オペレーティング・システム　56−11
(1992. 8. 19)

Datarolアーキテクチャにおけるスレッド実行機構に関する考察

川野哲生　星出高秀　日下部茂　谷口倫一郎　雨宮真人

九州大学大学院総合理工学研究科情報システム学専攻

我々はデータフローにメモリの概念を導入したDatarolアーキテクチャを提案してきた．Datarolとはデータフローグラフから余分なデータフロー制御を取り除いて最適化したマルチスレッドコントロールフロープログラムである．Datarolマシンは循環パイプライン型の計算機であり、プログラム中に十分な並列性が存在し循環パイプラインが満たされている場合においては高いスループットが得られる．しかしながら，並列性の少ない逐次的な処理に対しては循環パイプライン上に空きができスループットが低下する問題点がある。

本稿では、Datarolアーキテクチャ上に排他的なスレッドの実行を導入することによる逐次的な処理に対する効率改善、及びスレッド実行機構を設けたDatarolマシンの構成法について述べる。

Thread execution mechanism on Datarol architecture

Tetsuo KAWANO, Takahide HOSHIDE, Shigeru KUSAKABE,
Rin-ichiro TANIGUCHI and Makoto AMAMIYA

Department of Information Systems, Graduate School of Engineering Sciences
Kyushu University
6-1 Kasuga-koen Kasuga-shi Fukuoka, 816 Japan

We have proposed a Datarol architecture. The idea of Datarol is to eliminate redundant dataflow by introducing registers and by-reference concept. The Datarol program is a multithread controlflow program. The dataflow machine is consisted of circular pipeline, so that it can achieve high throughput when program has efficient parallelism and pipeline is filled. However, it suffers inefficiency when the program has poor parallelism.

In this paper, we introduce a thread execution mechanism on the Datarol architecture in order to support efficient execution for a sequential program. We also describe a configuration of Datarol processor which incoporate thread execution mechanism.

1 はじめに

我々はデータフローアーキテクチャの問題点を解決するアーキテクチャとしてメモリの概念を取り入れた Datarol アーキテクチャ[1][2][3] を提案してきた．Datarol マシンは 1000 台規模の Datarol プロセッサを多段結合網で結合したものである．Datarol プロセッサは循環パイプラインアーキテクチャを基に設計され，Datarol と呼ぶマルチスレッドコントロールフローグラフを実行する．Datarol はデータフローグラフに対してレジスタの割り付けや冗長なリンクの削除を施し，最適化したプログラムである．

Datarol マシンはプログラム中の並列性を実行時に自然に抽出し，高並列多重処理を行うことが出来る．しかしながら，循環パイプライン型のアーキテクチャでは，並列度の低い逐次的なプログラムに対しては循環パイプライン中に空きが生じ処理効率が低下する問題がある．そこで，本稿では Datarol アーキテクチャにスレッドの概念を導入することによりこの問題を解決し，細粒度のものから粗粒度のものまで様々な粒度のプログラムに対して効率的実行を行うことのできるアーキテクチャの考察を行う．

本稿では，まず 2 章で Datarol アーキテクチャについて述べる．3 章では，スレッドの概念を導入した Datarol アーキテクチャについて述べ，4 章でスレッド実行機構を導入したプロセッサエレメントの構成について説明する．5 章ではスレッドを用いたプログラム例について述べる．最後に 6 章で本稿の結論と今後の課題について述べる．

2 Datarol アーキテクチャ

データフローアーキテクチャは，プログラムに内在するすべての並列性を実行時に自然に引き出すことができ，またプロセススイッチをハードウェアで高速に実行できることから並列計算機アーキテクチャを考える際に重要なベースを与える．しかし，データフローアーキテクチャをそのままハードウェアで実現しようとすると，通信のオーバーヘッド，フロー制御のオーバーヘッド，メモリレス概念の弱点，といった問題が生じる．

筆者らは，データフローアーキテクチャのこのような欠点を改善する超多重処理向きのアーキテクチャとして，データフローアーキテクチャに by-reference メカニズムとメモリ概念を取り入れた Datarol アーキテクチャを提案してきた．

by-reference メカニズムとメモリの概念を採り入れた Datarol モデルは従来のデータフローモデルと比較して以下のような特徴を持つ．

- by-reference メカニズムではオペランドデータをシェアして用いるので，ある演算で作り出された値がいくつかの演算で参照されるとしてもそのデータのコピーを作る必要がない．
- オペランドデータを格納するメモリのアドレスはコンパイル時に決定できるので実行時のメモリアロケーションの必要が無く，マッチング・ストアのハードウェアコストを削減できる．
- by-reference メカニズムではある演算で用いるデータは，そのデータを生成する演算の実行が終了した時点で直ちにオペランドメモリに書き込まれるので，そのデータをオペランドとして用いる演算が実行されるときにはそのデータの存在が保証されており，マッチング・ストア操作のオーバーヘッドを削減できる．

Datarol プログラムはデータフローグラフに対してレジスタの割り付けや冗長なリンクの削除を施し，最適化したプログラムである．Datarol プログラムの実行は循環パイプラインを用いて行われる．スレッドの実行は排他的には行われず，実行時に他のスレッドとインターリーブされる．よって，同一インスタンス，あるいは他のインスタンスに拘らずスレッドはインターリーブされ実行する。これにより、プログラム中に十分な並列性があるとき，循環パイプラインは満たされ多重並行処理を効果的に行うことができる．

3 スレッド概念の導入

Datarol アーキテクチャでは各命令毎に継続命令の指定を行っている．これによりプログラム中に内在する並列性を最大限に利用できる．一方，並列度の低い逐次的なプログラムに対しては循環パイプラインに空きが生じ処理の効率が低下する．また，Datarol では可能な部分についてすべて並列展開を行なが，同一インスタンス内の各 activation は同一 PE 上で実行されるため，結局循環パイプライン上で逐次化されてしまう．逆に，同一インスタンス内での必要以上の並列展開は余分なペアオペランドマッチングの増加を招き，これにより FU 稼働率を低下させる原因にもなる．

このような逐次的な処理に対する非効率性や，必要以上の並列展開を防ぐため Datarol アーキテクチャにスレッドの概念を導入する．スレッドとは，外部とのインタラクションを持たない一気に実行することが出来るプログラムブロックである．すなわち，同期処理はすべてのスレッドの入り口で行い，連続したアドレスに並べたスレッド内の命令を逐次的に実行するというモデルである．また，不要な並列展開を抑制し，1 つのスレッド中に納めることによりペアオペランドマッチングの回数を削減する．

以下，3.1 で Datarol におけるスレッドモデルについて述べ，3.2 で Datarol グラフからスレッドへの変換法を示す．また，3.3 では実行時におけるスレッドの連結について述べる．

3.1 実行モデル

図 1 に Datarol におけるスレッドのモデルを示す．各スレッドは入り口の同期処理の部分とスレッド本体部分および継続スレッド起動のための fork 命令部分とに分けられる．スレッドは明示的な fork 命令および暗黙的な fork 命令 (call, link, rlink, read, etc.) により起動される．スレッドの入り口ではこの fork 命令に対して同期処理を行ない，同期に成功した時点でスレッド本体の実行が行われる．

ここで，スレッドの起動制御部 (fork,join) とスレッドの本体部分とを分離することにより，それぞれ起動制御部 (AC)

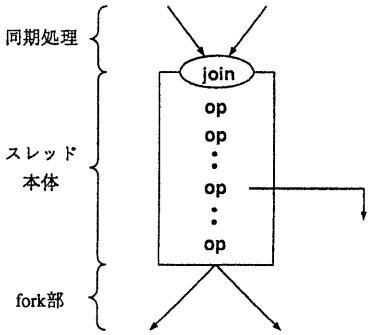


図 1: スレッドモデル

と関数実行部 (FU) で処理できる．このとき，AC でスレッドの起動制御を行い FU へタスクディスパッチを行なう形で処理が行われる．通常 fork 命令は AC 内で発行され AC 内で処理されるが，レイテンシの伴う命令や sw 命令による暗黙的な fork （後述）は FU 内で発行されるため，これらについては図 1 中に示すようにの横方向へのアークで表わし区別する．

3.2 Datarol グラフからスレッドへの変換

ここでは Datarol グラフからスレッドグラフへの変換法を示す．Datarol グラフからスレッドグラフへの変換は，

1. ノードの置換
   Datarol グラフ中の各ノードをそれぞれ独立したスレッドに置き換える．

2. スレッドの合成
   1. で得られたスレッドをもとにスレッドの合成を行い，細かなスレッドをつなぎ合わせてより大きなスレッドを得る．

の 2 つの段階で行う．

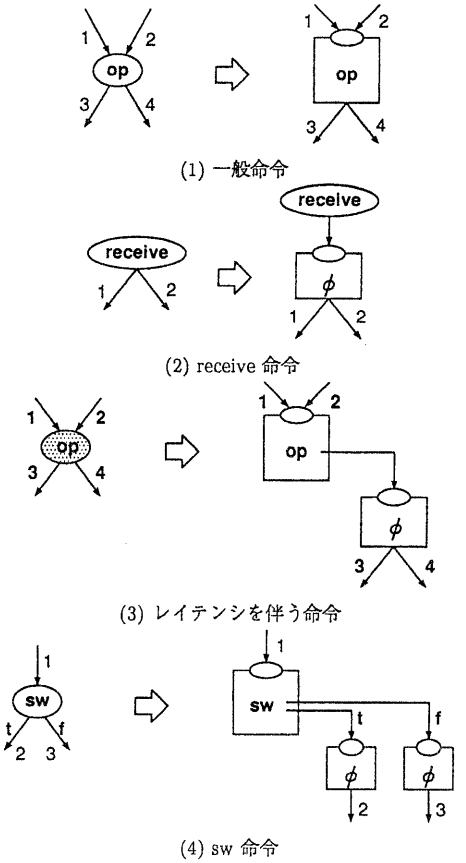
3.2.1 ノードの置換

Datarol グラフ中の各命令（ノード）は以下のように分類できる．

1. 一般命令
   加算・減算等の算術演算命令やビット操作，論理演算等の一般的な命令．(結果がすぐに利用可)

2. receive 命令
   関数の引数の受け渡しに用いる命令．

3. レイテンシを伴う命令
   リモートメモリアクセス (read, write) やリモート手続き呼び出し (call, rlink) 等の結果の到着にレイテンシを伴う命令．



(1) 一般命令

(2) receive 命令

(3) レイテンシを伴う命令

(4) sw 命令

図 2: ノードのスレッドへの置換

4. SW 命令
   オペランドデータの真偽値により分岐を行う命令．

ここでこれらのノードについて，それぞれ図 2 に示すように，スレッドへの置き換えを行う．また，ノード間のリンクはスレッド間のリンク (fork, join) へと置き換えられ，ノードの示す命令がスレッドの本体となる．レイテンシを伴う命令については結果の到着により新しいスレッドが起動されるように置き換えられる．すなわちレイテンシを伴う命令は暗黙的な fork 命令となる．sw 命令についてはその真偽値により 2 つの内の一方のスレッドが起動される形になる．

3.2.2 スレッドの合成

ここでは，3.2.1 で得られたスレッドグラフをもとに，スレッドの合成を行い，より大きなスレッドを作るとともに，スレッド間のリンクの削減を行い，スレッド起動制御のオーバーヘッドを軽減する．以下，スレッド合成の 2 つの方法について述べる．

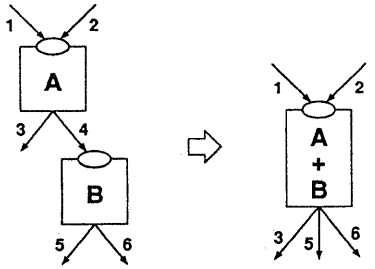


図 3: スレッドの合成 (I)

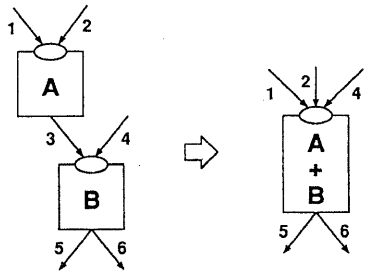


図 5: スレッドの合成 (II)

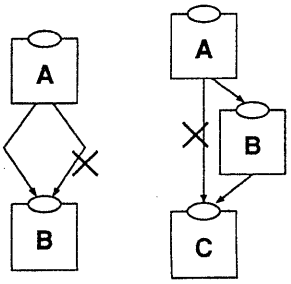


図 4: スレッド間の冗長なリンクの削除

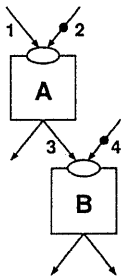


図 6: スレッドの連結

スレッドの合成 (I) 図 3 に示すように，スレッド A, B があり，スレッド B がスレッド A の fork 命令のみによって起動される場合（スレッド B の fun-in = 1），スレッド B はスレッド A に含めることが出来る．ここで，スレッド B からの fork 命令はスレッド A へ引き継がれる．

また，上記スレッドの合成により図 4 に示すようなスレッド間の冗長なリンクを生じた場合はそれを削除する．

ここでのスレッドの合成は，

1. 逐次処理部分のスレッド化
2. 不必要な横方向の並列展開の逐次化

を行っている．

スレッドの合成 (II) 図 5 にスレッドの合成法 (II) を示す．同図で，スレッド A の本体が小さいとき，右図に示すようにスレッド A, B の同期処理まとめて行い，スレッド A, B を一括して実行することによりコンテクストスイッチングの回数を減らすことができる．このような合成を行っても他の部分の実行に影響を及ぼさないためにはスレッド A は以下の条件をすべて満たさなければならない．

1. スレッド A の fork 命令はスレッド B の起動用のみ（スレッド A の fun-out = 1，スレッド B の fun-in 数は問わない）
2. スレッド A 中にはレイテンシを伴う命令や sw 命令等の他スレッド起動する命令が含まれない
3. スレッド A 中には return 命令が含まれない
4. スレッド A 中の命令数はある閾値以下である

ここで，条件 4 中の閾値はコンテクストスイッチに伴うオーバーヘッドにより決まる定数である．

3.3 実行時におけるスレッドの連結

前節で述べたスレッドの合成によりスレッドを長くすることによりコンテクストスイッチングによるオーバーヘッドの削減が行える．本節では実行時に同一コンテクスト（インスタンス）内の実行可能なスレッドを連続して実行することにより，さらにコンテクストスイッチングを減らす手法を示す．

図 6 において，アーク 2，4 上の fork は既に実行されておりこの状態で，アーク 1 上の fork が実行された場合を考える．このとき，まずスレッド A のタスクディスパッチが行われる．この時点で，スレッド A とスレッド B の実行順序は保証される．すなわちスレッド A のタスクディスパッチの直後にスレッド B のタスクディスパッチを行うことができる．これにより，FU は同一インスタンス内のスレッドを連続して実行することになり，コンテクストスイッチングのオーバーヘッドを削減できる．

4 スレッド実行機構を導入したプロセッサエレメントの構成

図 7 にスレッド実行機構を導入したプロセッサエレメントの概略図を示す．同図に示すように各プロセッサエレメントは FU(Function Unit), MU(Memory Unit), AC(Activation Controller), CU(Communication Unit), RQ(Ready Queue), MQ(Matching Queue) から成る．Datarol プログラムはスレッドの同期・起動用命令部分とスレッド本体部分とに分け，それぞれ AC 内と FU 内に納められ，それぞれ AC プログラム，FU プログラムと呼ぶ．また MU には各インスタンス毎に割り付けられたレジスタファイルの内容が格納される．

MQ にはスレッドの起動要求メッセージ <ins,ACaddr> が，また RQ にはスレッドの実行要求メッセージ <ins,FUaddr> がストアされる．AC は MQ からメッセージを取り出し，AC プログラムに基づきスレッドの起動制御を行う．AC で活性化されたスレッドは RQ へと送られる．FU はこれを RQ から取り出しスレッド本体の実行を行う．また，FU ではリモートメモリアクセスやリモート手続き呼出し等のメッセージも作られ，これらは CU を経てネットワークへ送出される．CU はネットワークより到達したメッセージに対し，データのメモリへの書き込み，AC へのスレッド起動要求の送出，call 命令に対する新しいインスタンスの割り付け等の処理を行う．

以下，FU, MU, AC の各部について説明する．

4.1 FU(Function Unit)

AC で活性化されたスレッドは RQ へストアされ，FU はそれを RQ から取り出し実行する．FU は図 7 に示すように内部に 3 枚程度のレジスタセットを持つ．このレジスタセットには MU からレジスタファイル内容が読み込まれ，これを用いて，FU 内でのスレッド実行を行う．

FU は PC (Program Counter) をもち一般の RISC 型のプロセッサと同様の 3 アドレス形式の命令（図 1参照）を実行する．FU の命令形式は

```
[l]:[!] w <- op(u v) [-> D]
```

で，ここで，[ ] は省略可能なことを示し，l はプログラムのアドレスを，u, v はオペランドのレジスタ名，w は結果の書き込みレジスタ名を示す．また，! はその命令でスレッドが終了 (terminate) することを表す．D はレイテンシを伴う命令等の結果が到着したとき新しいスレッドを起動するための AC プログラムのアドレスである．

FU でのスレッドの実行は，以下のようにして行われる．まず，RQ からスレッド実行要求のメッセージを取り出す．メッセージ中のインスタンス名を用いて，MU からレジスタファイルを FU 内のレジスタへ読み込む．次に，メッセージ内にある FU プログラムアドレスを PC にセットしスレッドの実行を開始する．terminate 命令に到達したらそこでスレッドの実行を終了し，レジスタの内容を再度 MU へ書き込む．以下これをくり返し行う．

w <- op(u v)	normal operation
w <- call(u) -> D	get new instace
link(# u v)	link data
w <- rlink(# u) -> D	link return regiser name
w <- read(u) -> D	remort memory read
write(u v) -> D	remort memory write
sw(u) -> D1|D2	switch operation

表 1: FU 命令

ここで，FU と MU 間のレジスタデータの入出力は FU 内の異なるレジスタセットを使用し，スレッドの実行と並行して行う．また，レジスタ内容の読み込みは RQ からのデータを先読みし，次に実行されるスレッドに対し先行制御を行う．これにより，高速なコンテクストスイッチングが行える．

ただし，FU と MU 間のデータ転送能力が低い，あるいは FU で実行されるスレッドの平均長が短い場合，レジスタ内容の転送が FU での実行に間に合わなくなり，FU がアイドル状態になることがある．したがって，FU と MU 間の転送能力を高くするとともにスレッドの長さをある程度以上にする必要がある．

4.2 MU(Memory Unit)

MU では各インスタンス毎に割り当てられたレジスタファイルの内容の保持をおこなっている．MU には図 7 に示すように 3 つのポートがある．これらは，

- **FU ポート**
  FU とのレジスタ内容の転送に使用される．データの転送はレジスタファイル単位で行われ，高速なコンテクストスイッチングを実現するためにもっとも速度の要求されるポートである．

- **CU ポート**
  CU で受け取ったメッセージに対してその内容のレジスタファイルへの書き込みをおこなう．

- **外部メモリポート**
  レジスタファイルの外部メモリへのスワップに使用される．

である．また MU 内のメモリにはレジスタファイルの内容が格納されている．このメモリには高速な 2 ポートのメモリを使用し，FU ポートの高速な転送能力を実現する．

4.3 AC(Activation Controller)

AC 内には PC (Program Counter), スレッド間の同期処理を行うための MM (Matching Memory), AC 内の fork 命令により発行されたスレッド起動要求メッセージを格納する ACQ (AC Queue) がある．AC ではこれらを用いてスレッドの起動制御をおこなう．

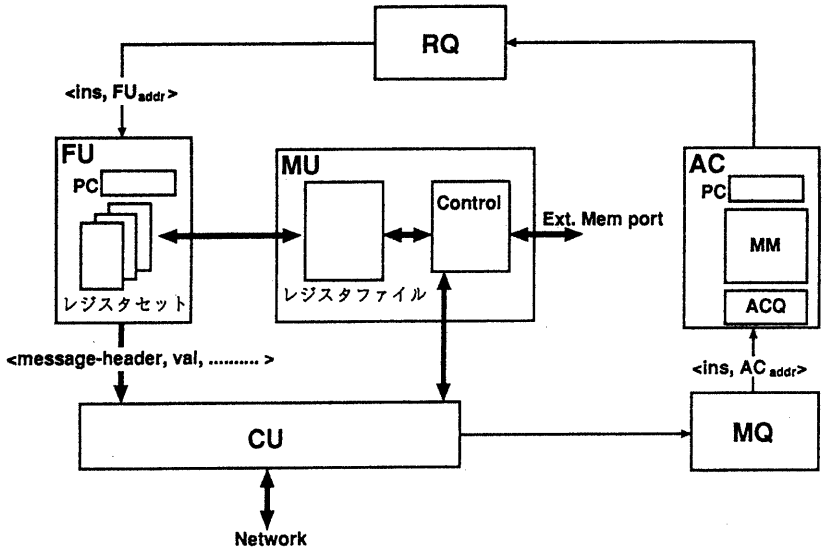


図 7: スレッド実行機構を導入したプロセッサエレメントの構成

AC 内での処理は AC プログラムにより制御される．AC プログラムで用いられる命令は send と fork の 2 種類だけであり，その命令形式は以下の通りである．

```
[l]:[!] op addr [matching_no]
```

ここで，l がプログラムアドレスを ! が AC プログラムの terminate 点を表す．addr には FU プログラムアドレス（send 命令の場合）あるいは AC プログラムアドレス（fork 命令の場合）が入る．また，matching_no はそこで指定されたビットを使用してマッチング (join) を行うことを意味する．AC での処理は MQ からスレッド起動要求メッセージを取り出し，メッセージ中の AC プログラムアドレスを PC にセットすることにより開始される．AC 内の処理は PC によって AC プログラムを順に実行し terminate を検出した時点でまた新たなスレッド起動要求メッセージを読み込み以下これを繰り返す．以下 send, fork 命令それぞれについて説明する．

- **send 命令**
  MM には各インスタンスに対するマッチング用のビットフィールドがある．send 命令では matching_no で指定されたビットを用いて binary matching を行い，マッチング成功時には インスタンス名と FU アドレスを RQ に送りタスクディスパッチを行う．マッチング不成功時には AC プログラムの実行は terminate する．matching_no 省略時には無条件にタスクディスパッチを行う．

- **fork 命令**
  send 命令と同様にマッチングを行い，マッチング成功時には ACQ に AC アドレスをセットする．ACQ 内に格納された AC アドレス情報はスレッド起動要求メッセージとして MQ に対して優先して扱われる．これにより，実行時におけるスレッドの連結機能を提供する．マッチング不成功時には terminate する．また，terminate マークのついた fork 命令は jump 命令として用いることができる．

AC では，この send, fork の 2 命令だけを用いて多入力同期や実行時におけるスレッドの連結等の多様なスレッド間の同期処理を行うことができる．

また，スレッドのサイズが小さくなるにつれ AC での処理に高速性が要求され，AC がボトルネックとなる可能性が大きくなる．そこで，AC で頻繁に用いられる join 操作とタスクディスパッチ，join 操作とスレッド起動をそれぞれ 1 命令で行うことにより，AC 内の処理を高速に行う．また，シンプルなハードウェア構成も処理の高速化に貢献する．

5 スレッド概念を導入した Datarol プログラム例

本節では，スレッドの概念を導入した Datarol プログラムについて fibonacci 数を求めるプログラムを例に説明する．

図 8 に fibonacci 数を求める Datarol グラフを，図 9 にスレッド化したグラフを示す．また，Datarol プログラムリストを図 10 に示す．ここで，図 8 中，破線で囲ったノードが 1 つのスレッドへと変換されている．

以下図 10 のプログラムについて簡単に説明する．

Datarol プログラムの実行は親インスタンスからの link, rlink 命令により開始される．この例では親インスタンスで

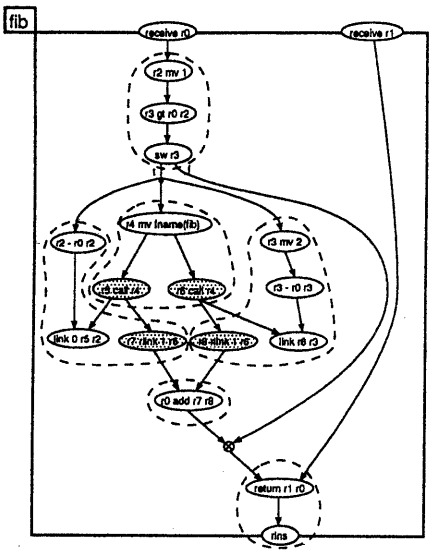


図 8: Fibonacci 数を求める Datarol グラフ

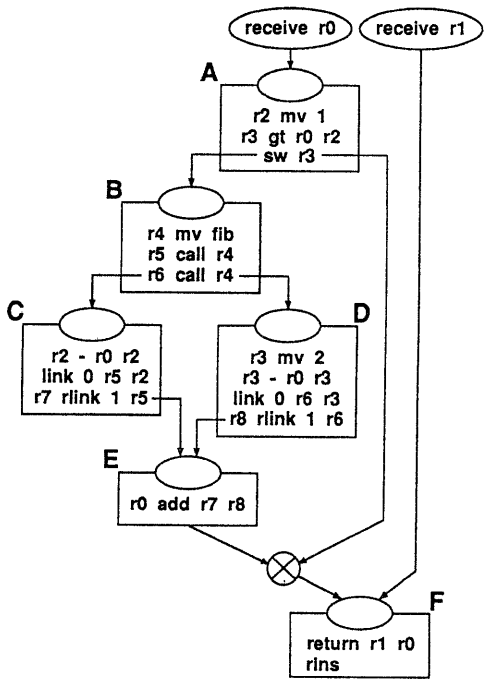


図 9: Fibonacci 数を求めるスレッドグラフ

```
;thread A
300: r2 <- mv 1
301: r3 <- gt(r0 r2)          ; ? n > 1
302:!       sw r3 -> 102 | 101
;thread B
303: r4 <- mv fname(fib)
304: r5 <- call(r4)
305:!r6 <- call(r4)
;thread C
306: r4 <- -(r0 r2)
307:        link(0 r5 r4)
308:!r7 <- rlink(1 r5) -> 105 ; r7<-fib(n-1)
;thread D
309: r3 <- mv 2
310: r3 <- -(r0 r3)
311:        link(0 r6 r3)
312:!r8 <- rlink(1 r6) -> 105 ; r8<-fib(n-2)
;thread E
313:!r0 <- +(r7 r8)
;thread F
314:        return(r1 r0)
315:!       rins
```

(a) FU Program

```
; FP(fib)=100
100:!send 300    ; by link(0 ins n)
                 ;   (r0 <- n)
101:!send 314 1 ; by rlink(1 ins ret_addr)
                 ;   (r1 <- ret_addr)
102:!send 303
103:!send 306
104:!send 309
105: send 313 2
106:!fork 101
```

(b) AC Program

図 10: Fibonacci 数を求めるプログラム

link 命令により第 1 番目の引数 n が送られたとき，CU でレジスタ r0 に書き込まれた後，CU は MQ に関数 fib(n) の AC 内での先頭アドレス 100 にオフセット 0 を加えたアドレスをセットする．これにより引数 n の到着により AC プログラムのアドレス 100 から実行がはじめられる．ここでは send 命令により FU へタスクディスパッチが行われ，FU でスレッド A が実行される．同様に，第 2 番目の引数（計算結果の返し先）も CU でレジスタ r1 へ書き込まれた後，MQ にアドレス 101 をセットし，AC ではこれを受け取りスレッド F の入り口でのマッチングを行う．スレッド A での sw 命令によりスレッド B または スレッド F の起動が行われる．スレッド C, D はスレッド B 内で発行された call 命令に対する結果の到着により起動される．また，スレッド E はスレッド C, D 内で発行された rlink に対する結果が両方そろった時点で起動される．AC でスレッド E がタスクディスパッチされるとき，既に 2 番目の引数が到着していれば，スレッド E に続いて スレッド F のディスパッチも行われる．

この例について，従来の Datarol マシンで実行した場合と，スレッドを導入した場合との比較を行う．いま，このプログラムが実行されている PE 内には他のインスタンス

が無く，かつ call,rlink 命令によるレイテンシを無視して考えると，従来の Datarol マシンでは図 8での最長パスより少なくとも 9 回循環パイプラインを回る必要があり，循環パイプラインの段数を 5 とすると，最短でも 45 clock 必要である．一方，スレッドを導入した場合最短パスは 5 スレッドであり，循環パイプライン 4 周分とスレッド内の命令実行分（ 13 命令）を合わせても $5*5+(13-5)=33$ clock である．さらに，スレッド実行の場合スレッド E, F は AC で連結される場合が高いので，実質的には循環パイプライン 4 周となり $5*4+(13-4)=29$ clock であり，従来のものと比べかなり少ないクロック数で実行できる．また，join の数も従来のものが 4 あるのに対し，スレッドを導入した場合は 2 に減少し，オペランドマッチングのオーバーヘッドも削減できる．

6 結論

循環パイプライン型の並列計算機において，プログラム中の並列度が，プロセッサエレメントの台数と循環パイプラインの段数の積より少ない場合，循環パイプライン中に空きができ，処理の効率が低下する．本稿では Datarol マシンにスレッド実行の概念を導入することにより，並列度の低い逐次的な処理に対する効率の改善を行った．

Datarol グラフは簡単な操作によりスレッド化できる．このスレッドについてスレッド間の起動制御部分とスレッドの本体部分とに分け，それぞれ専用の機構を用い処理を行う．

スレッドの本体部分を実行する FU では PC を導入し，RISC 型の実行機構によりスレッド内の高速処理を可能にした．しかし FU 内にレジスタファイルの内容をスレッド毎に読み込む必要が生じ，コンテクストスイッチングのコストが増大するが，スレッド長をある程度以上に大きくすることによりこのオーバーヘッドは隠蔽できる．

スレッド間の起動制御を行う AC 内にも PC を導入し多様なスレッド間起動制御を可能にし，FU 内のコンテクストスイッチングの回数を減少させる．また，AC 内はシンプルなハードウェア機構により高速な処理が行われ細粒度のスレッドにも対応できる．

以上のことから，Datarol にスレッドを導入することにより PE の処理効率を高めることができた．特に，PE 内のアクティブなインスタンスが少ないときに対して処理効率の低下が少ない．このことは，同じ問題に対してより多くの PE を使って高速に計算できることを示す．

今後，ハードウェアの詳細な設計を行い，それに基づきより多くのアプリケーションに対し，本方式による効率化についての評価を行う．

参考文献

[1] M.Amamiya and R.Taniguchi, "Datarol: A Massively Parallel Architecture for FunctionalLanguage", Proc. SPDP, pp.726-735, (1990)

[2] 薗田，上田，谷口，雨宮, "Datarol プロセッサの最適化設計と負荷制御方式"，情報処理学会「並列処理シンポジウム JSPP '90」, pp.121-128, (1990)

[3] 星出，園田，谷口，雨宮, "並列計算機 Datarol マシンにおける資源管理と負荷制御方式", 信学技法, CPSY91-7, pp.25-32, (1991)

[4] 児玉，坂井，山口, "データ駆動型シングルチッププロセッサ EMC-R の動作原理と実装"，情報処理学会論文誌，32, 7, pp.894-858, (1991)

[5] W.Dally, et. al,"The J-Machine: A Fine-grain Concurrent Computer", Proc. 11th IFIP, pp.1147-1153, (1989)

[6] R.S.Nikhil, at. al, "*T: A Multithread Massively Parallel Architecture", Proc. 19th ISCA, pp.156-167, (1992)