法としても[3]，そのインプリメンテーションのためにデータフローマルチプロセッサが物理的に考えられている[4]。データフロー計算機にみられるトップダウン・アプローチは高く評価されるとして、(i)並列処理の対象を基本演算単位（四則演算など）にまで許すことと、(ii)自由なデータフロー（通信）を許すという点で高速化を目的とする並列処理システムの設計手法にそのまま適用することは難しい。

そこで本稿では、我々の今までの物理面からのアプローチも考慮して[5]~[9]，(i)並列処理の対象は、一つの手続きぐらいのレベルとし、(ii)それら手続き間の通信にも一種の制約を加えた並列処理システムを対象とする。具体的には、本稿は、(ii)の制約として直並列フローで表現される手続き間の通信のみを許した場合を扱っている。むろんこのような論理構造の限定は、すべての並列処理を対象とするとき一つの制約となるが、応用範囲が極端に狭くなっているわけではなく、多くの並列パイプライン処理の記述に対して適用可能である。

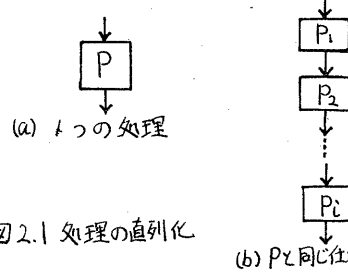
論理構造を直並列フローと限定した場合の特長は、(1)論理面の記述が構造化される。

(2)物理面との対応が容易に行なえる。

という点にある。本稿はまず(1)の直並列フローを記述手段とする論理構造を述べたのち、(2)の物理的実現方法について述べる。後者は、クラスタと呼ぶ基本単位からなりクラスタの階層的な結合により実現される。またこのクラスタは、我々の場合、過去に開発し研究室で利用しているマルチマイクロプロセッサUNIP[10]を用いる。具体例としては、2次元データ処理をとりあげる。

2. 直並列フロー

一つの処理が複数のより小さい処理の列に変換できる場合（図2.1），データ列は、この処理のシーケンスの上をパイプライン的に流れることができ、処理の高速化が行なえる。

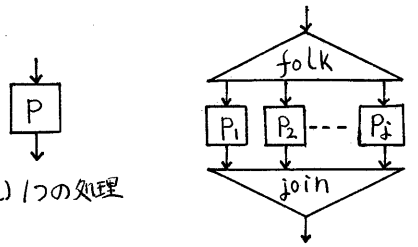


(a) 1つの処理　(b) Pと同じ仕事をするシーケンス

図2.1 処理の直列化

このような処理フローの変換を処理の直列化と呼ぶ。

次に一つの処理Pが複数の(より小さい)処理の束(並列に接続したもの)に置換えられる場合(図2.2), データ列は、分割(folk)されて適当な処理に与えられ、処理結果は、再び結合(join)される。



(a) 1つの処理　(b) Pと同じ仕事をする処理の束

図2.2 処理の並列化

このように一つの処理を並列フローに変換することを処理の並列化と呼ぶ。これら処理の直列化(パイプライン処理)や並列化(並列処理)が処理の高速化に役立つことはよく知られている。処理の直列化や並列化を再帰的に適用すれば、原理的には、処理の高速化が増進されるはずであるが、現実には高速化は、常に限界をもっている。その原因は、処理と処理の間で受渡されるデータの通信に要する時間がオーバヘッドになり、処理の単位が相対的に小さくなるほど無視できなくなる。我々は、データ通信に要する時間を処理時間で正規化したものを通信オーバヘッドと呼び、通信オーバヘッドをパラメータとし、処理の直列化や並列化と高速化の関係を定量化してきた。(5)~(9)

しかしながら、「1つの処理が直列化や並列化できるか?」などの問題とすればどのくらいの数の処理に分割できるか?」などの問題は、処理の内容に依存するし、トップダウン的に処理の分割を繰返す場合、その方法には、自由度がある。本編では、分割の方法自体はユーザが与えることを前提に、論理構造をユーザの記述した処理フローと考え、その記述のための手法を述べる。

処理の最小単位をタスクと呼び、このタスクの論理的順序、並列関係を示すために直並列フローを用意する。

直並列フロー

直並列フローは、次に定義されるモジュールによって構成される。

定義 モジュール

①タスクは、モジュールである。(図2.3)

②モジュールのシーケンスもモジュールである。(図2.4)

③複数のモジュールをフォークとジョインで結合したものもモジュールである。(図2.5)

④上の①,②,③を有限回適用して得られるものは、すべてモジュールである。

□

直並列フローは、以上で定義されたモジュールに一つのbeginと一つのendを前後に加えたものである。(図2.6)

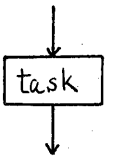


図2.3 モジュール①

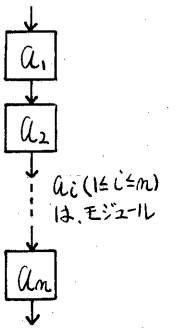
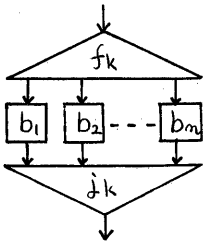


図2.4 モジュール②



bi (1≤i≤m)は、モジュール
fk, jkは、フォーク, ジョイン

図2.5 モジュール③

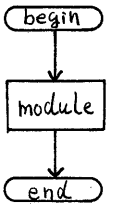


図2.6 直並列フロー

3. 直並列表現

トップダウン設計を行なうために、直並列フローを与えようとすると、描画されたフローから扱わなければならないが、現状では、処理がやや困難であるため、直並列フローと等価な直並列表現という文字列を出発点に置く。

直並列表現は正規表現の部分集合であり、直

並列フローへの変換は容易である。 つまり、次に定義する直並列表現なる文字列を入力したとき、直並列フローを出力するシステム(図3.1)を想定する。

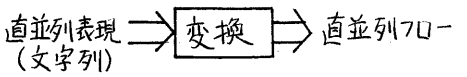


図3.1 直並列フロー描画システム

定義 直並列表現

①σ_i ($i=1,2,\cdots,r$) は、直並列表現である。
(σ_iは、アルファベットの元)
②αとβが直並列表現なら、$\alpha\beta$ も直並列表現である。
③αとβが直並列表現なら、$\alpha|\beta$も直並列表現である。
④上の①,②,③の規則を有限回適用して得られるものは、すべて直並列表現である。

□

以上の定義は、前節のモジュールの定義と一対一の対応をなし、一意的に直並列フローを発生させる。 例えば図3.2のような直並列フローを描画させたければ、$a_1(a_2|a_3|a_4)a_5$の文字列を入力すれば良い。

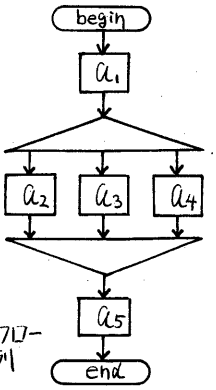


図3.2 直並列フローの例

一般的な手順を次に示す。

1 シーケンスの場合

文字列$\sigma_1\sigma_2\cdots\sigma_m$を入力する。(図3.3参照) このとき$\sigma_i$ ($1\leq i\leq m$)は、モジュールにつける名札である。

2 フォーク/ジョインを用いる場合

文字列($\sigma_1|\sigma_2|\cdots|\sigma_m$)を入力する。

1と2を繰返し用いれば任意の直並列フローの記述ができる。(図3.4)

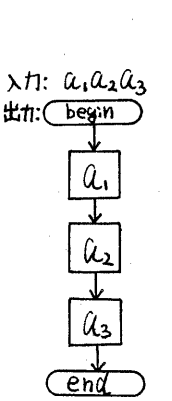


図3.3 直並列フローの例1

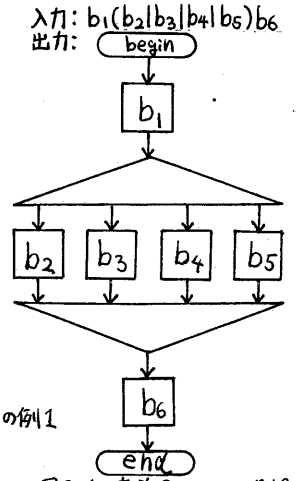


図3.4 直並列フローの例2

なお直並列表現の与え方として、代入(←)もメタ記号として許す。 一つの直並列表現に対する入力形式は一意ではないが、実用上は差仕えない。 例えば、

$main \leftarrow a_1(a_2|a_3)a_4$
$a_1 \leftarrow b_1b_2$
$a_2 \leftarrow c_1c_2$
$a_3 \leftarrow d_1d_2$
$b_1 \leftarrow (e_1|e_2)$

を入力すると、$(e_1|e_2)b_2(c_1c_2|d_1d_2)a_4$や$(e_2|e_1)b_2(d_1d_2|c_1c_2)a_4$のような直並列表現が生成され、図3.5や図3.6が描画される。

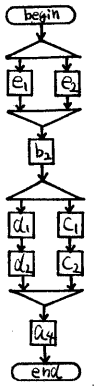


図3.5 生成された直並列フロー その1

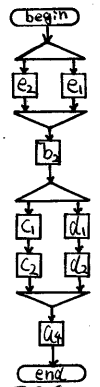


図3.6 その2

4. ハードウェアへの写像

4.1 直並列フローのブロック化

直並列フローで示される論理構造が簡単な場合は、そのまま物理構造に同形写像してよいが、ある程度以上の論理構造になると同形ではなく、(onto)な準同形をとるのが自然である。

もともと計算機は汎用性をもたせかつ、ハードウェア資源を有効利用するべく設計されたものであり、同形写像はワイヤード・ロジック的な対応にあたる。

物理構造の上への(onto)写像を考える場合、物理構造の側の要素数、すなわち、プロセッサ数が

① 固定の場合
② 可変の場合

の2通りが考えられる。（シングルプロセッサを含め）小規模マルチプロセッサの場合は、①に相当し、利用されるべきハードウェア資源に上限があるため、並列パイプライン性も制限をうける。（図4.1）

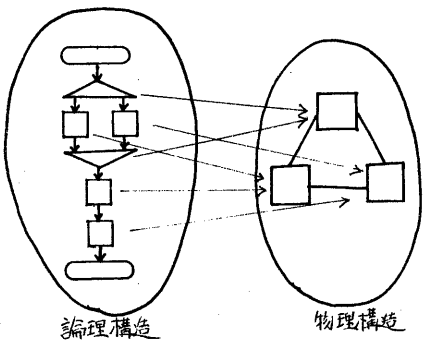


図4.1 論理構造から物理構造への写像の例

一方、半導体製造技術の進歩により最近では多数個マルチプロセッサも容易に製作できるようになったので、本稿では、②の立場をとることにする。 ただし、無制限にプロセッサ数を許すという意味ではなく、論理構造の側から必要な数だけ用意するという意味である。

ところで、必要最小なプロセッサ数を求める問題は易しい問題ではない。 ここでは、論理構造を直並列フローに限っているから、直並列フローの上を流れるデータ・フローに着目する。

直並列フローにおけるカットセットは必ず一方向の流れとなり、両方向のフローが混じることはない。（図4.2）

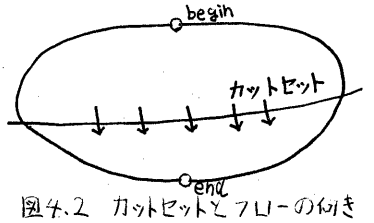


図4.2 カットセットとフローの向き

したがって、直並列フローにおけるデータフローは、カットセットを境界としてデータフローの有効な範囲（プロセッサを専有する範囲）をもっと考えてよい。 むろんこの有効範囲は、beginから始まりendの方へ時刻ごとに物動をする。（図4.3に一例を示す。）

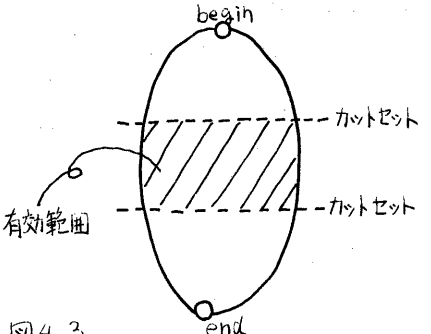


図4.3 データフローの有効範囲

有効範囲を正確に求めることは、必要なパラメータを与えれば不可能ではないが、実際のプロセッサとの対応では、(i)オーバラップの考慮と(ii)カットセット断面における結合ネットワークの問題が入ってくる。 オーバラップの除去は、有効範囲をどの時刻でも重複しないよう拡大することで対処できる。

一方、カットセット断面を結ぶ結合ネットワークは直並列フローゆえに簡単化できるのではないかという期待があるが、ここでは図4.4のようにカットセットを一点に縮約（つまりカットポイント化）して取扱うことにする。（この変換により直並列フローはブロックのシーケンスになる。） 一般的には、ボトルネックを生じる懸念はあるが、ハードウェアとの対応が楽になり、また直並列フローにおいてできるだけカットセットの要素数の少ないところを選べば

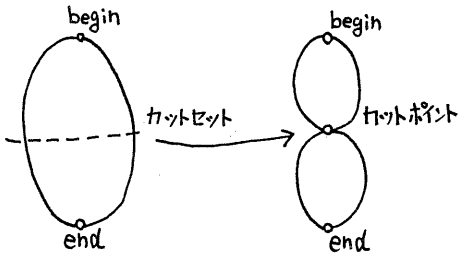


図4.4 カットポイント化

よいであろう。 以下カットポイント化した直並列フローのブロックをクラスタと呼ばれるプロセッサ群に対応させる方針をとる。(図4.5)

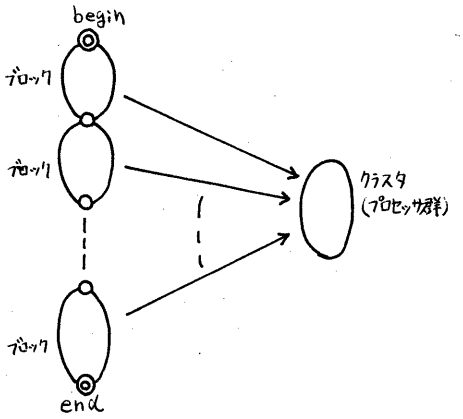


図4.5 カットポイント化した直並列フローのブロックとクラスタとの対応

4.2 ブロックの共通化問題

直並列フローを分解して求めたブロックは、それぞれ，また、直並列フローになる。

したがって、すべてのブロックに共通するハードウェア(クラスタ)を求めるために、ブロックの共通化を行なう。 与えられた直並列フローを $F_1, F_2, \cdots F_m$ なるブロックに分割したとしよう。 まず、直並列フロー F_i が直並列フロー F_k を内包するという定義を与える。

定義 直並列フローの内包

直並列フロー F_i と F_k に対し、F_i にタスクの開放除去(図4.6)または、短絡除去(図4.7)の操作を有限回加えた結果、F_k に同形になるとき、F_i は F_k を内包するといい、$F_i \supset F_k$ と書く。

□

例えば、図4.8の F_1 は、F_2 を内包している。

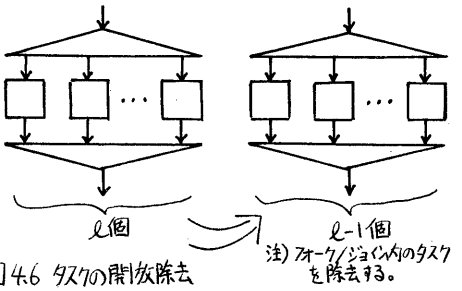


図4.6 タスクの開放除去

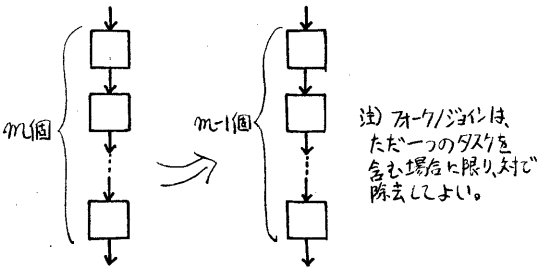


図4.7 タスクの短絡除去

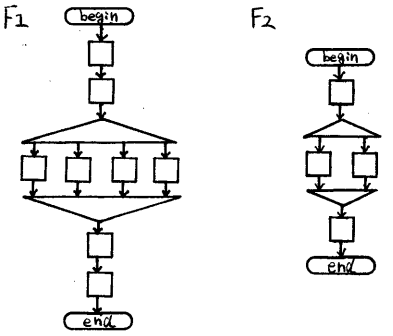


図4.8 直並列フローの内包例

次に、分解されたブロックに共通な直並列フロー F_k とは、すべての F_i $(i=1,\cdots,m)$ に対し、

$$F_k \supset F_i$$

となるものをいう。

ハードウェアの複雑さを考えると、要素数を最小とする F_k が望ましい。

ここでは、必ずしも要素数を最小とするとは限らないが、$O(m)$ の手数で F_k を求める方法を示す。 アルゴリズムの入力は直並列フローと等価な直並列表現とし、そのシンタックスを示す構文木を手段として共通な直並列フロー F_k を求める。

アルゴリズム S

ステップ1　直並列表現 F_i ($i=1,\cdots,m$) に対し、それぞれの構文木 T_i ($i=1,\cdots,m$) に変換する。

ステップ2　すべての T_i ($i=1,\cdots,m$) を含むような構文木 T_α を求める。

ステップ3　T_α を F_α に変換する。

□

例としてブロックが2つの場合を次に述べる。ステップ1は簡単である。図4.9のようなブロック F_1, F_2 の場合、図4.10に示す T_1, T_2 が得られる。

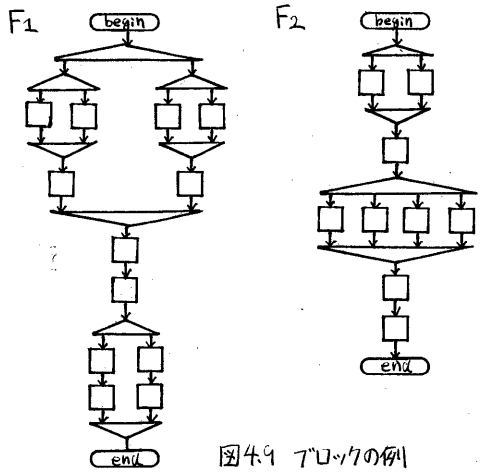
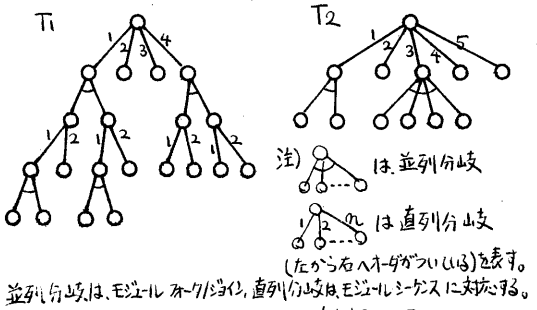


図4.9 ブロックの例



並列分岐は、モジュール オーケストライズ、直列分岐は、モジュール シーケンス に対応する。

図4.10 図4.9 F_1, F_2 に対する T_1, T_2

ステップ2では、T_1 の上に T_2 を含ませるような拡大を行なう。その結果 T_1 が T_α に変換される。(図4.11)

ステップ3では、ステップ1の逆の操作を行ない、構文木から直並列表現に直し、これと等価な直並列フローを求める。図4.11の T_α はこれを経て、図4.12の F_α となる。

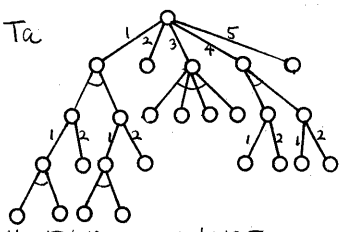


図4.11 図4.10の T_1, T_2 に対する T_α

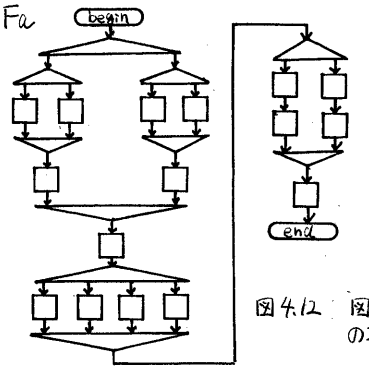


図4.12 図4.11の T_α の場合の F_α

5. ハードウェアモジュール

並列処理システムの設計をトップダウン的に行なうため、論理面からは、直並列フローを出発点とする設計を述べて来たが、縮約された論理構造をハードウェアで実現するため、クラスタと呼ばれるハードウェアモジュールを準備する。クラスタの特長は、(i)標準化されたハードウェアモジュールであること、および(ii)階層的に結合すれば、トータル・システムが構築できる、という点にある。

クラスタのブロック図は、図5.1のようにまん中にバス結合をもつホイール状に接続したマルチプロセサである。

クラスタは、バス制御や外部との通信を行なう一つのインタープロセッサ(IPUと略す)と、これに従属する複数の処理プロセッサ(PUi ($i=1,\cdots,m$) と書く)からなり、IPUと各PUiは、バス結合により結ばれ、また図5.1のようにすべてのプロセッサ間はループ状の結合がなされている。

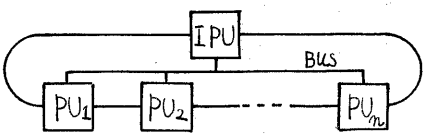


図5.1 クラスタ

直並列表現および直並列フローとクラスタは次のような対応をもつ。

1．直並列表現が $a_1 a_2 \cdots a_n$ の場合、図5.2のような部分直列フローを表すが、これはクラスタの上では矢印のようなデータの流れになる。

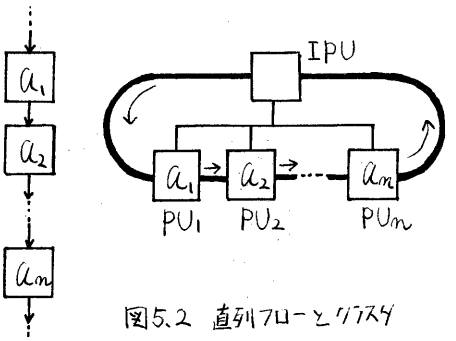


図5.2 直列フローとクラスタ

IPUは、外部から入力を受けとり、これを隣右うPU1にデータを転送する。PU1はPU2へ、順次、処理結果を転送し、最後にPUnを経て、外部へ出力するパイプライン処理をする。

2．直並列表現が $(b_1|b_2|\cdots|b_n)$ の場合、図5.3のような部分並列フローを表すが、これはクラスタの上では、矢印のようなデータの流れになる。

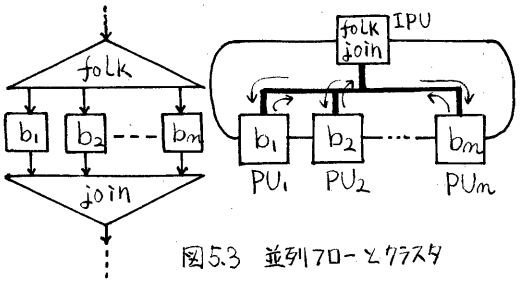


図5.3 並列フローとクラスタ

IPUには、フォークおよびジョイン部分がマッピングされるが、外部からの入力をバスを使って分配したのち、PUiの結果を収集し、出力する2つの処理を受けもつ。

直並列表現からクラスタへの対応は、直並列表現を直列成分と並列成分に分解することでなされる。その方法は一意的ではないが、一つの例を示そう。

$main \leftarrow (b_1|b_2|b_3|b_4)a_2a_3(d_1|d_2|d_3|d_4)c_2c_3c_4$

と等価な

$main \leftarrow a_1a_2a_3a_4$

$a_1 \leftarrow (b_1|b_2|b_3|b_4)$

$a_4 \leftarrow c_1c_2c_3c_4$

$c_1 \leftarrow (d_1|d_2|d_3|d_4)$

なる直並列表現を用いた場合、最上位レベルで対応する直並列フローおよびクラスタは、図5.4になる。

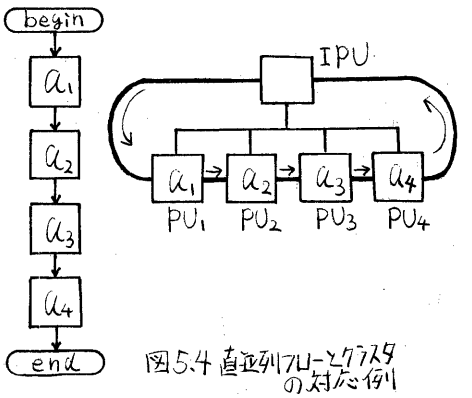


図5.4 直並列フローとクラスタの対応例

つづいて、a_1, a_4, c_1 にも同様の対応を行なえば、図5.5のような階層的なハードウェア構造が得られる。これは、一つのルーチンを並列またはパイプライン処理に置換えるときの、論理構造とハードウェア構造との対応を示しており、論理構造に対して柔軟にハードウェアも対応しうることを示している。

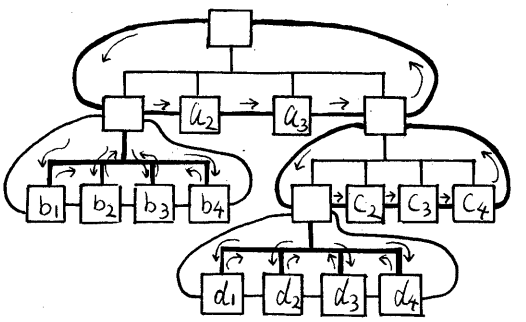


図5.5 直並列フローとクラスタの階層的な対応例

6. 設計および実験例

6.1 一般的な考え方

直並列フローで記述された並列パイプライン処理をハードウェアとして実現しようとするとき、プロセッサの数に低い限度を設ける必要はないけれども、無制限に多くしても意味はない。結局、適正な上限があり、単純な直列フローや並列フローでは、この数を解析的に求めることもできる[9]。

直並列フローは直列フローと並列フローの組合せからなるから、一般の直並列フローの場合の処理時間の解析も原理的には可能である。

具体的な設計条件を考えると、およそ3通りの制約が考えられる。

(i)処理時間の制約

処理時間の上限が与えられ、それを満足するよう並列パイプライン化を進めたときの構造を求める。

(ii)ハードウェアの制約

全体の構造には、ハードウェアの条件から一つの制約があり、その範囲で最良の構造を求める。

(iii)一般的な制約

並列パイプライン化を処理時間の向上が飽和するところまで進めたときの構造を求める。

これらのどの制約のもとで考えるべきかは、ニーズと環境に依存するが、実際的には(i)または(ii)の制約を考えるのが妥当であろう。設計システムとしては、図6.1のような流れを想定している。

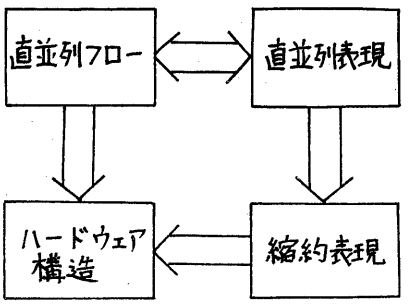


図6.1 設計システムの流れ

6.2 ハードウェアの制約からの具体例

一般的な設計システムはまだつくられていない。一方、すでに約40のプロセッサ・ボードの手持ちがあるので、クラスタを実現し、実験を試みた。

具体的には

処理1 データの特徴抽出(GA)

処理2 記憶あるいはパターンマッチ(REC)

というシーケンシャルな処理フローが基本である。(図6.2)

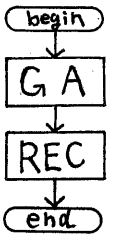


図6.2 基本処理フロー

データは2次元ビットパターンであり、学習モードではIDをもつ2次元図形に対してGAで特徴抽出を行ない、RECでその特徴データを格納する。認識モードになると、入力された2次元データをGAでは同じく特徴抽出し、RECではパターン・マッチによりサーチして、特徴が同じになるIDをさがしだす。(図6.3)

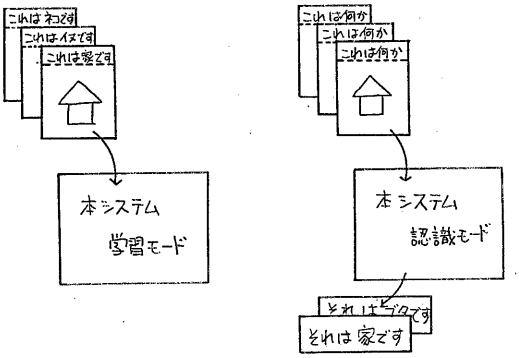


図6.3 2つの動作モード

次に、並列パイプライン処理を行なうため、

① GAは並列フロー化

② RECは直列フロー化

を導入する。

GAはi個の処理を並列に、RECはj個の処理を直列に行なうものとする。（図6.4）

具体的な2次元データとしては簡単な三角形や四角形の組合せを用いている（付録参照）。

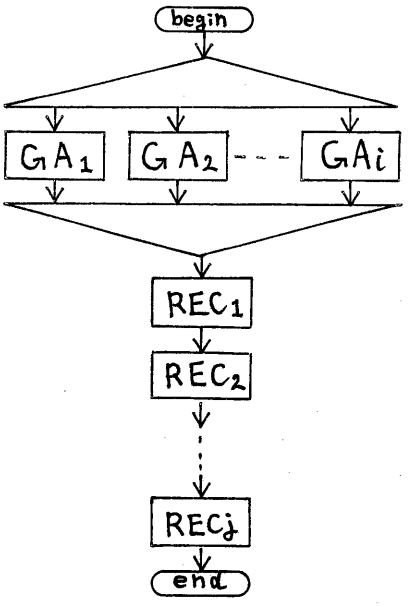


図6.4 直並列フロー

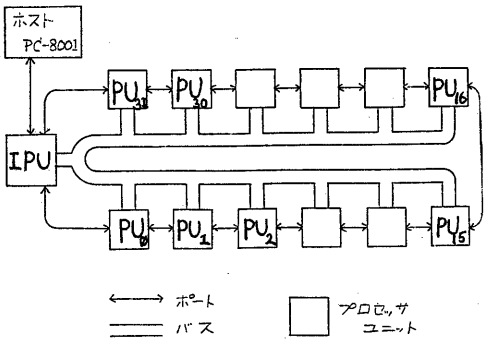


図6.5 UNIPハードウェア構成

図6.4のような直並列フローを先に述べた方法でハードウェア・モジュールであるクラスタに写像するとクラスタが2個必要になるが、今回の実験では図6.5に示すマルチマイクロプロセッサUNIPの上にマッピングした。

ただし、UNIPはIPUを除いて1ラックにPUを32もつが、32個のPUのうちデータ転送などに必要なものを除いて29個がGAとRECのために用意される。

実験ではGAのための並列化の数iとRECのための直列化の数jが、

$$i + j = 29$$

という条件のみで、設定を可変にできる。

1024件の入力データをiとjを変えて処理したときの処理時間の計測結果は、図6.6のようになった。パラメータとして、格納したデータの数をとっている。

この結果によると、たとえば、格納データ数320に対しては、i=9, j=20が最良となる。

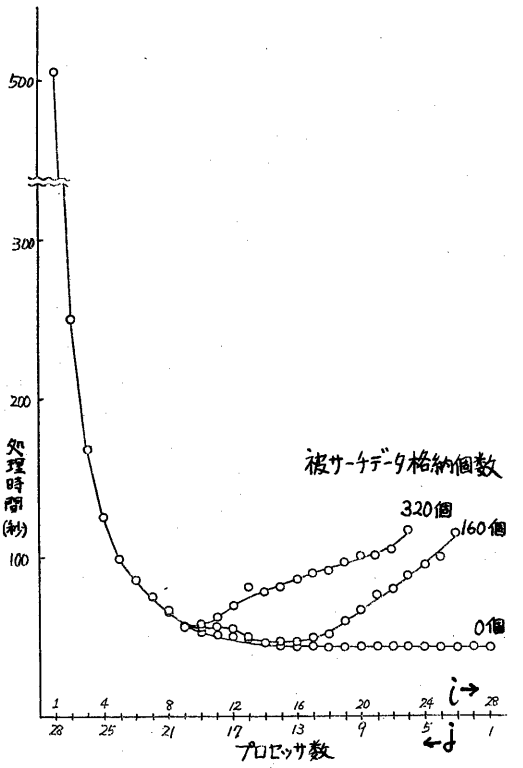


図6.6 実験結果

なお、プロセッサユニットはIPU, PUiともにZ-80CPUであり、処理はアセンブラで記述されている。

7. むすび

本稿では、論理構造に直並列フローを用いて並列パイプライン処理を記述したのち、直並列フローよりハードウェア構造を実現する方法を述べた。

マルチプロセッサに関しては、従来、ハードウェアが先に与えられたのち、その上に論理構造を実現するというやり方が主であった。しかしながら、VLSIなど製造技術の進歩により多数個マルチマイクロプロセッサの実現が容易になった最近の環境を考えると、設計から製造へ向かう方法、すなわち、論理構造を与えてハードウェア構造を求める手順の方が望ましい。

むろん、LSI/VLSIチップの設計／製作に比べて一段階上のシステムレベルを扱うため、トータル・システムをただちにつくることは困難である。そこで本稿では、このようなトータル・システムへ向かっての第一歩として、比較的性質のとらえやすい論理構造として直並列フローの場合をとりあげた。

本稿では、処理フローという概念的な表現を手段とし、直並列処理を記述するための言語はとくに定めていない。言語のレベルを考えると、繰返しをあらわす制御構造も必要となるが、while-doのような制御構造をあらわす図式(D-chart)の導入は容易である。ただし、このような繰返し構造を陽に導入した場合、並列処理を妨げるような影響を与える可能性もあり、今回は陽には表現しないという立場をとっている。

具体的な言語はこの研究に対してはまだ規定していないが、別途、マルチプロセッサ上の並行処理言語として、すでに検討や試作も行なっており[11][12]、やがては、融合させる方針である。

現在、計画を急いでいるのは、むしろ、直並列フローおよび各タスクの処理時間とタスク間の通信に要する時間を与えたとき、直列化と並列化の異なる種々のスキーマに対して、トータルの処理時間が直ちに求められるような設計シミュレータの製作である。特定のスキーマでも、直並列フローでない場合は、非常に時間がかかるものとなるし[13]、任意の構造を記述できるスキーマに対するシミュレータの製作は非常に難しい。

本稿で着目した直並列フローは、その点、実現は易しいし、また直並列表現と簡単に対応づけられるから、ユーザ・インターフェースの製作も、比較的簡単である。

本稿で示した実例は、既存のハードウェア上に直接実現したため、プロセッサも約30に制限されている。今後、シミュレータ上ではプロセッサ数100～1000という並列度の大きい場合をとり扱えるようにして、直並列フローで記述できる並列パイプライン処理の適用範囲について明らかにしていきたい。

文献

(1) Special Issue on Performance Evaluation of Multiple Processor Systems, IEEE Trans. Computer, vol. C-32, No. 1 (Jan. 1983)

(2) P.C. Treleaven et al.: "Data-Driven and Demand-Driven Computer Architecture", ACM computing Surveys, Vol. 14, No. 1 (Mar. 1982)

(3) J.B. Dennis et al.: "Data Flow Schemas", International Sympo. Theoretical Programming, Lecture Notes in Comuter Science 5, PP. 187-216, Springer-verlag (1974)

(4) J.E. Rumbaugh: "A data flow multiprocessor", IEEE Trans. Computer, Vol. C-26, No. 2, PP. 138-146 (Feb. 1979)

(5) 阿江, 高橋, 松本: "共有Xモリ結合によるマルチマイクロプロセッサの並列動作について," 信学論(D), J65-D, No. 3, pp. 322-329 (昭57-03)

(6) 阿江 他: "マルチマイクロプロセッサの応用-並列パイプラインモジュールPPM-," 信学技報, EC81-39 (1981-10)

(7) 相原, 阿江: "並列パイプラインプロセッサUNIPの応用", 信学技報, EC82-31 (1982-06)

(8) 相原 他: "多数個プロセッサ・アレイ・バス・アレイ", 信学技報, EC82-64 (1982-12)

(9) T. Ae and R. Aibara: "Experimentation and Analysis of Multiprocessor Systems", Proc. Real-Time Systems Sympo., pp. 69-80 (Dec. 1982)

(10) 阿江, 相原: "並列パイプラインプロセッサUNIP", 昭和57年度信学会総全大, S1-3 (昭57)

(11) T. Ae and S. Tenma: "A Real-Time Processing Language for Controller", Proc. International Computer Symposium 82, pp. 770-778 (Dec. 1982)

(12) T. Ae et al.: "A Distributed Real-Time Processing Language on Multi microprocessor System", IEEE Proc. Real-Time Systems Sympo., pp. 20-29 (Dec. 1983)
(13) 相原, 斎藤, 阿江: "付加マルチプロセッサ バス・アレイにおける並列処理の一検討", 昭和58年度信学会情報システム部門全大, 559 (昭和58)
(14) 阿江, 相原: "マルチプロセッサシステムのための光結合共有メモリ", 情処学会第28回全大, 3C-7 (昭59)

付録 実験に用いた2次元データ例