計算機アーキテクチャ 113−26
（1995. 8. 25）

RWC-1 のマルチスレッド処理機構

岡本 一晃 松岡 浩司 廣野 英雄 横田 隆史 坂井 修一

技術研究組合 新情報処理開発機構 つくば研究センタ

我々は、スレッド制御を自然に行える実行モデルとして、コンティニュエーション駆動実行モデルを考案し、それに基づいてマルチスレッド処理を最適化する並列処理アーキテクチャ RICA (Reduced Interprocessor-Communication Architecture) を提案している。現在開発を進めている RWC-1 のプロセッサは、RICA に基づくマルチスレッド処理機構を有しており、通信や同期のオーバヘッドを削減することで大域的な並列処理性能の向上を図っている。本稿では RWC-1 プロセッサのマルチスレッド処理機構について述べ、RWC-1 におけるスレッドレベル並列処理の基本動作を示す。

Thread Execution Mechanisms on a Massively Parallel Computer RWC-1

Kazuaki OKAMOTO Hiroshi MATSUOKA Hideo HIRONO Takashi YOKOTA
Shuichi SAKAI

Tsukuba Research Center, Real World Computing Partnership
Tsukuba Mitsui Building 16F, 1-6-1 Takezono,
Tsukuba-shi, Ibaraki 305 Japan

We have propsed "Continuation Driven Execution Model" which exploits multithreadings naturally and efficiently. Based on the model, RICA(Reduced Interprocessor-Communication Architecture) was proposed for optimizing thread-level parallelism. Massively parallel computer RWC-1 which based on the RICA is now under construction. RWC-1 processor has a thread control mechanisms especially for decreasing communication and synchronization overheads, and optimizing parallel execution among threads. This paper describes the thread execution of the RWC-1 processor, and also presents how thread-level parallel operations are optimized.

1 はじめに

我々は新情報処理開発機構(RWCP)において、中長期的な展望に立った汎用超並列計算機の研究・開発を行っており、その第一段階として、要素プロセッサ数1000規模の超並列計算機RWC-1の開発を進めている[1]。

一般に超並列計算機においては、問題に内在する並列性を最大限に抽出し、それぞれを計算機の演算処理ノードの物理的な並列性に写像することが重要である。そして処理の効率化を図るためには、それぞれの実行単位(アクティビティ)の各演算処理ノードへのマッピングやスケジューリングを最適化することが必要になる。複数の演算処理ノードの間でアクティビティの制御を効率良く行うためには、1) 高速でオーバヘッドの低い通信が行えること、2) アクティビティ間の同期が高速に行えること、3) アクティビティのスケジューリングが低いオーバヘッドで行えること、4) 計算機資源の物理的局所性が利用できること、などが必須である。さらに、アクティビティの粒度を最適化することも重要であると考えられる。

我々は、超並列計算機におけるアクティビティの基本単位としてスレッドを考え、マルチスレッド処理の最適化を図る独自のプロセッサアーキテクチャとして、RICA(Reduced Interprocessor-Communication Architecture)を提案している[2]。さらに負荷分散や並列性の制御などを効率的に支援する手段として、Super-Threadingの提唱を行っている[3]。我々が開発を進めている超並列計算機RWC-1にはこれらのアーキテクチャや手法を実装し、その有効性を検証することを目的としている。

本稿では、RICAに基づいたRWC-1用要素プロセッサにおけるマルチスレッド処理機構について述べる。

2 RWC-1のスレッド実行モデル

マルチスレッド計算機においてスレッド制御の最適化を実現するには、それを容易化するようなアーキテクチャの支援が必須である。ここで言うスレッドとは、一つのプロセッサの中で連続して処理される一連の命令列であり、通常スケジューリングの単位として扱われるものである。マルチスレッド処理を最適化するような並列アーキテクチャを構築するためには、まずスレッドの制御を自然な形で実現できる実行モデルを考えることが重要であると考えられる。これに対し我々は、コンティニュエーションが自動的にスレッドを起動するモデル「コンティニュエーション駆動実行モデル」を考え[1]、これに基くマルチスレッド計算機用のプロセッサアーキテクチャとしてRICAをすでに提案している。

コンティニュエーションとは、スレッドが実行される位置を示す指標である。具体的にはコンティニュエーションは、スレッドが実行されるプロセッサ位置(番号)、命令領域およびデータ領域のそれぞれの先頭位置へのポインタの組み合わせで表される。すなわちコンティニュエーションは、スレッドの実行開始位置を示している。コンティニュエーション駆動実行モデルにおいては、コンティニュエーションと引数データとの組み合わせ(activator)がメッセージの形でプロセッサ内もしくはプロセッサ間を輸送され、このメッセージの到着がスレッド起動のトリガとなる。したがって、本実行モデルにおいてマルチスレッド処理を効率化するためには、高速のプロセッサ間通信と効率良いメッセージ処理の実現が不可欠である。

我々が開発を進めている超並列計算機RWC-1は、コンティニュエーション駆動実行モデルに基づいたマルチスレッド計算機であり、マルチスレッド処理の最適化を実現する並列アーキテクチャとして提案したRICAの有効性を検証するものである。

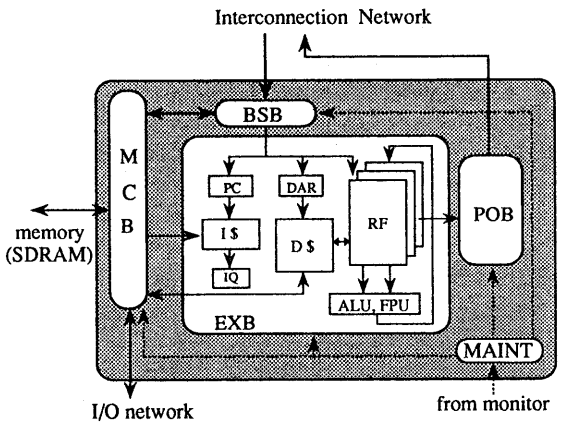
3 RWC-1のマルチスレッド処理機構

3.1 RWC-1プロセッサの内部構成

RWC-1における1つの処理ノードは、1) 演算処理を行う要素プロセッサ、2) 他ノードとの通信メッセージのルーティングを司るスイッチングユニット、および3) 外付けメモリから成る。このうち要素プロセッサは図1に示す通り、5つの機能ブロックで構成される[4]。

- BSB (Buffering and Scheduling Block)
  パケットの到着によるスレッドの起動を制御するブロック。
- EXB (EXecution Block)
  演算実行を行うブロック。64ビットRISCアーキテクチャをとる。
- MCB (Memory Control Block)
  外部メモリとの入出力を司るブロック。命令キャッシュやデータキャッシュの入れ換えの他、パケットの退避や外部I/Oからのメモリアクセスなどを制御する。
- POB (Packet Output Block)
  パケットの生成、およびその出力制御を行うブロック。
- MAINT (MAINTenance block)

チップの保守を行うブロック。実動作とは直接関係ないが、チップの動作を独立にモニタしてテストやデバッグを支援し、さらに必要に応じてボトルネック検出や効率測定などを行う。



PC : Program Counter
DAR: Data Address Register
I$: Instruction Cache
IQ : Instruction Queue
D$: Data Cache
RF : Register File

図 1: RWC-1 プロセッサの機能ブロック

3.2 パイプライン構成

RWC-1 プロセッサは、RISC アーキテクチャに基づく演算処理パイプラインと、マルチスレッド処理を実現するための通信パイプラインとが融合されて、実現されている (図 2)。

このうち演算処理パイプラインは、1) 命令フェッチ、2) 命令デコードおよびオペランド読み出し、3) 演算実行、4) レジスタ書き込み、の 4 ステージからなる。スレッドの実行中は、これらのパイプラインが一般的なスーパスカラ型 RISC プロセッサとして動作し、演算処理を行う。また、ロード・ストア、パケット生成・送出、I/O 入出力、浮動小数点演算など実行に複数クロックを要するものについては処理パイプラインを別置し、パイプラインの起動を 1 クロックで行って、特に依存関係がない限り並列に動作することを可能にしている。

一方、マルチスレッド処理のための通信パイプラインは、1) パケット処理 (スレッド起動)、2) 同期処理、3) スレッド実行、4) パケット生成および出力、からなる循環パイプラインを基本としている。RWC-1 ではプロセッサ間の通信はパケット交換により行われ、それぞれのパケットは、行き先 PE 番号、命令アドレス、データアドレス、などから成るコンティニュエーションと、引数となるデータ群との組み合わせで構成される。さらに付加情報として、パケットの先頭には実行優先度が搭載される (図 3)。各々のプロセッサでは、パケットの到着によってスレッドが起動され、演算処理が開始される。また、必要に応じて他プロセッサへパケットを送出することで、新規のスレッドを fork する。

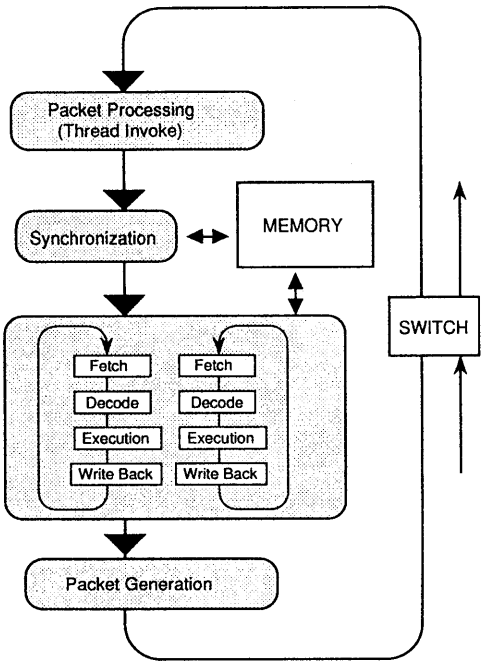


図 2: プロセッサのパイプライン構成

3.3 スレッドの起動と文脈切り替え

RWC-1 プロセッサでは、スレッドはパケットの到着により自動的に起動される。相互結合網を介して転送されて来るパケットは、まず入力パケットキューに取り込まれる。ひとたび入力パケットキューに蓄えられたパケットは、RICA のメカニズムに基づいて 1 パケットずつキューから抜き出され、引数データは演算実行部のレジスタファイルに自動的に注入される。それと同時に、パケット上のコンティニュエーションに従ってスレッドが起動される。この際、レジスタファイル

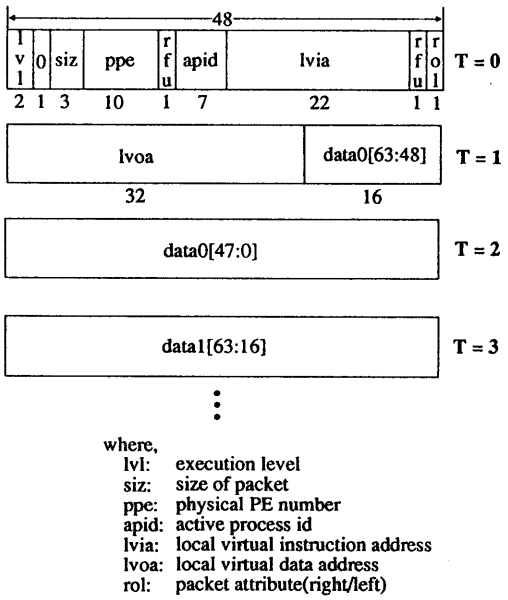


図 3: RWC-1 のパケット形式

はスコアボーディング方式で各レジスタデータごとに細かな同期をとりつつ管理される。

通常、一つのスレッドが実行されている時は、OS による介入がない限りスレッドの切り替えは発生せず、後続のパケットは入力キュー上で待ち行列を作る。しかし、緊急度の高い処理が、前のスレッドの実行によって待たされるのは好ましくないことが多く、こうした状況を回避するために RWC-1 ではパケットに 4 レベル (system mode 2 レベル + user mode 2 レベル) の優先度を用意している。

入力パケットキューは優先度ごとに管理されていて、実行中のスレッドより高い優先度を持つパケットが入力されると、実行中のスレッドがプリエンプトされ、文脈の切り替えが発生する。RWC-1 では、パケットによってプリエンプトされるのは user モードのスレッドのみであり、system モード間の優先度によるプリエンプションは起こさない。これを反映して RWC-1 では 3 セットのレジスタファイルを用意しており、それぞれを user low、user high、system(high or low) 用に割り当てている。これにより汎用レジスタの内容を退避することなくプリエンプトすることができる。

一方すべての優先度のスレッドは、割り込みの受け付けやトラップ命令の実行などによりプリエンプトされ、ハンドラのスレッドに切り替わる。この場合、user モードのスレッドがプリエンプトされる時には、上記と同様にレジスタの切り替えが起こるが、system モードのスレッドがプリエンプトされる時は、ハンドラの責任においてレジスタの退避が行われなければならない。

パケットによりプリエンプトされたスレッドは、上位の優先度のスレッドが終了した時に自動的に元の状態に復帰する。ここでは元のレジスタセットの内容がそのまま保持されているので、レジスタの復帰は必要ない。これに対してトラップや割り込みでプリエンプトされたスレッドは、ハンドラが RTE 命令を実行することで復帰する。この時、退避されていたレジスタの内容は、ハンドラの責任において復帰されなければならない。

3.4 同期処理

マルチスレッド処理においては、スレッド間の同期を高速に取ることが重要になり、ハードウェアによる支援がその効率化に大きく貢献すると考えられる。しかし、一般的には求められる同期の形態は処理する問題に依存し、複雑な形態の同期にまでハードウェアによる支援を要求するのは実装上困難である。そこで我々は、全ての同期における基本の形態として、2 つのメッセージ間の同期を取るマイクロ同期を考え、RWC-1 上に高速なマイクロ同期機構をハードウェアで実現した。そして、問題に依存するさまざまな形態の同期に関しては、すべてマイクロ同期をソフト的に組み合わせることで対応していく。

なお RWC-1 のマイクロ同期機構はハードウェアで実現されているが、ページフォルト時などの処理の簡単化のために、その起動に関しての自動化は行わない。実際に同期を行う場合には、bsync 命令の実行により陽にマイクロ同期機構を起動する。

bsync 命令は、先着したパケットを一度メモリ上に退避し、後続のパケットが到着したら退避したパケットを再びレジスタ上に戻して同期を成立させる命令である。データフロー計算機における direct matching 機構を命令で実行するものである (図 4)。

3.5 スレッドの実行と終了

スレッド内の局所実行は、スーパスカラアーキテクチャに基づいて行われる。一般的なプロセッサと同様に、整数演算命令、論理演算命令などを備え、また 64 ビットの浮動小数点演算命令も実装する。それぞれの演算処理部 (パケット生成を含む) は独立のパイプラインを持っており、各命令におけるパイプライン起動時間は 1 クロックであ

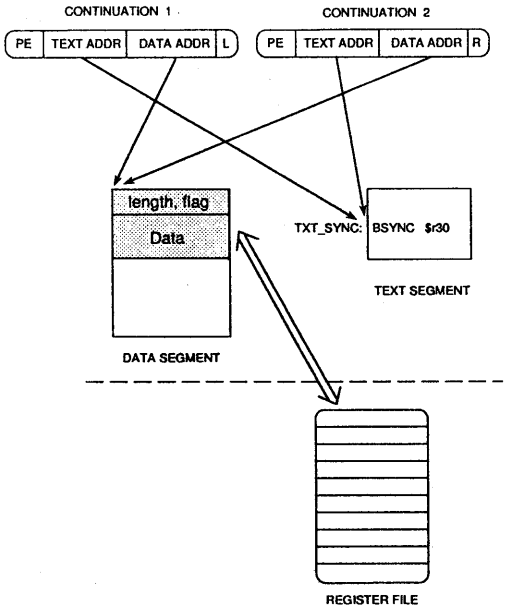


図 4: bsync 命令

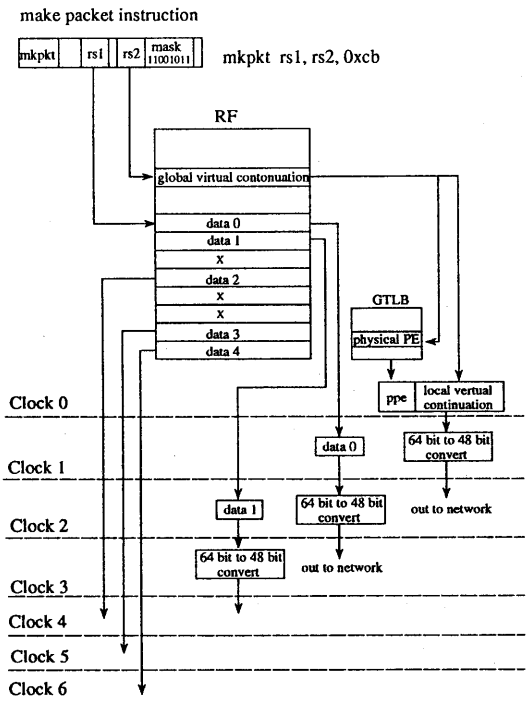


図 5: mkpkt 命令と動作

る。これらは特にデータ依存関係がない限り、並行して動作することが可能である。

スレッドの実行中、必要に応じて後述のパケット生成パイプラインを起動し、パケットを送出することができる。これにより他のプロセッサにスレッドを展開することが可能になる。また、分散共有メモリモデルにおける遠隔データのアクセスもこの枠組で行う。

実行中のスレッドは、break 命令 (例外ハンドラの場合は rte 命令) を実行することにより終了する。スレッドの実行が終了したプロセッサは、アイドル状態になってパケット待ちの状態になる。そして新たなパケットの入力により、次のスレッドが起動される。

さらに、必要に応じて break 時に trap を発生させ、OS に制御を移行することも可能である。

3.6 メッセージ生成

RICA では新たなメッセージの生成・送出のために独立したメッセージ生成パイプラインを想定しており、メッセージ生成・送出が他命令の実行と重畳化される。RWC-1 では、メッセージはパケットによって実現され、パケットはコンティニュエーションと引数データにより構成される。したがって RWC-1 のメッセージ生成は、1) コンティニュエーションの生成、2) パケット形成、の 2 ステップから成る [5]。

コンティニュエーションの生成は mkcnt 命令によって実行される。RWC-1 のコンティニュエーションは、論理プロセッサ番号、論理命令アドレス、論理データアドレスの組み合わせにより構成される。mkcnt 命令で生成されたコンティニュエーションは、一度汎用レジスタに格納される。そして次に述べる mkpkt 命令によって読み出され、パケットに埋め込まれる。

パケットの形成は mkpkt 命令によって行われる。パケットはプロセッサ間通信の手段であり、上述のコンティニュエーションと、引数データとして 1 ～ 8 ワードの任意の汎用レジスタの内容を組み合わせて形成される。この際、コンティニュエーション中の論理プロセッサ番号を GTLB(Global TLB) で変換して物理 PE 番号を得る。RWC-1 ではパケットのルーティングを容易化するため、パケット上には物理 PE 番号を搭載して配送する。したがって mkpkt 命令で起動されるパケット生成・送出パイプラインでは、以下の処理が行われる (図 5)。

1) コンティニュエーションからGTLBを用いて送り先の物理PE番号を獲得し、ヘッダ部を生成する。
2) register fileから転送データを読み出し、データ部とする。
3) 64bit → 48bit 変換をする。

一方、新たに起動しようとするスレッドが自プロセッサに閉じたローカルなものである時、メッセージを作らないでそのまま新しいスレッドに実行を移行することにより、局所性を利用することができる。これを実現するために、RWC-1ではコンティニュエーションを用いたジャンプ命令を用意している。これによりパケットを作らないで次のスレッドに移行することができ、パケット生成、送出、配送、およびスレッド起動の処理を削減できるだけでなく、レジスタやキャッシュの局所性を失わずに処理を続けることができる。これは、ソフトウェアの見地からは、スレッドの粒度の制御に寄与するものでもある。

4 プログラミングと評価

4.1 並列処理プリミティブ

ここでは、スレッドレベルの並列プログラミングを行う上での、並列処理プリミティブについて考える。

- fork
  あるプロセッサがスレッドを実行中に、別のプロセッサ上に新たなスレッドを起動させる時は、そのプロセッサに対してforkメッセージを送信する。forkメッセージに必要なものは、命令アドレスおよびデータアドレスからなるコンティニュエーションと、引数データであるから、RWC-1では、これは通常のパケットに相当する。したがってRWC-1で新規のスレッドをforkするためには、必要なコンティニュエーションをmkcnt命令で作成し、引数データとともにmkpkt命令でパケット化して送信する。

- join
  2つのメッセージの同期をとる。RWC-1では、bsync命令を実行することにより実現される。bsync命令はハードウェアによって支援される、マイクロ同期のための複合命令である。

- call
  あるプロセッサでスレッドを実行中、別のプロセッサにあらかじめ割り当てられている関数をcallする場合、前述のforkの枠組とは別に、関数の値の戻り位置が必要になる。したがって、callerは関数起動のコンティニュエーションと同時に、値の戻り位置のコンティニュエーションも作成しなければならない。関数起動のコンティニュエーションがcall時のパケット形成に使用されるのに対し、戻り位置のコンティニュエーションはcall時には単に引数データとして引き渡される。そしてreturn時のパケット形成に使用される。またRWC-1では、関数起動時に必要なデータ領域(フレーム)の管理はすべてランタイムソフトウェアに任せていて、ハードウェアによる支援は行わない。

- return
  関数の戻り位置は、上述の通りそのコンティニュエーションが引数データとしてあらかじめ渡されており、関数起動時に汎用レジスタ上に注入されている。return時にはこのコンティニュエーションを用いてパケット形成を行い、caller側に値を返す。

- リモートメモリアクセス
  分散共有メモリモデルにおいて、他プロセッサに属するメモリにアクセスする場合は、プロセッサ間の通信が必要である。この時の通信形態としてはいろいろなものが考えられるが、RWC-1ではページフォルト時の処理の簡単化を図るために、リモートメモリへのアクセスをそのプロセッサ上にメモリアクセスのスレッドを起動することで実現する。したがってリモートメモリへのアクセスは、上述の関数callと同様の枠組で実現できる。この際、メモリ側のプロセッサが別のスレッドを実行していて、メモリアクセスがスレッドの終了まで待たされるような場合に大幅に効率を落すことがある。そこでこれを避けるため、メモリアクセスのスレッドの優先度を上げ、プリエンプションを起こさせてメモリアクセスを実行する方法をとる。

4.2 プログラム例

ここで、いくつかのサンプルプログラムを考え、RWC-1のマルチスレッド処理がどのように行われるかを示す。

まず、リモートメモリ上のデータ読み出しを考える。前述のとおり、RWC-1ではリモートメモリへのアクセスは、リモートプロセッサ上のメモリアクセススレッドを実行することで実現する。

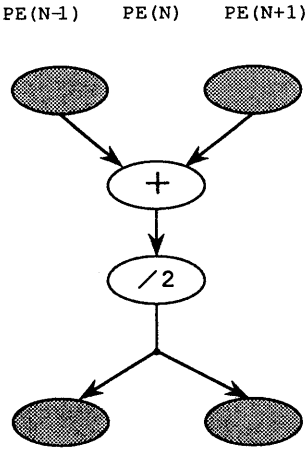
図6にリモートプロセッサ上のメモリ読み出しプログラムの例を示す。

```
rd_mem: ld      $r2, 0($r0)
        mkpkt   $r2, $r1, 1
        break
```

図6: メモリ読み出しプログラム

読み出し側は、このプロセッサに対してパケットを送信することでスレッドを起動し、メモリデータを得る。この時パケットのデータ部には引数として、リモートメモリの読み出しアドレスと、データの戻り位置(return continuation)を搭載する。

次に同期性能を評価するベンチマークとして知られる、PINGPONGのプログラム例を示す(図7)。これは隣接する2つのプロセッサから受けとったデータを平均し、さらに隣接する2つのプロセッサに返す、という処理の繰り返しである。ここでは送出先のコンティニュエーションは常に一定でありかつレジスタ上に常駐していることを想定していて、mkcnt命令は含めていない。




```
pingpong: bsync   $r30
          add     $r1, $r0, $r8
          sra     $r1, $r1, 1
          mkpkt   $r2, $r1, 1
          mkpkt   $r3, $r1, 1
          break
```

図7: PINGPONGのプログラム

さらに、関数callを用いたプログラム例として、フィボナッチ関数の並列化について考える。フィボナッチ関数は、

・$fib(n) = fib(n-1) + fib(n-2)$

で表される関数であり、再帰呼び出しを容易にマルチスレッド処理することができる。図8にRWC-1におけるフィボナッチ関数のプログラム例を示す。ただし、実際に再帰呼び出しを実現するためには、関数起動の際に割り当てるデータセグメントの管理が必要になる。

```
    fib: st      rtn_cont, $r0
         sub     $r5, $r1, 2
         ble     $r5, fib_1
         mkcnt   $r4, this_cntxt, fib_add, right
         mkcnt   $r6, new_cntxt1, fib, left
         mkpkt   $r4, $r6, 3
         sub     $r3, $r1, 1
         mkcnt   $r2, this_cntxt, fib_add, left
         mkcnt   $r6, new_cntxt2, fib, left
         mkpkt   $r2, $r6, 3
         break
fib_add: bsync   $r30
         add     $r1, $r0, $r8
         ld      $r0, rtn_cont
         bra     rtn
  fib_1: add     $r1, $r31, 1
    rtn: mkpkt   $r1, $r0, 1
         break
```

図8: フィボナッチ関数のプログラム例

4.3 評価

まず、前述のリモートメモリ読み出しにかかるコストを評価する。前節に示した、リモートプロセッサ上のスレッド実行には9clockかかる(キャッシュヒット時)。さらにスレッドの切り替え時には合計3clockのパイプラインバブルが生じるため、リモートメモリ読み出しの最大スループットは、12clockに1ワードということになる。これは50MHzで動作している時には、およそ33MB/secに相当する。一方、レイテンシに関してはプロセッサの物理位置に依存するため、一定ではない。しかも結合網上はパケットの非同期転送が行われるため、トポロジ上隣接するプロセッサ間であっても、実装上の物理位置によって到達時間が異なる。しかし、すべてのプロセッサは結合網のトポロジ上、8プロセッサごとに一つのループを形成しており(便宜上ここではグループと呼ぶ)、かつ実装もグループに関しては固定されているため、同一グループ内でのリモートメモリ読み出

しのレイテンシは一定である。RWC-1においては、グループ内では隣接するスイッチ間1ホップの転送が3clockで行われ、PEを経由する場合は8clock分余計にかかるので、先のリモートメモリ読み出しのためのパケット転送のレイテンシは40clockとなる見込みである。したがって、リモートメモリ読み出しにかかるレイテンシは、合計で52clockとなる予定であり、これは50MHzで動作している時は1.04μsecに相当する。

次に、PINGPONGプログラムによる同期性能を考察する。ここで隣接するプロセッサが絶え間なくパケットを送出しているとすると、パケットは入力パケットキューに溜るので見かけ上通信のレイテンシが無視できる。入力パケットキューからパケットを取り出し、スレッドが起動されるまでに要する時間は2clock、同期に失敗した時にスレッドの実行時間が12clock、同期に成功してパケットを出力する時のスレッドの実行時間が21clockである。同期に成功する回数と失敗する回数とは同じであるから、一回の同期に要する時間は35clock前後と思われる。50MHzで動作している時は、一回の同期がほぼ700nsecで取れることになる[1]。

5 おわりに

本稿では超並列計算機RWC-1のマルチスレッド処理機構について述べた。スレッド制御に適合した実行モデルであるコンティニュエーション駆動実行モデルと、それに基づくRWC-1プロセッサにおけるマルチスレッド処理機構がどのように実現されているかを述べ、さらにこのスレッド制御機構が、実際のプログラミングにどう作用するかを評価した。

RWC-1の要素プロセッサはすでに第一試作が完成しており、現在評価中である。また、ここでの検証結果を踏まえて改良版プロセッサの開発を本年度中に行い、1996年の1000台システムの稼働を目指す。今後はRWC-1の開発を通じ、本稿で述べた並列アーキテクチャの検証をより詳細に進めていく予定である。

謝辞

本研究を遂行するにあたり、有益な御指導、御討論をいただいた島田RWC研究所長、石川超並列ソフトウェア研究室長、超並列ソフトウェア研究室員の諸氏、ならびにRWC超並列アーキテクチャワーキンググループの諸氏に感謝いたします。

[1] これらはすべてキャッシュにヒットした時の値である。

参考文献

[1] S.Sakai, et.al., RWC-1 Massively Parallel Architecture, Proc. HPCC'94, pp.33-38 (1994).

[2] S.Sakai, et.al., Reduced interprocessor- communication architecture and its implementation on EM-4, Parallel Computing Vol.21, No.5, pp.753-769 (1995).

[3] S. Sakai, et.al., Super-Threading: Architectural and Software Mechanisms for Optimizing Parallel Computation, Proc. ICS'93, pp.251-260 (1993).

[4] 松岡他、超並列計算機RWC-1用プロセッサチップの設計、信学技報 CPSY95-18 (1995).

[5] 岡本他、超並列計算機RWC-1の命令セットアーキテクチャ、信学技報 CPSY95-36 (1995).