計算機アーキテクチャ　113−28
（１９９５．　８．　２５）

超並列計算機の要素プロセッサ向き メッセージ駆動スレッドアーキテクチャ

柴田 幸茂，　平田 博章，　新実 治男，　柴山 潔

京都工芸繊維大学　工芸学部　電子情報工学科

あらまし

分散メモリ型超並列計算機を指向した要素プロセッサアーキテクチャを提案する。

本アーキテクチャは、超並列処理において顕在化する同期および通信のオーバヘッドを削減・隠蔽することに重点を置いている。メッセージ待ち合わせをハードウェアで高速処理するために新たにQ-Registerと呼ぶ双方向キュー機能を備えたレジスタを設け、さらに高速コンテキスト切換え機能を備えて、演算とメッセージ処理とのオーバラップ実行を図る。

和文キーワード　スレッドレベル並列処理、メッセージ駆動、遠隔実行メッセージ

Message Driven Thread Architecture for Massively Parallel Computers

Yukishige Shibata, Hiroaki Hirata, Haruo Niimi, Kiyoshi Shibayama

Dept. of Electronics and Information Science,
Faculty of Engineering and Design,
Kyoto Institute of Technology

Abstract

This paper presents a Message Driven Thread (MDT) architecture for a processor element used in a massively parallel computer.

This architecture aims at minimizing inter-processor synchronization and communication overheads. The processor implementing the MDT architecture is facilitated with a fast context-switching mechanism and a Q-Register as a biderectional queue, allowing communication to overlap with computation .

英文 key words　Thread-Oriented Parallel Processing, Message Driven, Remote Execution Message

1 はじめに

近年、WS クラスタやスーパコンピュータを仮想的に並列処理環境化する PVM や、統一的なメッセージ交換インタフェースである MPI などを用いた並列処理が盛んに行なわれており、分散メモリ型超並列計算機への要望が高まってきている。

超並列計算機を実現するには、粒度とメッセージ交換に伴うオーバヘッドとのトレードオフの設定を適切に行ない、それに応じたアーキテクチャを開発する必要がある。

本稿では、分散メモリ型超並列計算機の要素プロセッサに着目し、メッセージパッシングモデルに適合した要素プロセッサ (Processor Element; PE) アーキテクチャについて述べる。

本稿は次のような構成をとる。第 2章で分散メモリ型超並列計算機に対する要件と、その解決方法を論じる。第 3章では、超並列処理に適した粒度について述べる。第 4章では、第 2章と第 3章で示された要件を満たすべく設計されたメッセージ駆動スレッド (Message Driven Thread; MDT) アーキテクチャについて述べる。最後に第 5章で、まとめと残された課題について述べる。

2 超並列計算機への課題

2.1 オーバヘッドの削減

分散メモリ型並列計算機では、プロセッサ間の通信をすべてメッセージ交換によって実現する。このために、分散メモリ型並列計算機では、メッセージの送受信処理に大きなオーバヘッドが存在すると十分な性能を発揮することができない。特に超並列処理では、並列処理単位の粒度が細かくなって通信/同期操作の発生する頻度が高くなる傾向があり、従って、分散メモリ型超並列計算機を開発する上では、メッセージ処理に関するオーバヘッドを削減することが性能向上のための大きな課題となる。

このような問題を解決する技術として、

1. コンテキスト切換えの高速化 [1]
2. 通信/同期処理と演算処理のオーバラップ実行 [2]
3. PE とネットワークインタフェース間のメッセージ処理に要するオーバヘッドの削減 [3]
4. 相互結合網の高速化によるレイテンシの削減 [4]

などが知られている。このうち、要素プロセッサに直接関連するのは、1. 2. 3. である。この点で、市販のマイクロプロセッサは逐次演算処理性能のみを目的にチューニングされており、上に挙げた機能を持っていない。このことは、超並列計算機には演算・通信・同期の各能力をバランスさせた専用の PE を開発する必要があることを示している。

2.2 マルチユーザ環境

超並列計算機は、システムコストの高さ故に貴重な計算機資源である。計算機間を接続するネットワークが整備されつつある現在では、超並列計算機を多数のユーザがアクセスできる環境に置くことが計算機資源の有効利用につながる。すなわち、超並列計算機にはマルチユーザ/マルチタスク環境が備わっていることが必須となる。

マルチユーザ環境の下では、多数のユーザによって科学技術計算やデータベース処理、記号処理などの多種多様なタスクが実行される。超並列計算機では、これら多種多様な粒質および異なる粒度を持つ複数のタスクを高速に実行できる性能が要求されることになる。このためには、メッセージ処理の高速化と並んで、さまざまな粒質[1]および異なる粒度を持つ複数のタスクを高速に実行できる性能が要求されることになる。このためには、メッセージ処理の高速化と並んで、さまざまな粒質に対応できる演算能力と効率の良い負荷分散を実現できなければならない。

[1] 並列処理の個々の単位で実行される処理の持つ性質。

3 スレッドレベル並列処理

超並列計算機の性能を引き出すには、その上で実行するタスクに高い並列性が本質的に備わっていることが要請される。すなわち、従来からある SPMD(Single Program Multiple Data streams) のような大きな粒度ではなく、もっと細かな粒度の処理機能を多数用意しなければならない。

処理粒度を機械命令レベルにとると計算機の並列度は最大になるが、あまりに細粒度であるために、1) 通信・同期に要するオーバヘッドの顕在化と、2) ハードウェア量の増大、を招く。そこで、スレッドを処理粒度の基本としたスレッドレベル並列処理を導入する。ここで、スレッドとは、比較的小さな一連の命令列を実行する処理単位である。このようなスレッドを超並列計算機内に分散配置し、スレッドが互いにメッセージを交換し合いながらタスクの実行を進めていく。

このスレッドレベル並列処理では次のような利点が得られる。

1. 同一のデータにアクセスするスレッドを同一の PE に配置することで、データ参照の局所性を活かせる。
2. スレッドのサイズが小さいので、スレッドを単位としたマイグレーションのコストが小さい。

他方で、

1. 単一の PE に多数のスレッドが配置されるとスレッド切り換えのコストが無視できなくなる。
2. 与えられた問題のスレッドへの分割を最適にするのが困難であり、プログラマやコンパイラへの負担が非常に大きい。
3. 最適な負荷分散を行なうことが困難である。

のような問題も生じる。

そこで、問題点の 1. については高速コンテキスト切り換え機構をハードウェアでサポートすることによって解決する。また、問題点の 2. については、並列オブジェクト指向言語などの分散メモリ型並列計算機に適合した並列処理記述言語によってプログラミングすることで、コンパイラにかかる負担をある程度緩和することができる [5]。

4 メッセージ駆動スレッドアーキテクチャ

本稿で提案するメッセージ駆動スレッド (MDT) アーキテクチャは、分散メモリ型超並列計算機用の PE として、特にスレッドレベル並列処理に焦点を合わせた PE アーキテクチャである。その特徴として、MDT アーキテクチャでは、核となる VLIW プロセッサ (4.4節参照) に、多重レジスタセット、双方向キューレジスタ (Q-Register)、メッセージ送信/受信ユニット、スレッド管理ユニットを加えている。(図 1参照)

これらの専用機構を設けることによって、

1. コンテキスト切換えのオーバヘッドの削減
2. メッセージ受信処理の高速化
3. 遠隔実行メッセージのサポート

を実現する。

4.1 高速コンテキスト切換えの実現

4.1.1 スレッド管理ユニット

スレッド管理ユニットには、PE に配置されたスレッドに関する情報を収めたスレッドテーブル (Thread Table; T-Table) と、スレッドのスケジューリングを行なうスレッド発行ユニット (Thread Dispatch Unit) から構成される。

T-Table は、スレッドの識別子である TID(Thread ID) と GID(thread Group ID) をキーとした連想記憶で構成されている。スレッドテーブルには、以下に挙げる情報が記憶される。

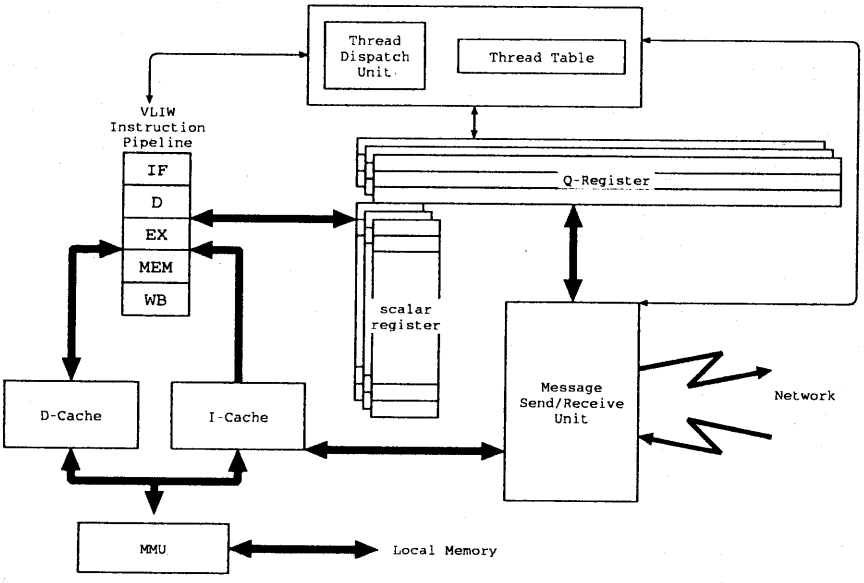


図 1: MDT アーキテクチャ

GID スレッドグループ ID。スレッドグループに対する識別子。計算機内にあるスレッドグループに対して一意に与えられる識別子。

TID スレッド ID。一つのスレッドグループに属する個々のスレッドに一意に割り当てられる。GID と TID によって計算機内のスレッドを一意に決定できる。

Status スレッドの状態を収めている。run, ready, wait, done の 4 状態がある。

RegNo スレッドが使用するレジスタセットの番号。

SyncCond スレッドの同期条件が格納される。格納される情報は、メッセージ待ち合わせ条件とスレッド間同期情報である。

SyncMask SyncCond に対するマスク。

Pri スレッドの優先度。

また、TDU は、

1. メッセージを受信した時、
2. 実行中のスレッドの状態が、wait、または done 状態に変化した場合

にスレッドの切換えを行なう。これらのイベントが起こった場合、SynCond フィールドと SyncMask フィールド、および Q-Register にあるフラグ (Q-flag) が参照される。SyncCond, SyncMask, Q-flag によって表される条件式が成立した場合には、Status フィールドを ready 状態へ変更する。。この結果、メッセージ待ち合わせの条件が成立したスレッドがスケジューリングの対象に上がる。(図 2参照)

スレッドのスケジューリングには、Pri フィールドの値が参考にされる。もし、現在実行中のスレッドよりも優先度が高いスレッドが ready 状態になるとプリエンプションを起こし、優先度の高いスレッドの実行に切換える。同一の優先度を持つ複数のスレッドが、ready 状態になった場合は、LRE(Least Recently Executed) アルゴリズムによって次実行スレッドを決定

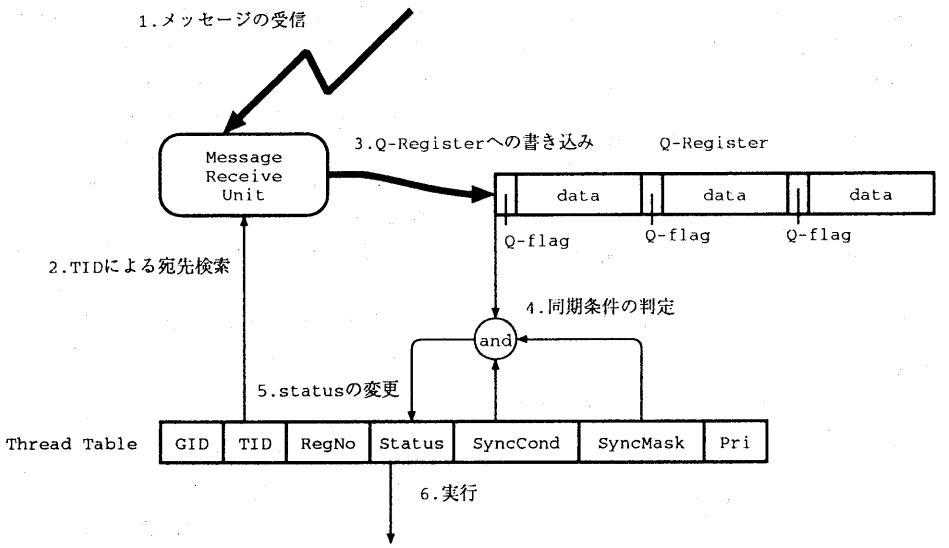


図 2: スレッドの起動

する。

4.1.2 多重レジスタセット

スレッドテーブル上の 1 スレッドに対して 1 つのレジスタセットが割り当てられる。

各レジスタセットは、複数本のスカラレジスタと 4.2.1節で述べる Q-Register から構成される。

4.2 メッセージインタフェース

4.2.1 Q-Register

受信したメッセージをメモリに格納することは、1) 受信メッセージのメモリ上のバッファ領域への格納、2)OS によるバッファからユーザ領域へのメッセージ転送 (読み出し、書き込み各 1 回)、3) 演算ユニットによるメモリからの読み込み、の 4 回のメモリアクセスを行なうことになり非常にオーバヘッドが大きい。またメッセージ格納の間、演算ユニットによるメモリへのアクセスが待たされる。

そこで、メッセージを直接レジスタに格納することによって、メモリアクセスを無くし、メッセージ処理のオーバヘッドを削減する。

しかし、メッセージによるデータ転送で、は転送効率を上げるために、1 レジスタのビット長よりもメッセージ長の方が長いのが普通である。このために、1) 転送データによってレジスタが喰い潰されたり、2) 転送データ長がレジスタのサイズに制限される、などの問題が生じる。

これらの問題を解決するために Q-Register を導入する。Q-Register は論理的には双方向キューを実現する。Q-Register の各エントリに対してデータの存在を示すフラグ (Q-flag) を設ける。この Q-Register は、演算ユニットとメッセージ送信/受信ユニットの間に置かれており、Q-Structure[6] と同等の双方向キューを構成している。

Q-Register は、両ユニット間のデータ転送バッファの役割を持つと同時に、演算時にベクトルレジスタとして用いることで、ソフトウェアパイプライニングを支援する働きも持つ。

Q-Register 中の先頭要素の Q-flag は常にスレッド管理ユニットによって監視されており、メッセージ待ち合わせ処理に用いられる。先

頭要素のQ-flagが立っている場合には、メッセージの待ち合わせに成功したことになる。

Q-Register導入による利点は、

1. 演算ユニットとメッセージ送信/受信ユニットとのオーバラップ動作が可能
2. メッセージ待ち合わせ処理のハードウェア化によるオーバヘッドの軽減
3. ソフトウェアパイプライニングの支援

である。

一方、欠点はハードウェア量の増大を招くことである。

4.2.2 メッセージ受信ユニット

メッセージには、D-Message(Data Message)と、E-Message(Executable Message)の2つの種類がある。D-Messageはヘッダ部とデータ部から構成されており、E-Messageはさらにデータ部に続いて演算ユニットで実行される命令列を含んでいる。

メッセージ受信ユニットがD-Messageを受信すると、メッセージのヘッダにある宛先のTIDをキーとして、T-Tableから宛先スレッドに割り当てられているレジスタセットを調べる。そして、ヘッダに続くデータをそのQ-Registerに書き込む。Q-Registerへのデータの書き込みに際しては、Q-flagを立てる。

E-Messageについては、4.3節で述べる。

4.2.3 メッセージ受信ユニット

D-Message送信に関しては、スレッドがメッセージ送信命令を実行することで、メッセージ送信ユニットを起動する。メッセージ送信ユニットは、Q-Registerからデータを読み出してメッセージを組み立てる。メッセージの宛先はレジスタあるいは命令の即値によって指定される。

4.3 遠隔実行メッセージ

E-Messageには命令列が格納されており、そのメッセージに対して受信側で行なうべき処理内容を指定することができる。E-Messageによってストライドデータアクセスに代表される構造化データへの遠隔メモリアクセスが特殊なハードウェアを付加することなしに実現される。また、スレッドをE-Messageとして移動させることにより、動的負荷分散[2]を柔軟に行なうことができる。

E-Messageの処理のために、各PEごとにスレッドテーブルの1エントリとレジスタセットがE-Messageのために予約されており、PEで受信されたE-Messageは、すべて予約されたスレッドテーブルのエントリとレジスタセットを使用する。これによって、スレッド受信側でのスレッド生成のオーバヘッドを生じることなく実行を開始できる。

以下は、E-Messageの処理の手順である。

1. データをQ-Registerに格納。
2. メッセージ受信ユニットが命令列を命令キャッシュに格納。
3. 送られてきた命令を実行するための新しいスレッドを生成。
4. 3.で生成されたスレッドの実行を開始。

なお、命令キャッシュには、送られてきた命令の入ったラインがフラッシュされるのを防ぐ機構を備える。

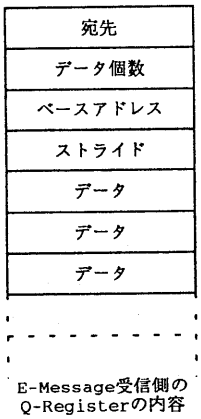
図3に、遠隔メモリ上にあるストライドデータにアクセスするために、E-Messageとして送信する命令列の例を挙げる。なお、この例では簡単のために1オペレーション/1命令として示す。

4.4 VLIWプロセッサ

4.4.1 VLIWアーキテクチャの採用

命令レベル並列処理として、ハードウェアによって動的に並列性の抽出および依存性を解決するスーパスカラ方式と、コンパイラによって静的にコードスケジューリングを行なうVLIW方式とがある。MDTアーキテクチャ

[2] E-Messageによる負荷分散の粒度の大きさは、すべてのデータがQ-Registerに格納でき、かつすべての命令がキャッシュの少数のラインに収まる程度のものとする。




```
; メッセージ宛先を pop
    pop   R0, Q0
; アクセスするデータ個数を pop
    pop   R1, Q0
; R1 を R2 をへコピー
    add   R2, R1, #0
; ベースアドレスを pop
    pop   R3, Q0
; ストライドを pop
    pop   R4, Q0
; メモリアクセス
loop:load  R5, mem(R3)
; ワード数をデクリメント
    sub   R1, R1, #1
; Q1 へデータを書き込み
    push  Q1, R5
; 次アドレスを生成
    add   R3, R3, R4
; ループ判定
;(R1 の値が 0 ならループ終了)
    bnz   R1, %loop
; データを返送
    send-D-message  R0, Q1
```

図 3: 遠隔メモリへのストライドデータアクセス (遠隔側で実行するコード)

では、以下のような理由によって VLIW 方式による命令レベル並列処理を選択した。

- スーパスカラ方式では依存性解消のためのハードウェアコストが大きい。
- 必要なコードスケジューリングの範囲がスレッドという比較的小さい単位であり、コンパイラでも最適に近いスケジューリングが可能である。

4.4.2 命令セットの拡張

MDT アーキテクチャには、整数演算ユニットや浮動小数点演算ユニット、ロードストアユニットに加えて、スレッド管理ユニット、Q-Register、メッセージ送受信ユニットが存在する。VLIW にはこれら新しいユニットを操作するフィールドを設ける。

新たに加えられるオペレーションを以下に挙げる。

(1) スレッド制御命令

スレッド管理ユニットに対するオペレーション。

make_thread スレッドを新たに生成する。具体的にはスレッドをスレッドテーブルへ登録する。

make_threadgroup 新たなスレッドグループを生成する。

delete_thread スレッドを消去する。スレッドテーブルからエントリを削除する。

suspend_thread スレッドの実行を強制的に中断し、他スレッドへ実行を切換える。

(2) Q-Register 操作命令

Q-Register に対するオペレーション。

push_Q-register Q-Register に 1 ワード分のデータを書き込むとともに、Q-flag を立てる。

pop_Q-register Q-flag が立っていれば Q-Register から 1 ワード分のデータを読み出す。Q-flag が立っていなければ例外を発生する。

clear_Q-flag Q-Register の Q-flag をクリアする。

(3) メッセージ生成命令

メッセージ送信ユニットに対するオペレーション。

send_D-message Q-Register の内容をデータとして、スカラレジスタで指定されたスレッド宛のメッセージを送る。

send_E-message スカラレジスタで指定されたスレッドへ E-Message を送る。命令列は命令キャッシュから読み出す。

5 まとめ

本稿では、1) スレッド管理機構、2)Q-Register を用いたプロセッサ間通信インタフェース、3) 遠隔実行メッセージ、を特徴とするメッセージ駆動スレッドアーキテクチャの概要について述べた。

今後の課題としては、ハードウェア構成を詳細に決定することが挙げられる。特に、PE の性能やハードウェア量に大きな影響を与えるキューレジスタの深さや、スレッドテーブルのサイズ、命令セットなどをパラメータとしたシミュレーションを行ない、最適な組合せを見い出す必要がある。

参考文献

[1] A. Agarwal, et al. : "The MIT Alewife Machine : A Large Scale Distributed-Memory Multiprocessor", MIT / LCS / TM-454.b, 1991.

[2] 堀江 健志, 他 : "メッセージ通信の分散メモリ型並列計算機性能への影響 – 通信と演算のオーバラップと直接メッセージ受信の効果–", JSPP'93, pp.23–30, 1993.

[3] W. J. Dally, et al. : "The J-Machine: a Fine-Grain Concurrent Computer", Proc. of IFIP World Computer Congress, pp.1147–1153,1989.

[4] C. Wu and T.Feng : "Tutorial: Interconnection Networks for Parallel and Distributed Systems", IEEE, Computer, Society, 1984.

[5] T. Yoshinaga, and T.Baba : "A Parallel Object-Oriented Language A-NETL and Its Programming Environment", Proc. COMPSAC '91, pp.189–196, 1991.

[6] 佐藤 久三, 他 : "並列計算機 EM-4 における分散データ構造を用いたマルチスレッドプログラミング", 情報処理学会研究報告, 92-ARC-92-7, 1992.