超並列計算機におけるマルチスレッド処理機構と基本性能

岡本一晃[†] 松岡浩司[†] 廣野英雄[†]
横田隆史[†] 坂井修一[††]

超並列計算機においては，遠隔メモリ操作や遠隔手続き呼び出しにともなうレイテンシが大きな問題となる．これを解決する手段として，マルチスレッド処理によるレイテンシの隠蔽があげられる．さまざまな粒度におけるマルチスレッド処理を効率良く実行するためには，効果的なハードウェアによる支援が期待できるようなスレッド処理機構を，プロセッサが備える必要がある．本稿では，効率良くマルチスレッド処理を実行するスレッド処理機構について考え，超並列計算機向けのプロセッサアーキテクチャを示す．そしてこれに基づいて現在開発中の，マルチスレッド型プロセッサ RICA-1 を紹介し，その基本性能を示す．また，遠隔メモリ操作や遠隔手続き呼び出しなどが，マルチスレッド処理によるレイテンシ隠蔽によって，効率良く実現できることを示す．

Multithread Execution Mechanisms on a Massively Parallel Computer

KAZUAKI OKAMOTO,[†] HIROSHI MATSUOKA,[†] HIDEO HIRONO,[†]
TAKASHI YOKOTA[†] and SHUICHI SAKAI[††]

Latencies of remote memory access and remote procedure call are serious problems on a massively parallel computer. In order to improve the machine performance, it is quite effective to hide these latencies by multithreading. Thread execution mechanism which is effectively supported by the hardware is indispensable to realize efficient multithread execution. In this paper, we propose the processor architecture for massively parallel computers with efficient thread execution mechanism, and present RICA-1 multithreaded processor based on it. On the RICA-1, both remote memory access and remote procedure call are realized efficiently.

1. はじめに

近年の高度情報化社会においては，計算量の増大と複雑化がますます進み，高速計算に対する需要は増加するばかりである．こうした中，デバイス技術に起因するプロセッサ単体性能の限界が指摘されており，超並列処理の実現は必要不可欠なものと考えられている．

一般に超並列処理システムにおいては，問題に内在する並列性を最大限に抽出し，それぞれをシステムにおける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．ここで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．このとき，多数の演算処理ノードの間でのアクティビティの並列度をあげると，演算処理ノード間での通信が頻繁に発生する．こうした超並列処理システムにおいて，大きな問題のひとつとなるのが，演算処理ノード間の通信や同期，さらにメモリアクセスなどの際に生じるレイテンシである．これを解決する有効な手段としては，マルチスレッド処理によるレイテンシの隠蔽が考えられるが，効果的にレイテンシを隠蔽するマルチスレッド処理を実現するためには，演算処理ノードが高速低オーバヘッドのスレッド切り替え機能を持つ必要がある．

高速なスレッド切り替えをハードウェアが支援するマルチスレッド計算機として，データ駆動計算機の研究が行われており，すでに実装を通じてその有効性が実証されている[1)~3)]．しかしこれは，命令レベルでのマルチスレッド処理であるため，問題に内在する並列性を自然な形で引き出せるという利点を持つ一方で，逐次性の高い問題に対する処理の非効率性が問題視されている[4)]．一方，近年の RISC 型マイクロプロセッサにおいては，デバイス技術やパイプライン技術の向上により，単体による逐次処理性能の向上が著しい．

† 新情報処理開発機構
Real World Computing Partnership
†† 筑波大学
Universitiy of Tsukuba

そこで，これを利用することで高い逐次処理性能を追求したマルチスレッド計算機が提案されている[5),6)]．しかし，現在の汎用マイクロプロセッサでは，スレッドの切り替えにかかるオーバヘッドが小さくないため，逆に細粒度のマルチスレッド処理に対しては自由度が低く，十分な性能を発揮するシステムを構築するのは困難である．

汎用の超並列計算機上で効率良いマルチスレッド処理を実現するためには，さまざまな粒度におけるマルチスレッド処理を効果的にサポートするスレッド処理機構を持つことが必須である．そこで本稿では，超並列計算機のためのマルチスレッド処理機構について検討し，中でも特に重要と思われる，メッセージの授受や同期に関するオーバヘッドの軽減を考える．そしてそれらの処理機構を組み込んだ独自のプロセッサアーキテクチャを提案し，その実装の一形態として，新情報処理開発機構において現在開発中の，マルチスレッド型プロセッサ RICA-1 について紹介する．

2. マルチスレッディングによる超並列処理

超並列システムにおいて，遠隔手続き呼び出しや遠隔データの操作などの際に生じるレイテンシは深刻な問題である．我々は，超並列システムを設計するうえで，この問題を解決する基本方針として，マルチスレッド処理によってレイテンシの隠蔽を図ることにした．

超並列システム上で，効率良いマルチスレッド処理を実現するには，プロセッサアーキテクチャと，相互結合網アーキテクチャとの両方からのハードウェア支援が必要である．我々はこの両者について検討を行い，それぞれ独自のアーキテクチャを提案しているが，このうち相互結合網アーキテクチャに関しては別途報告しているので[7)]，ここではプロセッサアーキテクチャについてのみ考える．

効果的なマルチスレッド処理を実現するために，超並列向けプロセッサに求められる機能は，次に示すとおりである．

1) プロセッサ間通信のオーバヘッド軽減

メッセージによってプロセッサ間通信を行うシステムでは，メッセージ授受の際のオーバヘッドを軽減する必要がある．メッセージの処理オーバヘッドには，網インタフェース部のハードウェアオーバヘッドと，処理ソフトウェアのオーバヘッドとが考えられるが，両者ともに軽減されなければならない．

2) コンテクストスイッチのオーバヘッド軽減

コンテクストの切り替えに際しては，前の処理の環境を退避し，次の処理の環境を復帰する作業が必要になり，これが大きなオーバヘッドとなる．効果的なマルチスレッド処理のためには，切り替えのオーバヘッドを軽減する必要がある．

3) スレッド間の同期にかかるオーバヘッド軽減

超並列システム上でマルチスレッド処理を行う場合，複数のスレッドが並行して処理されるため，スレッド間の同期処理が頻出することが考えられる．このため，同期にかかるオーバヘッドを軽減することが重要である．

これらの機能をスレッド処理機構として実現するためには，演算処理ノードとしてのプロセッサが，継ぎ目のない単純化された通信および同期のインタフェースを持つ必要がある．これまでには，たとえばメッセージ処理をすべてソフトウェアで解決するような方法が提案されているが[8)]，これではスレッド起動にかかるオーバヘッドが大きくなり，細粒度のマルチスレッド処理を効率良く実行することができない．したがって，こうした通信や同期のインタフェースをハードウェアが支援する必要がある．その実現形態のひとつとして，筆者らは演算処理パイプラインと通信処理パイプラインを高度に融合し，単純化した新しいプロセッサアーキテクチャを提案し[9)]，これに基づく超並列計算機向け要素プロセッサ RICA-1 を開発中である．この RICA-1 は，筆者らが現在開発を進めている超並列計算機 RWC-1 の要素プロセッサとして，独自に開発する RISC 拡張型マルチスレッドプロセッサである．

一般に実装までを含めたアーキテクチャの研究においては，既存のプロセッサを利用して周辺だけを開発する方法と，プロセッサそのものをスクラッチから開発する方法とが考えられる．前者に比べ後者の方法では，全体の開発時間が大きくなること，開発のコストがかかること，さらには必要な労力そのものが大きいことなどから，一般にはなかなか採用されない場合が多い．しかしその一方で，提案する最適アーキテクチャをそのまま実装し，実験および評価ができるという，非常に大きなメリットも有している．

筆者らの研究プロジェクトにおいては，21 世紀に向けての情報処理インフラを考えるという立場から長期的展望に立ち，提案するプロセッサアーキテクチャに基づいたプロセッサ LSI を，スクラッチから設計，試作することにした．そして，これを独自の結合網によって多数結合することにより，汎用超並列計算システムを構築する[10)]．

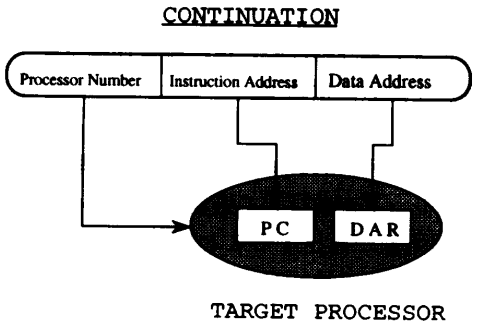


図1 コンティニュエーション
Fig. 1 Continuation.

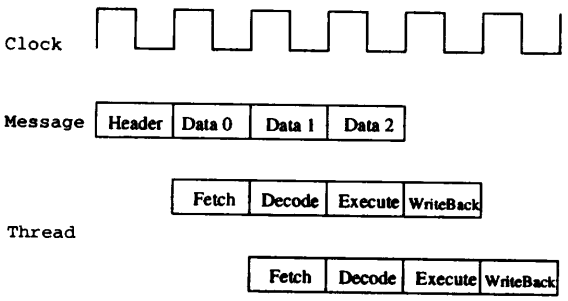


図2 メッセージ注入とスレッド実行の重畳化
Fig. 2 Concurrent processing of message injection and thread execution.

3. マルチスレッド処理機構

3.1 実行モデル

マルチスレッド処理を最適実行するような超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．これに対し筆者らは，メッセージによって運ばれる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」をすでに提案している[11]．ここでコンティニュエーションとは，スレッドが起動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，そのスレッドが起動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（図1）．

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が，メッセージの形でプロセッサ内もしくは複数の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，本実行モデルに基づくマルチスレッド計算機において処理を効率化するためには，メッセージ受信，スレッド起動，同期処理，メッセージ生成，メッセージ送信といった一連の処理がパイプライン化され，ハードウェアによって効率良く支援される必要がある．

3.2 処理機構

前述のコンティニュエーション駆動実行モデルに基づいて，効率良いマルチスレッド処理を実現するために必要な処理機構について考える．

● メッセージ処理機構

メッセージの処理オーバヘッドを軽減するためには，メッセージの到着と同時にデータがレジスタに注入され，かつスレッドが起動されるような機構が必要である．ここで，メッセージの輸送によって実現される通信は，処理の高速化を考えた場合プロセッサどうしのレジスタ間転送である必要がある．そこでは，受信したメッセージのコンティニュエーションが制御レジスタに，引数データは汎用レジスタに注入される．また，コンティニュエーションの中の命令番地がプロセッサのプログラムカウンタに書き込まれることで，スレッドの起動あるいは切り替えが起こる．

効果的なマルチスレッド処理の実現のためには，現在実行中のスレッドと独立に，次のスレッドを起動するメッセージの処理を実行することが重要である．したがって，到着したメッセージのレジスタへの注入は，ハードウェアにより自動的に行われるのが望ましい．さらには，メッセージの注入と新規スレッドの実行はパイプライン化されるべきである．これにより，引数データの汎用レジスタへの注入とスレッドの実行が，特にデータ依存関係のない限り並行して行われる（図2）．

スレッド切り替えの際には，現在実行中のスレッドの環境を保持しておかなければ，中断したスレッドが再開できない．一般的にはレジスタの内容をメモリ上に退避して保持することで対処するが，これでは退避および復帰にともなうオーバヘッドが大きくなってしまう．そこで，レジスタセットを複数用意し，新しいスレッドへの移行をレジスタセットの切り替えだけで行うようにする．さらにレジスタセットが溢れた場合でも，レジスタの退避および復帰の作業とスレッド実行とを重畳化して，そのオーバヘッドの軽減を図る．

● 同期処理機構

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と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．しかし並列処理において求められる同期の形態は，一般に処理する問題に依存し，多様である．たとえば，新たにforkしたスレッドからの戻り値を待つ場合（図3）や生産者-消費者間のデータ依存関係を保持するための同期，ベクトル演算のような定型的な繰返し演算に

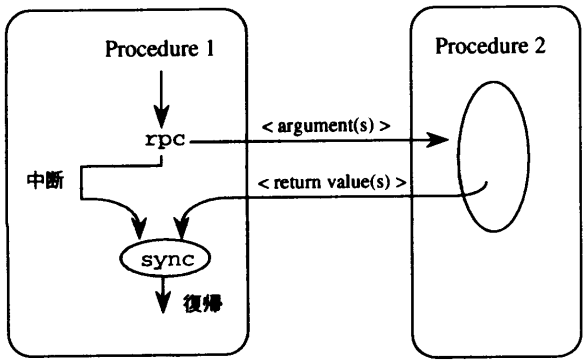


図3 RPCにおける同期処理
Fig. 3 Synchronization on RPC.

おけるベクタの要素どうしの対応付けなど，2つのスレッド間でとる同期もあれば，多数のスレッド間で処理の歩調を合わせるためにとる，バリア同期のようなものもある．こうしたさまざまな形態の同期すべてをハードウェアで支援するのは，実装上困難である．そこで，さまざまな形態の同期の効率化を考えるために，すべ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とるマイクロ同期を考え，高速なマイクロ同期機構のみをハードウェアで実現することを考える．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ウェア的に組み合わせることで柔軟に対応していくこととする．

同期もメッセージ処理同様，すべてをハードウェアにより自動的に処理されるのが効率面からは望ましい．しかし，メッセージ処理とは違い，同期処理にはメモリアクセスがともなうため，ページフォルトなどの例外が発生する場合が考えられ，すべてをハードウェアで対応するのは実装上困難である．そこで，同期処理に関しては，専用の命令を起動することにより，例外発生時の対応をソフトウェアでとれるよう実装する．

● メッセージ生成機構

効率良いメッセージの生成および送出のためには，ハードウェアの支援によるメッセージ生成・送出パイプラインが必須である．さらに独立したメッセージ生成・送出パイプラインの設置により，他の処理と並列に実行されることが重要である．メッセージ生成および送出は，ソフトウェアによって明示的に実行されるべきであり，専用命令により起動することとする．このとき，メッセージ生成パイプラインは，実行に複数クロックを要するので，パイプラインを別置し，起動のみを命令から1クロックで行うことにより，他命令の実行と重畳化されることが望ましい．また，メッセージの生成と送出は，通常一連の動作として実行されるので，送出時には，生成と送出を2命令に分割する必要はない．

具体的には，メッセージ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ので，メッセージの生成は，1) コンティニュエーションの生成，2) メッセージ形成，の2ステップから成る．しかし，1つのコンティニュエーションから複数のメッセージを形成する場合が少くないので，コンティニュエーションの生成は独立した命令によって実行され，生成されたコンティニュエーションはレジスタ上に保持されるのが望ましい．また，新たに起動するスレッドが戻り値を持つ場合の戻り場所もコンティニュエーションとして与えることができる．この場合は，コンティニュエーションが引数データとしてプロセッサ間を輸送されることが考えられるので，コンティニュエーションは制御レジスタ上ではなく，汎用レジスタ上に保持される方が柔軟性に富む．

また，処理のパイプラインを別置している場合，メッセージ送出とスレッド終了との非同期化も可能である．すなわち，メッセージ生成のパイプラインを起動すれば，メッセージが出終わっていなくても，スレッドの終了・切り替えができる．

メッセージ生成・送出パイプラインにおける処理の一例を図4に示す．メッセージはプロセッサ間通信の手段であり，前述のとおりコンティニュエーションと，引数データとの組み合わせにより生成される．通常，コンティニュエーションには論理プロセッサ番号が含まれているが，結合網上では物理プロセッサ番号を用いた方がルーティングが容易化されるため，プロセッサ番号の論理物理変換は，送出側で行わなければならない．ここではメッセージ生成の際，GTLB (Global TLB) を用いてプロセッサ番号の論理物理変換を行い，メッセージ上には物理プロセッサ番号を搭載して配送する．GTLBとは，論理プロセッサ番号から物理プロセッサ番号を導く変換テーブルの入ったバッファである．図4において，メッセージ生成命令はオペランドとして2つのソースレジスタをとり，それぞれはコンティニュエーション，引数データの格納先である．引数データは複数個存在することが考えられるので，先頭のレジスタ番号とする．この命令が実行されるとメッセージ生成・送出パイプラインが起動され，まずコンティニュエーション中の論理プロセッサ番号をGTLBにより物理プロセッサ番号に変換して，メッセージヘッダを生成，送出する．次に，引数データを順次汎用レジスタから読み出し，送出する．したがって本命令の実行には，メッセージの語数分だけのサイクルを要する．しかしパイプラインの起動は1クロッ

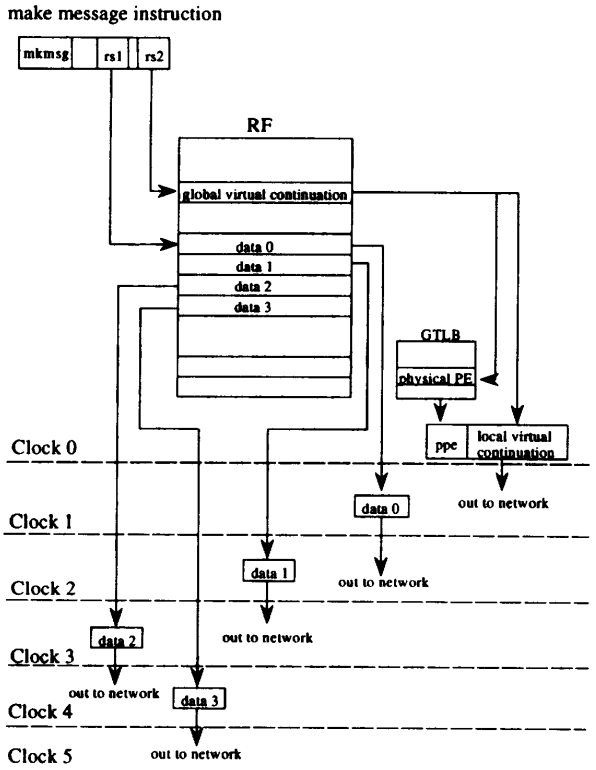


図4 メッセージ生成・送出パイプライン
Fig. 4 Message generation pipeline.

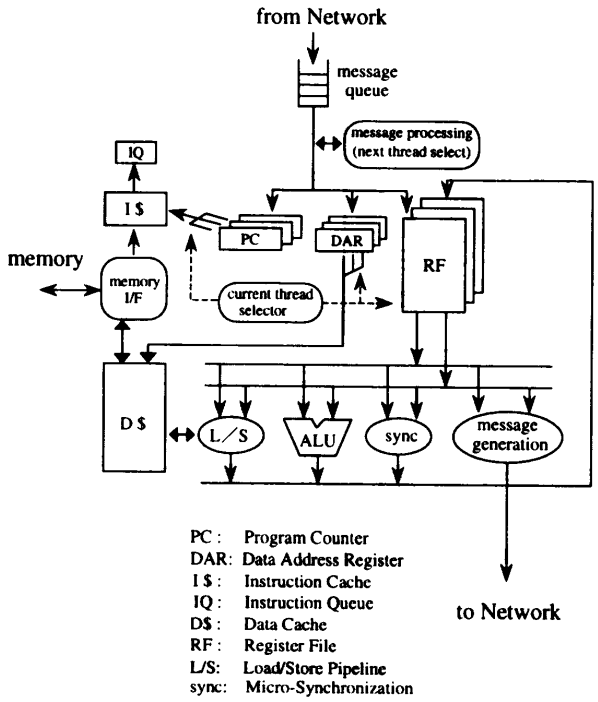


図5 マルチスレッド型プロセッサアーキテクチャ
Fig. 5 Multithreaded processor architecture.

クで行えるので，パイプラインを独立して設置することにより，汎用レジスタ上のデータに依存関係のない後続命令については，並行して実行することが可能である．データ依存関係については，たとえばレジスタ・スコアボーディングなどの技法により管理する．

3.3 プロセッサアーキテクチャ

前節までで，マルチスレッド処理に適した実行モデルとして，コンティニュエーション駆動実行モデルを示し，その実現に必要な処理機構について述べた．ここでは，それらの処理機構を実装し，コンティニュエーション駆動実行モデルに基づいて効率良くマルチスレッド処理を行うプロセッサアーキテクチャの全体像を示す．

図5 に示すとおり，プロセッサのコアにあたる演算処理部は，RISC アーキテクチャを採用することで，効率良い局所演算実行を実現する．ここでバスを多重化し，複数命令を同時発行してスーパスカラ化することで，さらに実行効率を向上させることができる．

一方，メッセージ処理に関しては専用のパイプラインを設置し，それが演算処理パイプラインと融合されている．メッセージ受信部ではメッセージの到着とともに自動的にレジスタへの注入が実行され，新規スレッドの起動を行う．一方，同期処理およびメッセージ生成・送出の処理部は，それぞれ演算処理系と並列につながっていて，命令実行により起動される．

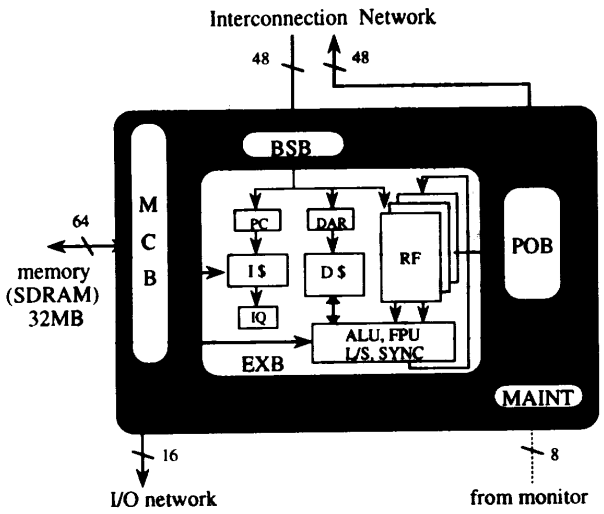
4. 実装と基本性能

前章で，効果的なマルチスレッド処理を実現するプロセッサアーキテクチャについて述べた．ここでは，その実装例として，我々が現在開発を進めている超並列計算機向けマルチスレッド型プロセッサ RICA-1 について紹介する．

4.1 RICA-1：マルチスレッド型プロセッサ

RICA-1 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のための通信パイプラインとが高度に融合された，マルチスレッド型プロセッサである．RICA-1 の基本構成図を**図6** に示す．

RICA-1 の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．RISC プロセッサとしての RICA-1 の諸元を**表1** に示す．また，ロード・ストア，メッセージ生成・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



BSB: Buffering & Scheduling Block
MCB: Memory Control Block
POB: Packet Output Block
MAINT: Maintenance Block
EXB: EXecution Block

PC: Program Counter
DAR: Data Address Register (26bits)
IQ: Instruction Queue
I$: Instruction Cache (4KB)
D$: Data Cache (8KB)
RF: Register File (64bits, 32words, 3sets)

(Internal Bus: Instruction 32bits, Data 64bits)

図6 RICA-1 の内部構成図
Fig. 6 Organization of RICA-1 processor.

表1 RICA-1 の演算処理部の諸元
Table 1 Execution unit of RICA-1.

命令処理：	
命令セット数	48
命令形式	3 オペランド
同時命令発行数	2
管理方式	スコアボーディング
汎用レジスタ：	
レジスタサイズ	64 bit × 32 word × 3 セット
キャッシュ：	
キャッシュサイズ	データ 8 kB 命令 4 kB
ラインサイズ	4 word
制御方式	2 ウェイセットアソシアティブ LRU 方式 (ページごとにライトスルー/コピーバック選択可)
その他：	
整数乗除算器	64 bit
浮動小数点演算器	64 bit
Load/Store	3 段

て，特にデータ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．さらに I/O ポートを別置し，大量の入出力によりプロセッサ間通信が妨げられることを防いでいる．

一方，マルチスレッド処理のための通信機構は，1) メッセージ処理（スレッド起動），2) 同期処理，3) スレッド実行，4) メッセージ生成および出力，からなる循環パイプラインを基本としている．このうち，メッセージ処理とスレッドの実行は，メッセージの到着によってハードウェアが自動的に起動する．これに対し，同期処理とメッセージ生成および送出は，命令実行により起動される．

RICA-1 のパイプライン構成を図7に，ハードウェア諸元を表2に示す．

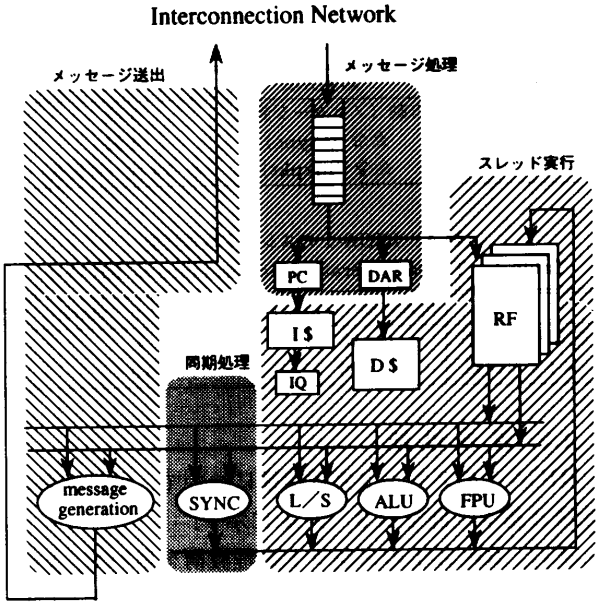


図7 RICA-1 のパイプライン構成
Fig. 7 RICA-1 pipeline organization.

表2 RICA-1 のハードウェア諸元
Table 2 Hardware profile of RICA-1.

LSI 実装：	
テクノロジ	0.5 μm CMOS Standard Cell 3 metal layers
パッケージ	527 pin Ceramic PGA
チップサイズ	20.0 mm × 20.0 mm
ゲート数	Random logic 200 k gates Internal RAM 13 k bytes
通信ポート：	
入力ポート	48 bits × 1 port
出力ポート	48 bits × 1 port
転送レート	300 MB/sec/port
I/O ポート：	
入力ポート	16 bits × 1 port
出力ポート	16 bits × 1 port
転送レート	50 MB/sec/port
動作周波数	50 MHz
各パイプラインの段数：	
整数乗除算器	8 段
浮動小数点演算器	6 段
Load/Store	3 段
メッセージ生成	4~14 段（可変長）

表3 RICA-1のスレッド処理機構の仕様
Table 3 Thread execuiton pipeline of RICA-1.

処理機構	起動方法	パイプライン長
メッセージ受信	自動（メッセージ到着）	3段
マイクロ同期	命令（bsync）	4段
メッセージ送信	命令（mkpkt）	4～14段（可変長）

表4 RICA-1のスレッド処理性能
Table 4 Thread operation performance of RICA-1.

	処理時間（clock）	オーバヘッド（clock）
遠隔メモリ操作	18	4
マイクロ同期	20	10

4.2 基本性能

RICA-1のスレッド処理機構について，それぞれの仕様と基本性能を表3，表4に示す．

次に，RICA-1のスレッド処理機構を用いて，いくつかの処理を行った場合の処理時間を評価する．まず，リモートメモリ参照の場合を考える．

RICA-1において，リモートメモリを参照する場合，リモートプロセッサにメッセージを送信し，リモートプロセッサ上にメモリ参照のスレッドを起動する方法と，I/O転送路を介してDMA転送を行う方法とがある．後者はページ単位の大量データのコピーに用いられるが，ここでは細粒度マルチスレッド処理により関係の深い前者について評価する（図8）．

リモートメモリを参照したい場合，まずcaller側のプロセッサではメッセージを生成・送信して，現スレッドを中断する．そして，別のスレッドに実行を切り替える．リモート側からメッセージが返ってきたら，元のスレッドが復帰され，処理が続行される．ここで，リモートメモリ参照に要する時間は，メッセージの生成・送出に4clock，callee側での処理（メッセージ受信，メモリ参照，メッセージ生成・送出）に9clock，さらにメッセージの受信・汎用レジスタへの注入に5clock，の計18clockにネットワークの転送時間を加えたものとなり，これらはそのままレイテンシとなって，避けることができない．しかし図8に示すマルチスレッド処理を採用すると，メッセージ生成パイプラインの起動（mkpkt命令の実行）に1clock，thread Aからthread Bへの切り替え（break命令の実行）に1clockの，計2clockの処理時間後はthread Bが実行される．その後callee側から戻ってくるメッセージは，thread Bの実行と並行して受信され，レジスタに注入される．このときthread Bからthread Aへの復帰は，ハードによって自動的に行われるので，ここでオーバヘッドは生じない．したがって，このマルチスレッド処理によるリモートメモリ参照では，最初の2clock以外のレイテンシをthread Bの実行により隠蔽することができ，実行効率を落とさない．このようなマルチスレッド処理は，たとえば遠隔データのプリフェッチの際に有効である．遠隔データのプリフェッチを行うプログラムを，レジスタ転送レベルのシミュレータ上で実行した結果を図9に示す．ここで，thread Aは遠隔データのフェッチを行うスレッド，thread Bはその遠隔データとは直接は関係ないローカルな処理である．図において，thread Aからthread Bへの切り替えはbreak命令で陽に記述しているため，切り替え時の命令発行にフェッチ，デコード，の2段分の隙間ができるのに対し，thread Bからthread Aへの切り替えはメッセージの到着により自動的に起こるので，隙間なく命令列を発行する．

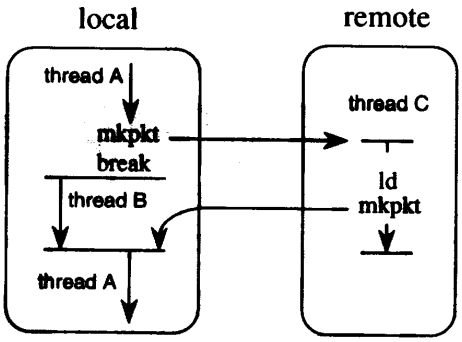


図8 リモートメモリ参照
Fig. 8 Remote memory access.

次に同期処理性能について考える．ここでは，すべての同期形態の基本として，ハードウェアが支援するマイクロ同期機構を評価する．RICA-1のマイクロ同期機構は，専用のbsync命令によって起動される．bsync命令は，先に受信したメッセージをメモリ上に退避し，同期の対になるメッセージを受信したときに，退避した先着メッセージを汎用レジスタ上に復帰して，後続の命令実行を続行する．この際，先着メッセージを受信・処理してから後続メッセージを受信するまでの間は，他のスレッドに実行を切り替えることにより，実行効率を落とさない．マイクロ同期の処理時間をレジスタ転送レベルシミュレータ上で評価すると（図10），先着メッセージを受信してからbsync命令が起動されるまでに4clock，bsync命令が実行され先着メッセージがキャッシュ上に退避される時間が5clock，同じく後続メッセージの受信処理が4clock，bsync命令が実行され先着メッセージが復帰されるまでの時間が8clockである．したがってこのマイクロ同期機構により，1回の同期に要する処理時間は20clockになり，これは，プロセッサが50MHzで動作していると

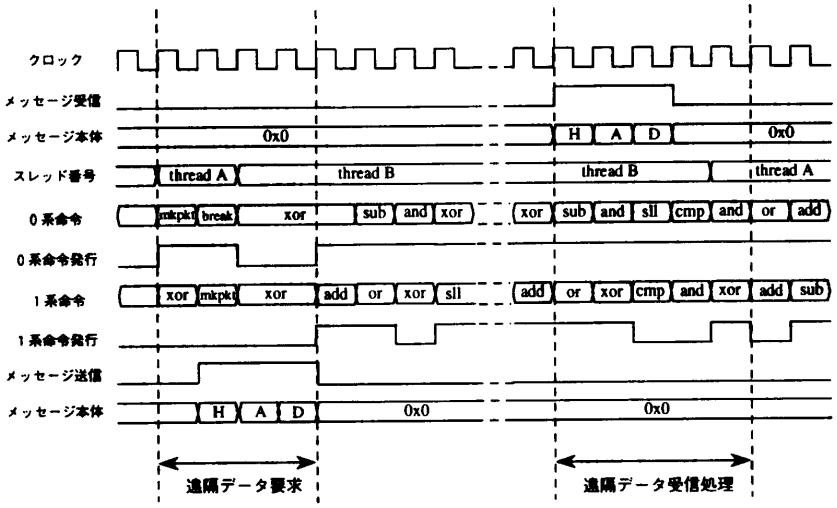


図9 リモートメモリ参照の評価
Fig. 9 Evaluation of remote memory access.

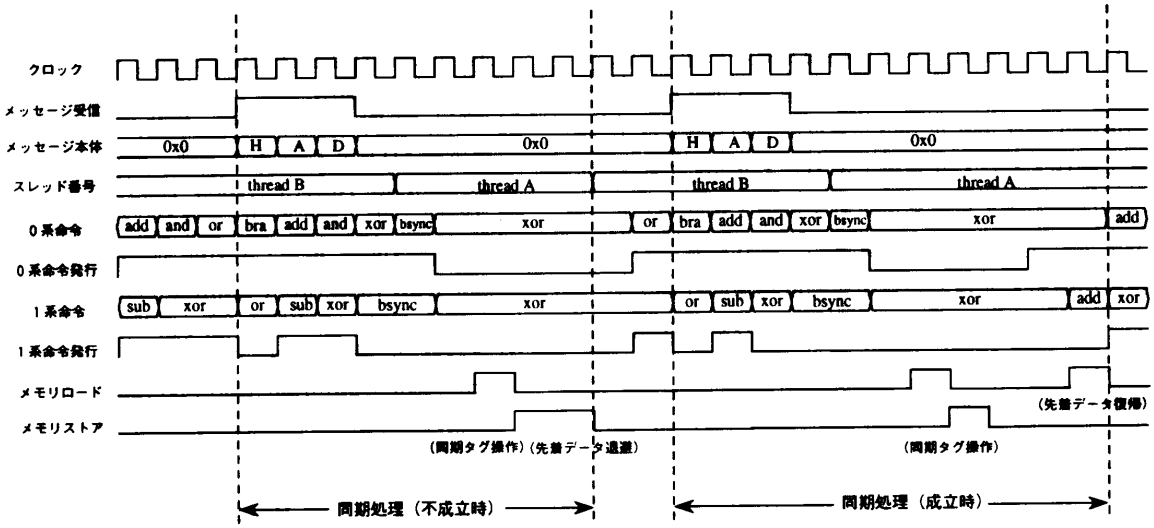


図10 マイクロ同期処理の評価
Fig. 10 Evaluation of micro-synchronization.

き，1回の同期が400 nsecで行えることを示している．また，メッセージ受信処理と，先着メッセージの復帰処理の一部とは，他スレッドの実行と重畳化できる．図10の例では，同期処理20 clockのうち10 clockは，他のスレッドの命令実行とオーバラップしているのが分かる．

さらに，遠隔プロセッサへの手続き呼び出しについて考える．新たなスレッドをforkする際に必要な処理は，戻り値がない場合，caller側では新規スレッドのコンティニュエーションを生成し，引数とともにメッセージに載せて送出することである．一方，callee側では，メッセージを受け取った後，スレッドを起動するための新たなフレームを獲得しなければならない．さらにスレッドが戻り値を持つ場合，caller側では戻り位置（return continuation）の生成を行い，callee側ではそれを用いて戻り値を載せたメッセージを生成し，送出する処理が加わる．プロセッサが50 MHzで動作しているときの，遠隔手続き呼び出しにかかる処理時間を，**表5**に示す．また，これらの処理時間のうち，マルチスレッド処理により隠蔽できないものを，オーバヘッドとして示す．

以上のように，複数のプロセッサにまたがるような処理においても，レイテンシを隠蔽し数クロックのオーバヘッドでこれが実現できることが示された．これは，RISC拡張型のプロセッサRICA-1が，細粒度のマルチスレッド処理についても効率良く実行できることを示している．

表5 遠隔手続き呼び出し処理
Table 5 Remote procedure call.

	caller 側		callee 側	
	処理時間 (μsec)	オーバヘッド (clock)	処理時間 (μsec)	オーバヘッド (clock)
remote fork				
(1 args)	0.16	5	0.22	7
(7 args)	0.36	5	0.30	7
remote fork with return				
(1 args)	0.46	19	0.26	8
(7 args)	0.60	19	0.32	9

5. おわりに

本稿では，超並列計算機における遠隔メモリ操作などのレイテンシを，マルチスレッド処理により隠蔽することを目的に，効果的なマルチスレッド処理を実現するスレッド処理機構を検討した．スレッドの起動や切り替えを高速に行うには，通信の手段であるメッセージによって直接スレッドを起動するのが効果的であり，そのためメッセージの受信処理や生成・送出などの通信パイプラインが，ハードウェアの支援により効率良く動作することが重要であることを示した．そして，これを実現するスレッド処理機構を組み込んだプロセッサアーキテクチャを提案し，これに基づいて現在我々が開発を進めている超並列計算機向け要素プロセッサ LSI，RICA-1 の内部構成とその基本性能を紹介した．

RICA-1 はランダムゲート数 20 万ゲート，内蔵メモリ 13KB のゲートアレイによって実現される，超並列計算機向けマルチスレッド型プロセッサである．RISC アーキテクチャに基づいた演算処理パイプラインと，大域並列処理を容易化する通信パイプラインとが高度に融合し，また 3 組のレジスタセットを用意することで，高効率のマルチスレッド処理を実現している．

今後は，レジスタ転送レベルのシミュレータを用い，複数プロセッサの動的な挙動について詳細な評価を行うとともに，実機の開発を進めて，実機上での評価を行う予定である．さらにこのプロセッサを 1024 台実装する，超並列計算機 RWC-1 の開発を進めていく予定である．

謝辞 本研究を遂行するにあたり，有益なご指導，ご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．

参考文献

1) Shimada, T., Hiraki, K., et al.: Evaluation of a Prototype Data Flow Processor of the SIGMA-1 for Scientific Computation, *Proc. ISCA'86*, pp.226–234 (1986).
2) Iannucci, R.A.: Toward a Dataflow/Von Neumann Hybrid Architecture, *Proc. ISCA'88*, pp.131–140 (1988).
3) Papadopoulos, G.M. and Culler, D.E.: Monsoon: An Explicit Token-Store Architecture, *Proc. ISCA'90*, pp.82–91 (1990).
4) 坂井，平木，山口，児玉，弓場：データ駆動計算機のアーキテクチャ最適化に関する考察，情報処理学会論文誌，Vol.30, No.12, pp.1562–1572 (1989).
5) Nikhil, R.S., Papadopoulos, G.M. and Arvind: *T: A Multithreaded Massively Parallel Architecture, *Proc. ISCA'92*, pp.156–167 (1992).
6) Agarwal, A., et al.: The MIT Alewife Machine: Architecture and Performance, *Proc. ISCA'95*, pp.2–13 (1995).
7) 横田，松岡，岡本，廣野，坂井：超並列向け相互結合網 MDCE の提案と評価，情報処理学会論文誌，Vol.36, No.7, pp.1600–1609 (1995).
8) Eicken, T. von, Culler, D.E., Goldstein, S.C. and Schauser, K.E.: Active Messages: A Mechanism for Integrated Communication and Computation, *Proc. ISCA'92*, pp.256–266 (1992).
9) Sakai, S., Kodama, Y., et al.: Reduced Interprocessor-Communication Architecture and its Implementation on EM-4, *Parallel Computing*, Vol.21, No.5, pp.753–769 (1995).
10) Sakai, S., et al.: RWC-1 Massively Parallel Architecture, *Proc. HPCC'94*, pp.33–38 (1994).
11) Sakai, S., Okamoto, K., Kodama, Y. and Sato, M.: Reduced Interprocessor-Communication Architecture for Supporting Programming Models, *Proc. Programming Models for Massively Parallel Computers*, pp.134–143 (1993).

(平成 8 年 2 月 1 日受付)
(平成 8 年 9 月 12 日採録)

岡本 一晃（正会員）

1962年生．1986年慶應義塾大学理工学部電気工学科卒業．同年三洋電機（株）に入社．主にデータ駆動計算機を中心とする並列処理アーキテクチャの研究に従事．1992年10月より（技組）新情報処理開発機構に出向，超並列アーキテクチャの研究に従事．主任研究員．ICCD Outstanding Paper Award（1995年）受賞．

松岡 浩司（正会員）

1961年生．1984年東京工業大学工学部電気電子工学科卒業．1986年同大学院理工学研究科電子物理工学専攻課程修了．同年日本電気（株）に入社．1992年10月より（技組）新情報処理開発機構に出向，主任研究員．現在，計算機システム全般，特にプロセッサアーキテクチャの研究に従事．ICCD Outstanding Paper Award（1995年）受賞．

廣野 英雄

1968年生．1991年筑波大学第三学群基礎工学類卒業．同年三洋電機(株)に入社．1992年より（技組）新情報処理開発機構に出向中．計算機アーキテクチャの研究に従事．ICCD Outstanding Paper Award（1995年）受賞．電子情報通信学会会員．

横田 隆史（正会員）

1960年生．1983年慶應義塾大学工学部電気工学科卒業．1985年同大学院電気工学専攻修士課程修了．同年三菱電機（株）に入社，知識処理向けアーキテクチャおよび並列アーキテクチャの研究に従事．1993年12月より（技組）新情報処理開発機構に出向．超並列アーキテクチャの研究に従事．ICCD Outstanding Paper Award（1995年）受賞．電子情報通信学会会員．

坂井 修一（正会員）

1958年生．1981年東京大学理学部情報科学科卒業．1986年同大学院情報工学専門課程修了．工学博士．同年，電子技術総合研究所入所．1991年4月より1年間米国MIT招聘研究員．1993年3月より1996年2月までRWC超並列アーキテクチャ研究室室長．1996年10月より筑波大学助教授（電子情報工学系）．計算機システム一般，特にアーキテクチャ，並列処理，スケジューリング問題などの研究に従事．情報処理学会研究賞（1989年），同論文賞（1990年），日本IBM科学賞（1991年），元岡記念賞（1991年），市村学術賞（1995年），ICCD Outstanding Paper Award（1995年）各受賞．